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.
$\boldsymbol{x}^\intercal, \boldsymbol{X}^\intercal$	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 $\boldsymbol{f}$).
$\nabla\nabla$	The Hessian.
$\sim$	Distributed according to, example $x \sim \mathcal{N}(0,1)$
$\mathbf{0}$ 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. The identity matrix in the case $n = m$.

$\boldsymbol{A}_{(\cdot,\cdot)}$	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 $\mathbb{R}^{3 \times (j'-j+1)}$ submatrix 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.
$\text{cholesky}\left(\boldsymbol{A}\right)$	A function to compute the Cholesky decomposition, $\boldsymbol{L}$, of the matrix $\boldsymbol{A}$, where $\boldsymbol{L}\boldsymbol{L}^{\intercal} = \boldsymbol{A}$.
$\text{cov}\left(\boldsymbol{f}\right)$	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, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^{n}$.
$\text{diag}\left(\boldsymbol{w}\right)$	Vector argument, a diagonal matrix containing the elements of vector $\boldsymbol{w}$.
$\text{diag}\left(\boldsymbol{W}\right)$	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.
$\text{l. s}\{\boldsymbol{x}_i\}_{i=1}^n$	The linear-span of $\{\boldsymbol{x}_i\}_{i=1}^n$, that is, $\{\sum_{k=1}^n \lambda_k \boldsymbol{x}_k \mid \lambda_k \in \mathbb{R}\}$.
$\boldsymbol{K}_{\boldsymbol{W}\boldsymbol{W}'}$	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 $\boldsymbol{X} = \boldsymbol{A}^{-1}\boldsymbol{B}$ for 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 is a member of $\mathcal{O}(g)$ 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.
$\text{tr}(\boldsymbol{A})$	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 20^{th} 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 a novel 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 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 sensing tasks [SD22] the team 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

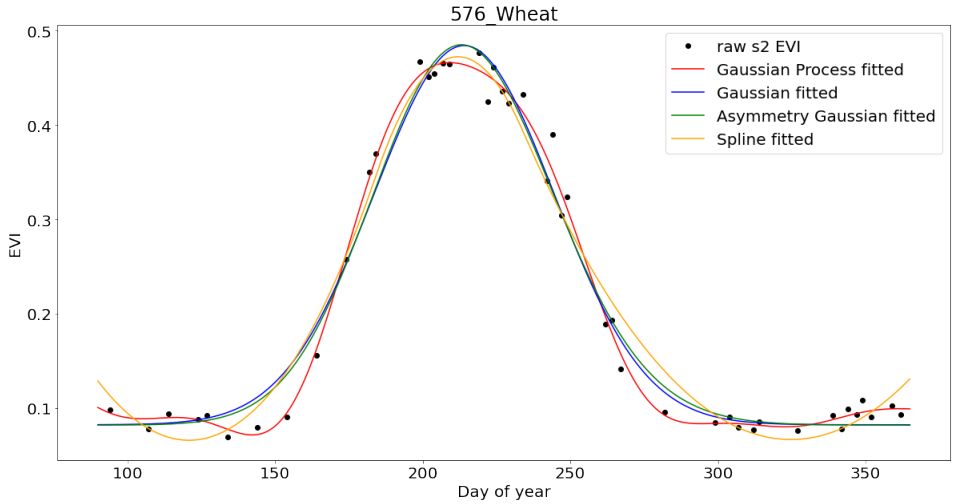


FIGURE 1. Potgieter's team found that GPs where superior in terms of predicting a phenological timeline for a number of common seasonal crops over other parameteric models.

collecting new 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 **??** 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 1 and 2 will cover techniques for approximating a large matrix used with GPs that provides information on how similar each observation is to one other. Chapter 3 then gives alternative methods for solving linear systems, an essential component required for the GP algorithm to work.

1. The Nystrom Method

In Chapter **??** we saw that GP regression and classification relied on a Gram matrix (see **??**) 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 on account of the $\mathcal{O}\left(n^2\right)$ kernel evaluations. Even before the rise of ML, 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 burden to build exactly. These methods are based on the idea of capturing the columns space of the matrix that best describes the the action of $\boldsymbol{A}$ as an operator. Mahoney provides an enlightened 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 Gram matrices.

1.1. The Nystrom Method.

Attempting to compute an entire kernel matrix can be a computational headache prompting an investigation of estimative alternatives. The approximation techniques studied in this chapter have been spurred on by the John-Lindenstrauss lemma stated in Lemma 1.

Lemma 1 (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 $\ell = \mathcal{O}(\ln(n))$ 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 $\boldsymbol{A} \in \mathbb{R}^{n\times m}$ where $\boldsymbol{QQ}^*$, in some sense, projects onto a rank-k subspace of $\boldsymbol{A}$'s column space. This is because if $\boldsymbol{QQ}^*$ closely 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 in a more short hand notation as

$$\boldsymbol{A} \simeq \boldsymbol{QQ}^*\boldsymbol{A}. \tag{1}$$

This is commonly called the *fixed precision approximation problem*. To simplify algorithmic development, a value of k is specified in advance (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 $\boldsymbol{A} \simeq \boldsymbol{AQQ}^*$ meaning

$$\boldsymbol{A} \simeq \boldsymbol{QQ}^* (\boldsymbol{A}) \simeq \boldsymbol{QQ}^* \boldsymbol{AQQ}^*. \tag{2}$$

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 (2)

$$\begin{aligned} \boldsymbol{A} &\simeq \boldsymbol{Q} (\boldsymbol{Q}^* \boldsymbol{AQ}) \boldsymbol{Q}^* \\ &= \boldsymbol{Q} (\boldsymbol{Q}^* \boldsymbol{AQ}) (\boldsymbol{Q}^* \boldsymbol{AQ})^\dagger (\boldsymbol{Q}^* \boldsymbol{AQ}) \boldsymbol{Q}^* \\ &\simeq (\boldsymbol{AQ}) (\boldsymbol{Q}^* \boldsymbol{AQ})^\dagger (\boldsymbol{Q}^* \boldsymbol{A}) . \end{aligned} \tag{3}$$

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 1.

Algorithm 1: General Nystrom Framework

input : A positive semi-definite matrix $\boldsymbol{A} \in \mathbb{R}^{n \times m}$, a matrix $\boldsymbol{Q} \in n \times k$ that satisfies (1).

output: A rank k approximation $\overline{\boldsymbol{A}} \simeq \boldsymbol{A}$.

1 $\boldsymbol{C} = \boldsymbol{AQ}$

2 $\boldsymbol{W} = \boldsymbol{Q}^* \boldsymbol{C}$

3 **return** $\boldsymbol{CW}^\dagger \boldsymbol{C}^*$

However, Algorithm 1 assumes that $\boldsymbol{Q}$ has already been computed. Naturally, the next question is then how to efficiently construct a suitable matrix $\boldsymbol{Q}$ that satisfies (1)? We can do this through a very popular column sampling technique ubiquitous in numerical linear algebra literature. This technique has been driven by Theorem 2.

Theorem 2. *Every $\boldsymbol{A} \in \mathbb{R}^{n \times m}$ matrix contains a $k-$column submatrix $\boldsymbol{C}$ for which*

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

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

Before we delve further into this column sampling Nystrom method, we must first cover the random matrix multiplication algorithm which serves as a backbone for this technique. Therefore, 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

$$\boldsymbol{A} = \sum_{i=1}^{I} \boldsymbol{A}_i. \tag{4}$$

The basic idea is to consider a Monte-Carlo approximation of (4) that randomly selects $\boldsymbol{A}_i$ according to the distribution $\{p_i\}_{i=1}^I$ to give an estimate

$$\boldsymbol{A} \simeq \frac{1}{c}\sum_{t=1}^{c} p_{t_i}^{-1}\boldsymbol{A}_{t_i} \tag{5}$$

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 $\boldsymbol{B} \in \mathbb{R}^{I\times m}$. 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 (4) to the corresponding rank-1 outter product summand $\boldsymbol{A}_{(:,i)}\boldsymbol{B}_{(i,:)}$. This justifies the random matrix multiplication algorithm seen in Algorithm 2 [PDaMWM17, page 16].

Algorithm 2: Random Matrix Multiplication

input : $\boldsymbol{A} \in \mathbb{R}^{n\times I}$ and $\boldsymbol{B} \in \mathbb{R}^{I\times m}$, the number of samples $1 \le c \le I$ and a probability distribution over I, $\{p_i\}_{i=1}^I$.

output: Matricies $\boldsymbol{C} \in \mathbb{R}^{n\times c}$ and $\boldsymbol{R} \in \mathbb{R}^{c\times m}$ such that $\boldsymbol{CR} \simeq \boldsymbol{AB}$.

for $t = 1, \dots, c$ **do**
  Pick $i_t \in \{1, \dots, I\}$ 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 2, 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

$$\boldsymbol{CR} = \boldsymbol{ASS}^\intercal \boldsymbol{B} \simeq \boldsymbol{AB}. \tag{6}$$

Notice that $\boldsymbol{S}$ is generally a very sparse matrix and therefore is generally not constructed explicitly and instead the matrix products $\boldsymbol{AS}$ and $\boldsymbol{S}^\intercal\boldsymbol{B}$ are done through row and column rescaling [PDaMWM17, page 17]. Lemma 3 provides some bounds on $\boldsymbol{CR}$ as an estimate for $\boldsymbol{AB}$.

Lemma 3. *Let $\boldsymbol{C}$ and $\boldsymbol{R}$ be constructed as described in Algorithm 2, 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 $t = 1, \ldots, c$ 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,

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

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 $\mathbb{1}_{n\times n}$. (6) 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 2 provides a sketching matrix $\boldsymbol{S}$ that satisfies the properties of $\boldsymbol{Q}$ from (1) meaning that $\boldsymbol{S}$ can be used in place of $\boldsymbol{Q}$ within the Nystrom estimate from (3). These ideas are used together in Algorithm 3 that uses the column sampling technique from

Algorithm 2 together with the general Nystrom framework (Algorithm 1) lends a new column sampling Nystrom method to approximate a Gram matrix for a provided dataset and probability distribution [PDaMWM05, AGaMWM13].

Algorithm 3: 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, $\{p_i\}_{i=1}^n$.

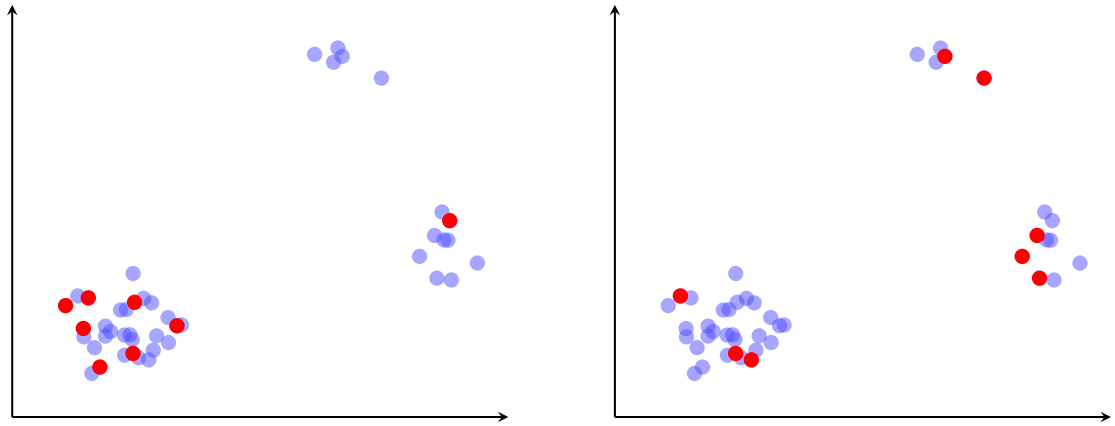
output: An approximation of the Gram matrix corresponding to $\boldsymbol{X}$, that is $\overline{\boldsymbol{K}} \simeq \boldsymbol{K}$ where $\boldsymbol{K}_{ij} = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$.

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**
    $\boldsymbol{K}_{(:,i)} = [k(\boldsymbol{x}_1, \boldsymbol{x}_i), \ldots, k(\boldsymbol{x}_n, \boldsymbol{x}_i)]^\intercal$
    $\boldsymbol{C}_{(:,i)} = \boldsymbol{K}_{(:,i)} / \sqrt{cp_i}$
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 distribution to select the columns. As seen in Lemma 3 any probability distribution will provide an unbiased estimate. However, 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 although it should be cautioned that this is seldom a good idea since uniform sampling tends to over sample landmarks from one large cluster while under sampling or even missing entire 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 2.

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 better error bounds within the Nystrom framework [PDaMWM05, AGaMWM13, CMaCM17, PDe11, MBCaCMaCM15, Kum09]. A few of the more common distributions will be discussed in the coming sections.

1.2. **Column Probabilities.** Recall that the Nystrom method from Algorithm 3 is largely dependent on the random matrix multiplication algorithm (see Algorithm 2) to produce a suitable sketching matrix. Moreover, improvements in the sketching matrix produced by the random matrix multiplication algorithm are 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



(A) (B)

FIGURE 2. Employing uniform sampling in the column sampling Nystrom estimate can lead to oversampling from denser parts of the data set (Panel (A)). Instead data dependent probability densities are commonly used to better cover the relevant data (Panel (B)). Example taken from [CMaCM17, page 4].

of the form

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

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 4 [PDaMWM05, page 2158].

Theorem 4. *Given $\boldsymbol{A} \in \mathbb{R}^{n\times I}$, $1 \leq c \leq I$ and the probability distribution $\{p_i\}_{i=1}^{I}$ described in Section 1.2. Construct $\boldsymbol{C}$ using Algorithm 2, then*

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

[PDaMWM05, page 2158].

To show Theorem 4, we can actually prove something a little more general.

Lemma 5. *Given $\boldsymbol{A} \in \mathbb{R}^{n\times I}$, $\boldsymbol{B} \in \mathbb{R}^{I\times m}$, $1 \leq c \leq I$ and the probability distribution $\{p_i\}_{i=1}^{I}$ as follows*

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

Construct $\boldsymbol{C}$ using Algorithm 2, 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 $\mathbb{E}\left[\|\boldsymbol{AB}-\boldsymbol{CR}\|_F\right]$ *among all possible sampling probabilites* [Dri06, pages 9-12].

Proof. First note that

$$\mathbb{E}\left[\|\boldsymbol{AB}-\boldsymbol{CR}\|_F^2\right]=\sum_{k=1}^{n}\sum_{j=1}^{m}\mathbb{E}\left[(\boldsymbol{AB}-\boldsymbol{CR})_{kj}^2\right]=\sum_{k=1}^{n}\sum_{j=1}^{m}\mathbb{V}\left[(\boldsymbol{CR})_{kj}\right].$$

Thus from Lemma 3, it follows that

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

Substituting in a probability of

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

yields

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

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

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

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

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

The minimum is then

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

Thus

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

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

1.3. **Leverage Scores.**

1.3.1. *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 are not any $\boldsymbol{x} \in \mathbb{R}^m$ for which $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. Instead, alternative criteria must be employed 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]. The optimal value for $\boldsymbol{x}$ can be solved as $\boldsymbol{x}_{opt} = (\boldsymbol{A}^\intercal\boldsymbol{A})^{-1}\boldsymbol{A}^\intercal\boldsymbol{b}$. 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 a strong influence over 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 remedied through normalisation, in this case dividing each statistical leverage score by $\mathrm{tr}\left(\boldsymbol{H}\right)$. The idea behind using statistical leverage scores as a probability distribution in the Nystrom method is that they help prioritize selection of columns that are more linearly independent from other columns so that the range of our approximate better aligns with the range of our original $\boldsymbol{A}$.

1.3.2. *Rank$-k$ Statistical Leverage Scores.* We can generalize this notion of statistical leverage scores to include lower rank approximations. Let $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal$ be the compact SVD of a $\boldsymbol{A}$ real $n \times m$ matrix. Setting $r = \min\{n, m\}$, the compact SVD can be partitioned as

$$\boldsymbol{U} = [\boldsymbol{U}_1, \boldsymbol{U}_2] \in \mathbb{R}^{n\times r}, \qquad \boldsymbol{\Sigma} = \begin{bmatrix}\boldsymbol{\Sigma}_1 & \\ & \boldsymbol{\Sigma}_2\end{bmatrix} \in \mathbb{R}^{r\times r}, \qquad \boldsymbol{V} = [\boldsymbol{V}_1, \boldsymbol{V}_2] \in \mathbb{R}^{m\times r}.$$

Here $\boldsymbol{U}_1$ contains the first $k \leq r$ 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 $\boldsymbol{A}_k = \boldsymbol{U}_1\boldsymbol{\Sigma}_1\boldsymbol{V}_1$ serves as 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

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

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

1.3.3. *Ridge Leverage Scores.* The low rank leverage scores we saw in Section 1.3.2 will not always be unique and can be sensitive to perturbations [MBCaCMaCM15, page 6]. Consequently these scores can vary drastically when $\boldsymbol{A}$ is modified slightly or when we only have access to partial information on the matrix. This undermines the the possibility of computing good quality low rank approximations from statistical leverage scores. This shortcoming is addressed in the next class of leverage score, that is, ridge leverage scores. Ridge leverage scores are similar to statistical leverage scores although a ridge regression term (hence the name) is added to the hat matrix with a 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 scores do not necessarily form a probability distribution) is bounded by $2k$, stated more formally in Lemma 6.

Lemma 6. *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].

Proof. Writing r_i^λ using the SVD of $\boldsymbol{A}$ where $\lambda = \frac{\|\boldsymbol{A}-\boldsymbol{A}_k\|_F^2}{k}$ gives

$$\begin{aligned} r_i^\lambda &= \boldsymbol{A}_{(i,:)} \left(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{U}^\intercal + \frac{\|\boldsymbol{A} - \boldsymbol{A}_k\|_F^2}{k}\boldsymbol{U}\boldsymbol{U}^\intercal\right)^{-1} \boldsymbol{A}_{(i,:)}^\intercal \\ &= \boldsymbol{A}_{(i,:)} \left(\boldsymbol{U}\overline{\boldsymbol{\Sigma}}^2\boldsymbol{U}^\intercal\right)^{-1} \boldsymbol{A}_{(i,:)}^\intercal \\ &= \boldsymbol{A}_{(i,:)} \left(\boldsymbol{U}\overline{\boldsymbol{\Sigma}}^{-2}\boldsymbol{U}^\intercal\right) \boldsymbol{A}_{(i,:)}^\intercal \end{aligned}$$

where $\overline{\boldsymbol{\Sigma}}_{ii}^2 = \sigma_i^2\left(\boldsymbol{A}\right) + \frac{\|\boldsymbol{A}-\boldsymbol{A}_k\|_F^2}{k}$. Then

$$\begin{aligned} \sum_{i=1}^n r_i^\lambda &= \operatorname{tr}\left(\boldsymbol{A}^\intercal\boldsymbol{U}\overline{\boldsymbol{\Sigma}}^{-2}\boldsymbol{U}^\intercal\boldsymbol{A}\right) \\ &= \operatorname{tr}\left(\boldsymbol{V}\boldsymbol{\Sigma}\overline{\boldsymbol{\Sigma}}^{-2}\boldsymbol{\Sigma}\boldsymbol{V}^\intercal\right) \\ &= \operatorname{tr}\left(\boldsymbol{\Sigma}^2\overline{\boldsymbol{\Sigma}}^{-2}\right). \end{aligned}$$

Here we have

$$\left(\boldsymbol{\Sigma}^2\overline{\boldsymbol{\Sigma}}^{-2}\right)_{ii} = \frac{\sigma_i^2\left(\boldsymbol{A}\right)}{\sigma_i^2\left(\boldsymbol{A}\right) + \frac{\|\boldsymbol{A}-\boldsymbol{A}_k\|_F^2}{k}}.$$

For $i \leq k$ we simply upper bound this by 1, yielding

$$\operatorname{tr}\left(\boldsymbol{\Sigma}^2\overline{\boldsymbol{\Sigma}}^{-2}\right) = k + \sum_{i=k+1}^{n} \frac{\sigma_i^2(\boldsymbol{A})}{\sigma_i^2(\boldsymbol{A}) + \frac{\|\boldsymbol{A}-\boldsymbol{A}_k\|_F^2}{k}} \leq k + \sum_{i=k+1}^{n} \frac{\sigma_i^2(\boldsymbol{A})}{\frac{\|\boldsymbol{A}-\boldsymbol{A}_k\|_F^2}{k}} = k + \frac{\sum_{i=k+1}^{n}\sigma_i^2(\boldsymbol{A})}{\frac{\|\boldsymbol{A}-\boldsymbol{A}_k\|_F^2}{k}} \leq k + k.$$

□

From now on (unless otherwise stated) the regularization parameter seen in Section 1.3.3 will always be used for ridge leverage scores where the notation

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

is employed to show that the best rank$-k$ matrix is utilized 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]. Alaoui and Mahoney [Ala15] prove that ridge leverage scores provide theoretically better bounds over uniform sampling techniques when the number of sampled columns is proportional to $\operatorname{tr}(\boldsymbol{H}_\lambda)\cdot\ln(n)$ where $\boldsymbol{H}_\lambda$ is the hat matrix with added regularization, that is $\boldsymbol{H}_\lambda = \boldsymbol{A}\left(\boldsymbol{A}^\intercal\boldsymbol{A} + \lambda\mathbb{1}_{n\times n}\right)^{-1}\boldsymbol{A}^\intercal$. With the rising popularity of ridged leverage scores, a number of iterative methods have been devised (and continue to be developed) that take advantage of parallel computing to provide fast approximations [PGMaJT21, page 90].

1.4. **Data Probabilities.** Here we present two novel Nystrom probability sampling distributions that make use of the data matrix, $\boldsymbol{X}$, directly without requiring any knowledge of the associated kernel matrix. The first of these two methods is the data-column sampling distribution defined simply as

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

This sampling distribution can be thought of as the data-column dual of the kernel column sampling probabilities seen in Section 1.2. The second sampling method is the Q matrix column sampling distribution defined as

$$p_i = \left\|\boldsymbol{Q}_{(i,:)}\right\|_2 .$$

where $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$ is the QR factorisation (see Theorem 20) of $\boldsymbol{X}$. This sampling technique can be thought of as the data matrix equivalent of leverage score sampling. There are numerous benefits computing the sampling probabilities using the data matrix directly without needing any knowledge about the corresponding matrix. To start, the obvious benefit is that no properties or elements (along with their corresponding calculations) of the kernel matrix are required saving large amounts of computation time. Second, if the parameters of the kernel function are changed, or an entirely different kernel function is used altogether, then the data-column sampling probabilities need not be re-evaluted.

The main grounds for the theoretical justification for using the data matrix directly to produce a sampling distribution comes from a result shown in [Kar10], presented below.

Theorem 7 (Inner Product of Kernel Matrices)**.** *Let us assume that we observe n i.i.d. random vectors, $X_i \in \mathbb{R}^p$. let us consider the kernel matrix $\boldsymbol{M}$ with entries*

$$\boldsymbol{M}_{ij} = f\left(\frac{\boldsymbol{X}_i^\intercal \boldsymbol{X}_i}{p}\right).$$

We assume that

- (*a*) *We have n/p and p/n remain bounded as $n \to \infty$.*
- (*b*) *$\boldsymbol{\Sigma}_p$ is a positive semi-definite $p \times p$ matrix, and $\|\boldsymbol{\Sigma}_p\|_2 = \sigma_1(\boldsymbol{\Sigma}_p)$ remains bounded in p, that is, there exists $K > 0$, such that $\sigma_1(\boldsymbol{\Sigma}_p) \leq K$, for all p.*
- (*c*) *$\boldsymbol{\Sigma}_p/p$ has a finite limit, that is, there exists $l \in \mathbb{R}$ such that $\lim_{p\to\infty} \operatorname{tr}(\boldsymbol{\Sigma}_p)/p = l$.*
- (*d*) *$\boldsymbol{X}_i = \boldsymbol{\Sigma}_p^{1/2} \boldsymbol{Y}_i$.*
- (*e*) *The entries of $\boldsymbol{Y}_i$, a $p-$dimensional random vector, are i.i.d. Also, denoting by $\boldsymbol{Y}_i(k)$ the k^{th} entry of $\boldsymbol{Y}_i$, we assume that $\mathbb{E}(\boldsymbol{Y}_i(k)) = 0$, $\mathbb{V}(\boldsymbol{Y}_i(k)) = 1$ and $\mathbb{E}\left(|\boldsymbol{Y}_i(k)|^{4+\varepsilon}\right) < \infty$ for some $\varepsilon > 0$.*
- (*f*) *f is a C^1 function is a neighborhood of $l = \lim_{p\to\infty} \operatorname{tr}(\boldsymbol{\Sigma}_p)/p$ and a C^3 function in a neighborhood of 0.*

Under these assumptions, the kernel matrix $\boldsymbol{M}$ can (in probability) be approximated consistently in the operator norm, when p and n tend to ∞, by the matrix $\boldsymbol{K}$, where

$$\boldsymbol{K} = \left(f(0) + f''(0)\frac{\operatorname{tr}\left(\boldsymbol{\Sigma}_p^2\right)}{2p^2}\right) 11' + f'(0)\frac{\boldsymbol{X}\boldsymbol{X}^\intercal}{p} + v_p \mathbb{1},$$

and where

$$v_p = f\left(\frac{\operatorname{tr}(\boldsymbol{\Sigma}_p)}{p}\right) - f(0) - f'(0)\frac{\operatorname{tr}(\boldsymbol{\Sigma}_p)}{p}.$$

In other words,

$$\|\boldsymbol{M} - \boldsymbol{K}\|_2 \to 0$$

in probability, when $p \to \infty$ [Kar10, page 9].

When using the RBF kernel these assumptions typically hold. This theorem shows that as the size of the kernel matrix becomes large, it begins to behave like the underpinning data matrix, up to a factor of a constant.

2. Random Fourier Features

As seen in **??** GPs rely heavily on the Gram matrix (see **??**) 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].

2.1. 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 8, which characterises positive definite functions.

Theorem 8 (Bochner's)**.** *A continuous and shift-invariant function $k(\boldsymbol{x}, \boldsymbol{y}) = k(\boldsymbol{x} - \boldsymbol{y}) = k(\Delta)$ is positive definite (see **??**) 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

$$p(\boldsymbol{w}) = \frac{1}{\sqrt{(2\pi)^D \left|\frac{\sigma^2}{2} \mathbb{1}_{D \times D}\right|}} \exp\left(-\frac{1}{2} \boldsymbol{w}^\intercal \left(\frac{\sigma^2}{2} \mathbb{1}_{D \times D}\right)^{-1} \boldsymbol{w}\right) \tag{7}$$

[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 9.

Definition 9 (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

$$k(\boldsymbol{x}-\boldsymbol{y}) = \int_{\mathbb{C}^d} \exp\left(i\langle \boldsymbol{\omega}, \boldsymbol{x}-\boldsymbol{y}\rangle\right) p(\boldsymbol{\omega})\, d\boldsymbol{\omega}. \tag{8}$$

The main idea of RFF is to approximate the integral in (8) using the following Monte-Carlo estimate

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

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

$$\varphi(\boldsymbol{x}) = \frac{1}{\sqrt{D}}\left[z(\boldsymbol{\omega}_1, \boldsymbol{x}), z(\boldsymbol{\omega}_2, \boldsymbol{x}), \ldots, z(\boldsymbol{\omega}_D, \boldsymbol{x})\right]^{\intercal} \tag{9}$$

where for convenience of notation $z(\boldsymbol{\omega}, \boldsymbol{x}) = \exp\left(i\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle\right)$. This allows the Gram matrix to be estimated as $\boldsymbol{K} \simeq \widetilde{\boldsymbol{K}} = \boldsymbol{Z}\boldsymbol{Z}^{\intercal}$ 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 $z(\boldsymbol{\omega}, \boldsymbol{x})$ satisfies equation (8). The first embedding takes the form

$$z(\boldsymbol{\omega}, \boldsymbol{x}) = \left[\cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle\right), \sin\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle\right)\right]^{\intercal} \tag{10}$$

which satisfies (8) since

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

The other embedding Rahimi and Recht give is

$$z(\boldsymbol{\omega}, \boldsymbol{x}) = \sqrt{2}\cos\left(\langle \boldsymbol{\omega}, \boldsymbol{x}\rangle + b\right) \tag{11}$$

where $b \sim U[0, 2\pi]$. Using a similar argument we can show that this embedding also satisfies (8). However, Sutherland and Schneider [DJSaJS15] argue that the Gaussian RBF kernel is better suited for the

embedding given in (10). To summarise their argument we denote

$$\varphi_1(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \begin{bmatrix} \cos\left(\langle \boldsymbol{\omega}_1, \boldsymbol{x} \rangle\right) \\ \cos\left(\langle \boldsymbol{\omega}_2, \boldsymbol{x} \rangle\right) \\ \vdots \\ \cos\left(\langle \boldsymbol{\omega}_{D/2}, \boldsymbol{x} \rangle\right) \\ \sin\left(\langle \boldsymbol{\omega}_1, \boldsymbol{x} \rangle\right) \\ \vdots \\ \sin\left(\langle \boldsymbol{\omega}_{D/2}, \boldsymbol{x} \rangle\right) \end{bmatrix} \tag{12}$$

to be the feature map corresponding to embedding in equation (10) and

$$\varphi_2(\boldsymbol{x}) = \sqrt{\frac{2}{D}} \begin{bmatrix} \cos\left(\langle \boldsymbol{\omega}_1, \boldsymbol{x} \rangle + b_1\right) \\ \vdots \\ \cos\left(\langle \boldsymbol{\omega}_D, \boldsymbol{x} \rangle + b_D\right) \end{bmatrix} \tag{13}$$

to be the feature map corresponding to equation (11). 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 10.

Proposition 10. *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 (12) 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 10 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 4, which was used to approximate a Gram matrix for the data set $\boldsymbol{X}$ using the feature map from (12).

Algorithm 4: 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 $\boldsymbol{W} \triangleq [\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D]^\intercal \in \mathbb{R}^{D \times d}$ where $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} p(\cdot)$
$\boldsymbol{Z} = \frac{1}{\sqrt{D}} \left[\cos\left(\boldsymbol{W}\boldsymbol{X}^\intercal\right), \sin\left(\boldsymbol{W}\boldsymbol{X}^\intercal\right)\right]^\intercal$
$\widetilde{\boldsymbol{K}} = \boldsymbol{Z}\boldsymbol{Z}^\intercal$
return $\widetilde{\boldsymbol{K}}$

Algorithm 4 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 (7) corresponds to a multivariate Gaussian distribution with mean $\mathbf{0}$ and covariance matrix $\left(\frac{\sigma}{\sqrt{2}}\right)^{-2} \mathbb{1}_{D \times D}$. This means $\boldsymbol{W}$ can simply be constructed as $\boldsymbol{W} = \left(\frac{\sigma}{\sqrt{2}}\right)^{-1} [\boldsymbol{\omega}_1, \ldots, \boldsymbol{\omega}_D]^\intercal$ where $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} \mathcal{N}\left(\mathbf{0}, \mathbb{1}_{D \times D}\right)$. 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 4 then it produces an unbiased estimate, $\widetilde{\boldsymbol{K}}_{\text{RFF}}$, for the Gram matrix. This stated more precisely in Lemma 11.

Lemma 11. *$\widetilde{\boldsymbol{K}}_{RFF}$ is an unbiased estimate of $\boldsymbol{K}$, that is*

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

[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

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

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

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

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.

2.2. **Orthogonal Random Features.** In the previous chapter Algorithm 4 assumed some sort of mechanism for producing the transformation matrix $\boldsymbol{W}$. The construction presented in 2.1 involved sampling $\boldsymbol{\omega}_i \overset{\text{iid}}{\sim} p(\cdot)$. For the Gaussian RBF kernel this meant sampling from the multivariate Gaussian distribution $\mathcal{N}(\mathbf{0}, \mathbb{1}_{D\times D})$. 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 2.1 [Liu21]. We shall consider the two methods proposed by Yu *et al.* [Yu16]; the first method here and the second in the following section (2.3). 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 3.2) 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

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

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 $\boldsymbol{S} = \text{diag}(\psi_1, \psi_2, \ldots, \psi_D)$ where $\psi_i \overset{\text{iid}}{\sim} \chi_D$. This means

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

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

$$\boldsymbol{W}_{\text{ORF}} = \left(\frac{\sigma}{\sqrt{2}}\right)^{-1} \boldsymbol{SQ}. \tag{14}$$

The main downside the the ORF method is that the QR-factorization brings a computational cost of $\mathcal{O}(Dd)$. Fortunately when using $\boldsymbol{W}_{\text{ORF}}$ as our transformation matrix in Algorithm 4 the approximate Gram matrix $\widetilde{\boldsymbol{K}}_{\text{RFF}}$ is an unbiased estimate of $\boldsymbol{K}$, stated more formally in Theorem 12.

Theorem 12. *$\widetilde{\boldsymbol{K}}_{ORF}$ is an unbiased estimate of $\boldsymbol{K}$, that is*

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

[Yu16, page 3].

Proof. Again, we shall prove this for the Gaussian kernel, although proofs for other kernels are analogous. Let $\boldsymbol{z} = \boldsymbol{x} - \boldsymbol{y}$.

$$\mathbb{E}\left[\left(\widetilde{\boldsymbol{K}}_{\text{ORF}}\right)_{ij}\right] = \mathbb{E}\left[\frac{1}{D}\sum_{j=1}^{D}\cos\left(\langle\boldsymbol{\omega}_j, \boldsymbol{z}\rangle\right)\right] = \frac{1}{D}\sum_{j=1}^{D}\mathbb{E}\left[\cos\left(\langle\boldsymbol{\omega}_j, \boldsymbol{z}\rangle\right)\right].$$

Based on the definition of the ORF method, $\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_D$ are D random vectord given by $\boldsymbol{w}_i = s_i \boldsymbol{u}_i$, with $\boldsymbol{u}_1, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_d$ a uniformly chosen random orthonormal basis for $\mathbb{R}^d$, and s_i's are independently

$\chi-$distributed random variables with d degrees of freedom. It is easy to show that for each i, $\boldsymbol{w}_i$ is distributed according to $\mathcal{N}(\mathbf{0}, \mathbb{1}_{d\times d})$ and hence by Bochner's theorem

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

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

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 $\tau = \left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_2 / \frac{\sigma}{\sqrt{2}}$ 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].

2.3. **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

$$\boldsymbol{W}_{\text{ROM}} = \sqrt{d}\prod_{i=1}^{k}\boldsymbol{S}\boldsymbol{D}_i \tag{15}$$

where $\boldsymbol{S} \in \mathbb{R}^{D\times D}$ has orthogonal rows and $\boldsymbol{D} = \text{diag}\left(\delta_1, \ldots, \delta_D\right) \in \mathbb{R}^{D\times D}$ where $\delta_i \overset{\text{iid}}{\sim} U\left(\{-1, 1\}\right)$. 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 13, 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 13 (Hadamard Matrix)**.** *The Hadamard matrix* $\boldsymbol{H}_i \in \mathbb{R}^{\left(2^{i-1}\times 2^{i-1}\right)}$ *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 13 or by padding with 0. This provides a concrete means for which one can generate a transformation matrix

$$\sqrt{d}\prod_{i=1}^{k}\boldsymbol{H}\boldsymbol{D}_i \tag{16}$$

where $\boldsymbol{H}$ is an appropriately sized Hadamard matrix. It is easy to check that the matrix generated by equation (16) 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 $\mathcal{O}\left(D\log(d)\right)$ 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 $\mathcal{O}\left(D\log(d)\right)$ 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]	$\mathcal{O}\left(D\log(d)\right)$	$\mathcal{O}(D)$

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 (16). They also show that choosing odd values of k in (16) 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 (17). The method for constructing transformation matrices in this manner is referred to as Structured Orthogonal Random Features (SORF).

$$\boldsymbol{W}_{\text{SORF}} = \sqrt{d}\boldsymbol{H}\boldsymbol{D}_3\boldsymbol{H}\boldsymbol{D}_2\boldsymbol{H}\boldsymbol{D}_1 \tag{17}$$

This is the same transformation matrix estimate that Yu *et al.* provides. Unfortunately using the SORF method in Algorithm 4 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 $\left\|\boldsymbol{x}_i - \boldsymbol{x}_j\right\|_2 / \frac{\sigma}{\sqrt{2}}$ [Liu21, page 8].

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

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

[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 $\boldsymbol{HD}_2$) 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 3.

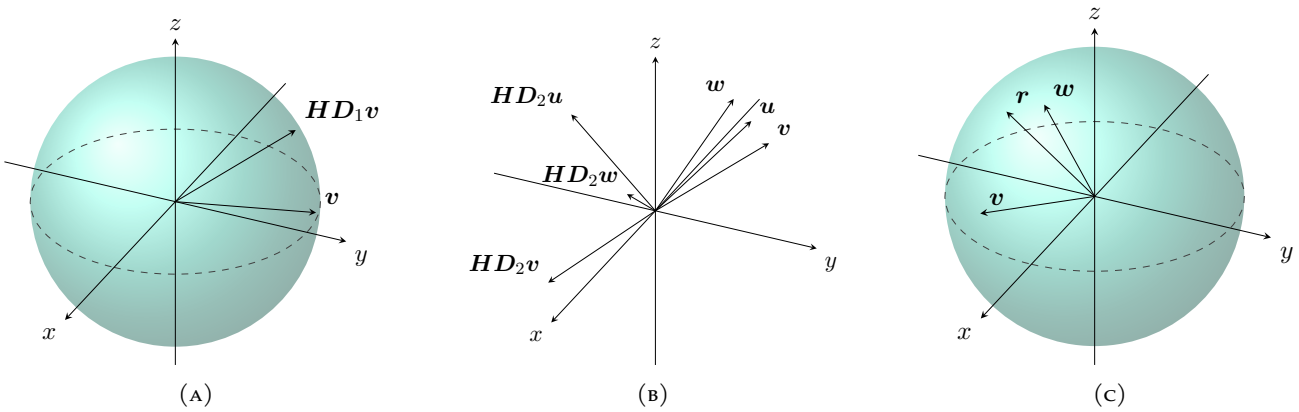


FIGURE 3. A visual representation for the roles of each matrix block in the SORF method. The first block $\boldsymbol{HD}_1$ rotates $\boldsymbol{v}$ so that single dimension bears too much of the l^2 norm seen in panel (A). In panel (B) the second block $\boldsymbol{HD}_2$ transforms vectors so that 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.

3. Krylov Subspace Methods

In this section we will focus on how iterative methods and, in particular a class of iterative methods known as Krylov Subspace methods, may be used to solve a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. 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 computation intractable. 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 by nature endow these systems with these desired properties. For example, 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. Even if the method is prematurely stopped before reaching an exact solution, the solution obtained on the final iteration will, in some sense, be a good enough estimate. Additionally, 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 for computing $\boldsymbol{A}\boldsymbol{x}$.

3.1. Krylov Subspaces.

The main usefulness of Krylov subspaces come from their ability in assisting us to solving linear systems. To this end, consider the problem of solving the linear system

$$\boldsymbol{A}\boldsymbol{x}^\star = \boldsymbol{b} \tag{18}$$

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 $\boldsymbol{A}\boldsymbol{v}$ for any $\boldsymbol{v}$. How could this routine be utilized in such a manner to provide with a solution to (18)? Before answering this, consider the following theorem.

Theorem 14. *For $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ 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, page 287].

Proof. Let us show that the sequence $\boldsymbol{S}_n = \sum_{k=0}^{n} \boldsymbol{A}^k$ (where n is some natural number) of partial sums of the series on the right-hand side of Theorem 14 is fundamental in $\mathbb{C}^{n\times n}$. Indeed, by using the fact that $\|\boldsymbol{A}\boldsymbol{B}\| \leq \|\boldsymbol{A}\|\|\boldsymbol{B}\|$ for any $\boldsymbol{A}, \boldsymbol{B} \in \mathbb{C}^{n\times n}$, we conclude that

$$\|\boldsymbol{S}_{n+p} - \boldsymbol{S}_n\| \leq \|\boldsymbol{A}^{n+1}\| + \cdots + \|\boldsymbol{A}^{n+p}\| \leq q^{n+1} + \cdots + q^{n+p}$$

for all $n, p \in \mathbb{N}$. Since $q < 1$, this yields the desired result. Since $\mathbb{C}^{n\times n}$ is a Banach space, the sequence $\{\boldsymbol{S}_n\}_n^\infty$ uniformly converges to an operator $\boldsymbol{S} \in \mathbb{C}^{n\times n}$. Let us show that $\boldsymbol{S}\left(\mathbb{1}_{n\times n} - \boldsymbol{A}\right) = \left(\mathbb{1}_{n\times n} - \boldsymbol{A}\right)\boldsymbol{S} = \mathbb{1}_{n\times n}$. To this end, we note that

$$\|\left(\mathbb{1}_{n\times n} - \boldsymbol{A}\right)\boldsymbol{S}_n - \left(\mathbb{1}_{n\times n} - \boldsymbol{A}\right)\boldsymbol{S}\| \leq \|\mathbb{1}_{n\times n} - \boldsymbol{A}\| \cdot \|\boldsymbol{S} - \boldsymbol{S}_n\| \to 0$$

as $n \to \infty$. Therefore, it suffices to show that the sequence $\{(\mathbb{1}_{n\times n} - \boldsymbol{A})\, \boldsymbol{S}_n\}_{n=1}^{\infty}$ uniformly converges to the identity operator. We have

$$\boldsymbol{S}_n\,(\mathbb{1}_{n\times n} - \boldsymbol{A}) = (\mathbb{1}_{n\times n} - \boldsymbol{A})\,\boldsymbol{S}_n = \sum_{k=0}^{n} \boldsymbol{A}^k - \sum_{k=1}^{n+1} \boldsymbol{A}^k = \mathbb{1}_{n\times n} - \boldsymbol{A}^{n+1},$$

that is, $\|\,(\mathbb{1}_{n\times n} - \boldsymbol{A})\,\boldsymbol{S}_n - \mathbb{1}_{n\times n}\| = \|\boldsymbol{A}^{n+1}\| \leq q^{n+1}$, which vanishes as $n \to \infty$. Thus $\boldsymbol{S} = (\mathbb{1}_{n\times n} - \boldsymbol{A})^{-1}$. □

Consider a matrix for which $\|\boldsymbol{A}\| < 2$, it follows that $\|\mathbb{1}_{n\times n} - \boldsymbol{A}\| < 1$ meaning $\mathbb{1}_{n\times n} - (\mathbb{1}_{n\times n} - \boldsymbol{A})$ is invertible and $\boldsymbol{A}^{-1} = (\mathbb{1}_{n\times n} - (\mathbb{1}_{n\times n} - \boldsymbol{A}))^{-1} = \sum_{k=0}^{\infty} (\mathbb{1}_{n\times n} - \boldsymbol{A})^k$. Returning to (18) for any $\boldsymbol{x}_0 \in \mathbb{R}^n$ we have

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

where $\boldsymbol{r_0} = \boldsymbol{A}\boldsymbol{x}^\star - \boldsymbol{A}\boldsymbol{x_0}$. An obvious question that arises is whether a closed form solution of the above equation can be found. To answer this, we need to enlist the help of the Cayley-Hamilton theorem.

Theorem 15 (Cayley-Hamilton)**.** *Let $p_n(\lambda) = \sum_{i=0}^{n} c_i \lambda^i$ be the characteristic polynomial of the matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$, then $p_n(\boldsymbol{A}) = \boldsymbol{0}$* [Hor13, page 109].

The Cayley-Hamilton Theorem implies that

$$\begin{aligned}
\boldsymbol{0} &= c_0 + c_1\boldsymbol{A} + \ldots + c_{n-1}\boldsymbol{A}^{n-1} + c_n\boldsymbol{A}^n \\
\boldsymbol{0} &= \boldsymbol{A}^{-1}c_0 + c_1 + \ldots + c_{n-1}\boldsymbol{A}^{n-2} + c_n\boldsymbol{A}^{n-1} \\
\boldsymbol{A}^{-1} &= \alpha_1\mathbb{1}_{n\times n} + \ldots + \alpha_{n-1}\boldsymbol{A}^{n-2} + \alpha_n\boldsymbol{A}^{n-1}
\end{aligned}$$

where $\alpha_i = -c_i/c_0$. The above 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

$$\boldsymbol{x}^\star = \boldsymbol{x}_0 + \boldsymbol{A}^{-1}\boldsymbol{r}_0 = \boldsymbol{x}_0 + \alpha_1\boldsymbol{r}_0 + \ldots + \alpha_{n-1}\boldsymbol{A}^{n-2}\boldsymbol{r}_0 + \alpha_n\boldsymbol{A}^{n-1}\boldsymbol{r}_0. \tag{19}$$

This also shows that $\boldsymbol{x}^\star \in \boldsymbol{x}_0 + \text{l. s}\left\{\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, \ldots, \boldsymbol{A}^{n-1}\boldsymbol{r}_0\right\}$. One idea for finding a solution to (18) 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 16 (Krylov Subspace)**.** *The Krylov Subspace of order k generated by the matrix $\boldsymbol{A} \in \mathbb{C}^{n\times n}$ and the vector $\boldsymbol{v} \in \mathbb{C}^n$ is defined as*

$$\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v}) = \text{l. s}\left\{\boldsymbol{r}_0, \boldsymbol{A}\boldsymbol{r}_0, \boldsymbol{A}^2\boldsymbol{r}_0, \ldots, \boldsymbol{A}^{k-1}\boldsymbol{r}_0\right\}$$

for $k \geq 1$ and $\mathcal{K}_0(\boldsymbol{A}, \boldsymbol{v}) = \{\boldsymbol{0}\}$ [Tre97, page 245].

For the purposes of solving (18) it is important to understand how $\mathcal{K}_k(\boldsymbol{A}, \boldsymbol{v})$ grows for larger and larger k since a solution for (18) will be present in a Krylov Subspace that cannot be grown any larger, according

to (19). 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 17. *There is a positive called the grade of $\boldsymbol{v}$ with respect to $\boldsymbol{A}$, denoted $t_{\boldsymbol{v},\boldsymbol{A}}$, where*

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

Theorem 17 essentially tells us 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 $\left\{\boldsymbol{v}, \boldsymbol{A}\boldsymbol{v}, \boldsymbol{A}^2\boldsymbol{v}, \ldots, \boldsymbol{A}^{k-1}\boldsymbol{v}\right\}$ serves as a basis for $\mathcal{K}_k\left(\boldsymbol{A},\boldsymbol{v}\right)$ and $\mathcal{K}_{k-1}\left(\boldsymbol{A},\boldsymbol{v}\right) \subsetneq \mathcal{K}_k\left(\boldsymbol{A},\boldsymbol{v}\right)$. Conversely, any new vectors formed beyond $t_{\boldsymbol{v},\boldsymbol{A}}$ will be linearly independent meaning $\mathcal{K}_k\left(\boldsymbol{A},\boldsymbol{v}\right) \subsetneq \mathcal{K}_{k+1}\left(\boldsymbol{A},\boldsymbol{v}\right)$ for $k \geq t_{\boldsymbol{v},\boldsymbol{A}}$. While $t_{\boldsymbol{v},\boldsymbol{A}}$ obviously plays a central role in determining a suitable basis whose span contains $\boldsymbol{A}^{-1}\boldsymbol{b}$, its importance is made abundantly clear in the following corollary.

Corollary 18.

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

Proof. Recall from Cayley-Hamilton (Theorem 15) that

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

But since $\mathcal{K}_k\left(\boldsymbol{A},\boldsymbol{v}\right) = \mathcal{K}_{k+1}\left(\boldsymbol{A},\boldsymbol{v}\right)$ 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 $\boldsymbol{A}^{-1}\boldsymbol{v} \in \mathcal{K}_k\left(\boldsymbol{A},\boldsymbol{v}\right)$ 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 $\left\{\boldsymbol{v}, \boldsymbol{A}\boldsymbol{v}, \boldsymbol{A}^2\boldsymbol{v}, \ldots, \boldsymbol{A}^{t-1}\boldsymbol{v}\right\}$ are linearly dependent which means that $\dim\left(\mathcal{K}_k\left(\boldsymbol{A},\boldsymbol{v}\right)\right) < t$, which provides us with our contradiction. □

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

Corollary 19. *For any $\boldsymbol{x_0}$ we have*

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

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

3.2. **Gram-Schmidt Process and QR factorisations.** Many areas of linear algebra involve studying the column space of matrices. The QR factorisation both provides a powerful tool to better understand the column space of a matrix and serves as an important factorisation mechanism for many numerical methods. Suppose that a matrix $\boldsymbol{A} = [\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_n] \in \mathbb{R}^{n\times n}$ 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 $\text{l. s}\{\boldsymbol{a}_i\}_{i=1}^n = \text{l. s}\{\boldsymbol{q}_i\}_{i=1}^n$. First set $\boldsymbol{q}_1 = \frac{\boldsymbol{a}_1}{\|\boldsymbol{a}_i\|}$, clearly $\text{l. s}\{\boldsymbol{a}_1\} = \text{l. s}\{\boldsymbol{q}_1\}$. 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 $\text{l. s}\{\boldsymbol{a}_1, \boldsymbol{a}_2\} = \text{l. s}\{\boldsymbol{q}_1, \boldsymbol{q}_2\}$. Next, the vector $\boldsymbol{q}_3'$ is constructed so that

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

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 $\text{l. s}\{\boldsymbol{a}_1, \boldsymbol{a}_2, \boldsymbol{a}_3\} = \text{l. s}\{\boldsymbol{q}_1, \boldsymbol{q}_2, \boldsymbol{q}_3\}$. Continuing in this fashion the k^{th} vector in our orthornormal basis is computed as

$$\boldsymbol{q}_k = \frac{\boldsymbol{a}_k - \sum_{i=1}^{k-1} r_{i,k}\cdot\boldsymbol{q}_i}{r_{k,k}} \tag{20}$$

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 $\text{l. s}\{\boldsymbol{a}_1, \boldsymbol{a}_2, \ldots, \boldsymbol{a}_k\} = \text{l. s}\{\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots, \boldsymbol{q}_k\}$. This procedure is known as the Gram-Schmidt process [Ber96, Tre97, Dem97] and is summarized in the following algorithm.

Algorithm 5: Classical Gram-Schmidt

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

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 $\{\boldsymbol{q}_i\}_{i=1}^n$ can be done in a matrix form as

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

or more written succinctly as

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{R} \tag{21}$$

where $\boldsymbol{Q} = [\boldsymbol{q}_1, \boldsymbol{q}_2, \dots \boldsymbol{q}_n]$ 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 the 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 sometimes the thin) QR-factorization. The following two theorems on the QR factorization are stated without proof.

Theorem 20. *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 21. *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 5 is rarely used as the procedure becomes numerically unstable if $\{\boldsymbol{a}_i\}_{i=1}^n$ are almost linearly dependent. Before looking at ways to resolve these numerical instabilities, a quick recap of projectors is useful. 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 $\boldsymbol{Q} = [\boldsymbol{q}_1, \boldsymbol{q}_2, \dots \boldsymbol{q}_m, \boldsymbol{0}, \dots, \boldsymbol{0}] \in \mathbb{R}^{n\times n}$. 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 creates $\boldsymbol{q}_k'$ using the following orthogonal projection

$$\boldsymbol{q}_k' = \boldsymbol{P}_{A_k^\perp}\boldsymbol{a}_k \tag{22}$$

where $A_k = \mathrm{l.s}\,\{\boldsymbol{a}_i\}_{i=1}^k$ and $\boldsymbol{P}_{A_1^\perp} = \mathbb{1}_{n\times n}$ 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

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

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

where $\boldsymbol{Q} = [\boldsymbol{q}_1, \boldsymbol{q}_2, \ldots \boldsymbol{q}_k, 0, \ldots, 0]$. 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 equivalent, 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 6: 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 $\{\boldsymbol{q}_i\}_{i=1}^n$

3.3. **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 3.1 it was shown 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 16 indicated 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 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 appropriate 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 other terms

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

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 $\boldsymbol{V}$ matrix offers in the QR-factorisation of $\boldsymbol{K}$. With this in mind let $\boldsymbol{K} = \boldsymbol{V}\boldsymbol{R}$ be the full QR-factorisation of $\boldsymbol{K}$. Then

$$\begin{aligned}\boldsymbol{A}\boldsymbol{V}\boldsymbol{R} &= \boldsymbol{A}\boldsymbol{K}\\ \boldsymbol{A}\boldsymbol{V} &= \boldsymbol{A}\boldsymbol{K}\boldsymbol{R}^{-1}\\ \boldsymbol{A}\boldsymbol{V} &= \boldsymbol{K}\boldsymbol{C}\boldsymbol{R}^{-1}\\ \boldsymbol{A}\boldsymbol{V} &= \boldsymbol{V}\boldsymbol{R}\boldsymbol{C}\boldsymbol{R}^{-1}\\ \boldsymbol{A}\boldsymbol{V} &\triangleq \boldsymbol{V}\boldsymbol{H}.\end{aligned}$$

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 $\boldsymbol{V}$ such that the range of $\boldsymbol{V}$ is $\mathcal{K}_n(\boldsymbol{A}, \boldsymbol{r}_0)$ and

$$\boldsymbol{V}^\intercal\boldsymbol{A}\boldsymbol{V} = \boldsymbol{H}. \tag{23}$$

Again, in practice, it may be very difficult to compute this entire expression forcing us to search for approximative alternatives. Consider (23) for which the only first k columns of $\boldsymbol{V}$ have been computed. Let $\boldsymbol{V}_k = [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k]$ and $\boldsymbol{V}_u = [\boldsymbol{v}_{k+1}, \boldsymbol{v}_{k+2}, \ldots, \boldsymbol{v}_n]$. Then

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

$$\begin{bmatrix} \boldsymbol{V}_k^\intercal \boldsymbol{A}\boldsymbol{V}_k & \boldsymbol{V}_k^\intercal \boldsymbol{A}\boldsymbol{V}_u \\ \boldsymbol{V}_u^\intercal \boldsymbol{A}\boldsymbol{V}_k & \boldsymbol{V}_u^\intercal \boldsymbol{A}\boldsymbol{V}_u \end{bmatrix} = \begin{bmatrix} \boldsymbol{H}_k & \boldsymbol{H}_{u,k} \\ \boldsymbol{H}_{k,u} & \boldsymbol{H}_u \end{bmatrix}$$

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

$$\boldsymbol{V}_k^\intercal \boldsymbol{A}\boldsymbol{V}_k = \boldsymbol{H}_k \tag{24}$$

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 $\boldsymbol{V} \in \mathbb{K}^{n\times n}$ and $\boldsymbol{H} \in \mathbb{K}^{n\times n}$ that satisfies $\boldsymbol{A}\boldsymbol{V} = \boldsymbol{V}\boldsymbol{H}$. However, in general, we may not be so fortunate in finding a $\boldsymbol{V}_k \in \mathbb{K}^{n\times k}$ and $\boldsymbol{H}_k \in \mathbb{K}^{n\times k}$ to satisfy $\boldsymbol{A}\boldsymbol{V}_k = \boldsymbol{V}_k\boldsymbol{H}_k$ for any $k < n$. Instead we can adjust this equality by adding an error $\boldsymbol{E}_k \in \mathbb{K}^{n\times k}$ to force an equality. Our expression now becomes

$$\boldsymbol{V}_k^\intercal \boldsymbol{A}\boldsymbol{V}_k = \boldsymbol{H}_k + \boldsymbol{E}_k. \tag{25}$$

A judicious choice of $\boldsymbol{E}_k$ must be made to also retain equality in (24), meaning $\boldsymbol{V}_k^\intercal \boldsymbol{E}_k = \boldsymbol{0}$. Since $\{\boldsymbol{v}_i\}_{i=1}^k$ forms an orthornormal basis for $\mathcal{K}_n(\boldsymbol{A}, \boldsymbol{r}_0)$, consider the following choice of $\boldsymbol{E}_k$,

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

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

$$\boldsymbol{V}_k^\intercal \boldsymbol{E} = \boldsymbol{V}^\intercal(\boldsymbol{v}_{k+1}\boldsymbol{h}_k) = (\boldsymbol{V}^\intercal \boldsymbol{v}_{k+1})\,\boldsymbol{h}_k^\intercal = \boldsymbol{0}.$$

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 $\boldsymbol{A}\boldsymbol{V}_k$ can be written as

$$\boldsymbol{A}\boldsymbol{V}_k = \boldsymbol{V}_k\boldsymbol{H}_k + \boldsymbol{v}_{k+1}\boldsymbol{h}_k^\intercal \tag{26}$$

where

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

Equating the j^{th} columns of (26) yields

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

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

$$\boldsymbol{v}_m^\intercal \boldsymbol{A}\boldsymbol{v}_j = \sum_{i=1}^{j+1} h_{i,j}\boldsymbol{v}_m^\intercal \boldsymbol{v}_i = h_{m,j}$$

and so

$$h_{j+1,j}\boldsymbol{v}_{j+1} = \boldsymbol{A}\boldsymbol{v}_j - \sum_{i=1}^{j} h_{i,j}\boldsymbol{v}_i. \tag{27}$$

From (27) we find that $\boldsymbol{v}_{j+1}$ can be computed using a recurrance involving its previous Krylov factors. This bears a striking resemblance to (20) having a virtually identical setup to computing an orthornormal

basis using the modified Gram-Schmidt process (Algorithm 6). As such, values for $\boldsymbol{v}_{j+1}$ 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 7.

Algorithm 7: Arnoldi Algorithm

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

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

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 $\boldsymbol{AV} = \boldsymbol{VT}$ yields

$$\boldsymbol{A}\boldsymbol{v}_j = \beta_{j-1}\boldsymbol{v}_{j-1} + \alpha_j \boldsymbol{v}_j + \beta_j \boldsymbol{v}_{j+1}. \tag{28}$$

Again since $\{\boldsymbol{v}_i\}_{i=1}^n$ form an orthornormal basis, multiplying both sides of (28) by $\boldsymbol{v}_j$ gives $\boldsymbol{v}_j \boldsymbol{A} \boldsymbol{v}_j = \alpha_j$. A simplified version of the Arnoldi algorithm can be devised to compute $\{\boldsymbol{v}_i\}_{i=1}^n$ and $\boldsymbol{T}$ for symmetric matrices known as the Lanczos algorithm [Dem97]. The Lanczos algorithm is presented in Algorithm 8.

Algorithm 8: Lanczos Algorithm

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

$\boldsymbol{v}_1 = \boldsymbol{r}_0/\|\boldsymbol{r}_0\|$, $\beta_0 = 0$, $\boldsymbol{v}_0 = 0$
for $j = 1$ **to** k **do**
  $\boldsymbol{z} = \boldsymbol{A}\boldsymbol{v}_j$
  $\alpha_j = \langle \boldsymbol{v}_j, \boldsymbol{z} \rangle$
  $\boldsymbol{z} = \boldsymbol{z} - \alpha_j \boldsymbol{v}_j - \beta_{j-1}\boldsymbol{v}_{j-1}$
  $\beta_j = \|z\|$
  if $\beta_j = 0$ **then**
    return $\boldsymbol{V}_k, \boldsymbol{T}_k$
  end
  $\boldsymbol{v}_{j+1} = \boldsymbol{z}/\beta_j$
end
return $\boldsymbol{V}_k, \boldsymbol{T}_k$

For the Lanczos algorithm, (26) can be re-written in the a more compact form as

$$\boldsymbol{A}\boldsymbol{V}_k \triangleq \boldsymbol{V}_k \boldsymbol{T}_{k+1,k} \tag{29}$$

where $\boldsymbol{T}_{k+1,k} = \boldsymbol{T}_k + \boldsymbol{v}_{k+1}\boldsymbol{t}_k^\intercal$.

3.4. **Optimality Conditions.** So far we have shown that $\boldsymbol{x}^\star \in \mathcal{K}_{t_{\boldsymbol{r}_0},\boldsymbol{A}}(\boldsymbol{A}, \boldsymbol{r}_0)$ where $n = t_{\boldsymbol{r}_0}$ is the grade of $\boldsymbol{r}_0$ with respect to $\boldsymbol{A}$. Moreover from Section 3.3 we found ways to construct a basis for $\mathcal{K}_{t_{\boldsymbol{r}_0},\boldsymbol{A}}(\boldsymbol{A}, \boldsymbol{r}_0)$ allowing us to generate vectors with these affine spaces, namely the Arnoldi algorithm (Algorithm 7) and Lanczos algorithm (Algorithm 8) for non-symmertic and symmertic systems, respectively. From now on $\mathcal{K}_{t_{\boldsymbol{r}_0},\boldsymbol{A}}(\boldsymbol{A}, \boldsymbol{r}_0)$ will be abbreviated to $\mathcal{K}_{t_{\boldsymbol{r}_0},\boldsymbol{A}}$ when the context is clear. The question still remains however, how should one choose an $\boldsymbol{x}_k$ that best approximates $\boldsymbol{x}^*$ satisfying (18)? The following are some of the most well known methods for selecting a suitable $\boldsymbol{x}_k$.

(a) Select an $\boldsymbol{x}_k \in \boldsymbol{x}_0 + \mathcal{K}_k$ which minimizes $\|\boldsymbol{x}_k - \boldsymbol{x}^*\|_2$. While this method seems like the most intuitive and natural way to select $\boldsymbol{x}_k$, it is unfortunately of no practical use since there is not enough information in the Krylov Subspace to find an $\boldsymbol{x}_k$ which matches this profile.
(b) Select an $\boldsymbol{x}_k \in \boldsymbol{x}_0 + \mathcal{K}_k$ which minimizes $\|\boldsymbol{r}_k\|_2$ (recall this is the residual of $\boldsymbol{x}_k$, that is, $\boldsymbol{r}_k = \boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_k$). 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 $\boldsymbol{A}$ respectively.
(c) When $\boldsymbol{A}$ is a positive definite matrix it defines a norm $\|\boldsymbol{r}\|_{\boldsymbol{A}} = (\boldsymbol{r}^\intercal \boldsymbol{A}\boldsymbol{r})^{\frac{1}{2}}$, called the energy norm. Select an $\boldsymbol{x}_k \in \boldsymbol{x}_0 + \mathcal{K}_k$ which minimizes $\|\boldsymbol{r}_k\|_{\boldsymbol{A}^{-1}}$ which is equivalent to minimizing $\|\boldsymbol{x}_k - \boldsymbol{x}\|_A$. This technique is known as the CG (conjugate gradient) algorithm.

(d) Select an $\boldsymbol{x}_k \in \boldsymbol{x}_0 + \mathcal{K}_k$ for which $\boldsymbol{r}_k \perp \mathcal{W}_k$ 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 22.

Theorem 22. *In the context of the above selection method, if $\boldsymbol{A} \succ \boldsymbol{0}$ and $\mathcal{W}_k = \mathcal{K}_k$ in method (4) then it produces the same $\boldsymbol{x}_k$ in method (3)* [Dem97].

Proof. Since $\mathcal{W}_k = \mathcal{K}_k$, any basis for $\mathcal{W}_k$ is a basis for $\mathcal{K}_k$. In fact, we can write $\boldsymbol{W}_k = \boldsymbol{K}_k\boldsymbol{B}$ for some non-singular $\boldsymbol{B} \in \mathbb{R}^{k\times k}$. This means

$$\boldsymbol{W}_k^\intercal \boldsymbol{A}\boldsymbol{K}_k = \boldsymbol{B}^\intercal \boldsymbol{K}_k^\intercal \boldsymbol{A}\boldsymbol{K}_k$$

and since $\boldsymbol{A}$ is symmetric positive definite then $\boldsymbol{K}_k^\intercal \boldsymbol{A}\boldsymbol{K}_k$ is also symmetric positive definite, and hence the entire product is non-singular. □

In fact the 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 $\{\boldsymbol{v}_i\}_{i=1}^k$ and $\{\boldsymbol{w}_i\}_{i=1}^k$ are basis elements for $\mathcal{K}$ and $\mathcal{W}$ respectively. Let

$$\begin{aligned}\boldsymbol{K}_k &\triangleq [\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_k] \in \mathbb{R}^{n\times k}\\ \boldsymbol{W}_k &\triangleq [\boldsymbol{w}_1, \boldsymbol{w}_2, \ldots, \boldsymbol{w}_k] \in \mathbb{R}^{n\times k}\end{aligned}$$

then the Petrov-Galerkin conditions can be imposed as follows

$$\begin{aligned}\boldsymbol{K}_k\boldsymbol{y}_k &= \boldsymbol{z}_k, \quad \text{for some } \boldsymbol{y}_k \in \mathbb{R}^k\\ \boldsymbol{W}_k^\intercal\left(\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{K}_k\boldsymbol{y}_k\right) &= \boldsymbol{0}.\end{aligned}$$

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

$$\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{K}_k\left(\boldsymbol{W}_k^\intercal \boldsymbol{A}\boldsymbol{K}_k\right)^{-1}\boldsymbol{W}_k\boldsymbol{r}_0. \tag{30}$$

This justifies a general form of the projection method algorithm presented in Algorithm 9.

Algorithm 9: General Projection Method

output: An approximation of $\boldsymbol{x}^*$, $\boldsymbol{x}_k$.

for $k = 1, \ldots$ **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 $\boldsymbol{x}_k$

3.5. **Conjugate Gradient Algorithm.** From Section 3.4 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 $\boldsymbol{A}\boldsymbol{V}_k = \boldsymbol{V}_k\boldsymbol{T}_k + \boldsymbol{v}_{k+1}\boldsymbol{t}_k^\intercal$ where $\boldsymbol{t}_k \triangleq [0, 0, \ldots, 0, \beta_k]^\intercal \in \mathbb{K}^k$ and the columns of $\boldsymbol{V}_k$ span $\mathcal{K}_k$. Recall that $\boldsymbol{x}_k$ can be expressed as $\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{K}_k\left(\boldsymbol{W}_k^\intercal \boldsymbol{A}\boldsymbol{K}_k\right)^{-1}\boldsymbol{W}_k\boldsymbol{r}_0$ ((30)) 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. Thus 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 $\mathcal{K}_k = \text{Range}\left(\boldsymbol{V}_k\right)$ where $\boldsymbol{V}_k$ has full column rank then $\boldsymbol{z}_k$ can be represented as $\boldsymbol{z}_k = \boldsymbol{V}_k\boldsymbol{y}$ for a unique $\boldsymbol{y} \in \mathbb{R}^k$ so that

$$\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{V}_k\boldsymbol{y}. \tag{31}$$

Coupling this with the Petrov-Galerkin conditions ensures

$$\begin{aligned} \boldsymbol{V}_k^\intercal\left(\boldsymbol{r}_0 - \boldsymbol{A}\boldsymbol{V}_k\boldsymbol{y}\right) &= \boldsymbol{0} \\ \boldsymbol{V}_k^\intercal\boldsymbol{A}\boldsymbol{V}_k\boldsymbol{y} &= \boldsymbol{V}_k^\intercal\boldsymbol{r}_0 \\ \boldsymbol{T}_k\boldsymbol{y} &= \|\boldsymbol{r}_0\|\boldsymbol{e}_1. \end{aligned} \tag{32}$$

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

(a) The approximate solutions $\boldsymbol{x}_k$
(b) The residual vectors $\boldsymbol{r}_k$
(c) The conjugate gradient vectors $\boldsymbol{p}_k$

The conjugate gradient vectors include the word 'gradient' since the attempt to find the direction of steepest descent that minimizes $\|\boldsymbol{r}_k\|_{\boldsymbol{A}^{-1}}$. It also includes the word '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 (shown in Lemma 23).

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

$$\boldsymbol{T}_k = \boldsymbol{L}_k\boldsymbol{D}_k\boldsymbol{L}_k^\intercal \tag{33}$$

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 (31), (32) and (33)

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

where $\tilde{\boldsymbol{P}}_k = \boldsymbol{V}_k \boldsymbol{L}_k^{-\intercal}$ 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 23 shows that the columns of $\tilde{\boldsymbol{P}}_k$ are A-conjugate.

Lemma 23. *The columns of $\tilde{\boldsymbol{P}}_k$ are A-conjugate, in otherwords $\tilde{\boldsymbol{P}}_k^\intercal \boldsymbol{A} \tilde{\boldsymbol{P}}_k$ is diagonal.*

Proof. We compute

$$\begin{aligned} \tilde{\boldsymbol{P}}_k^\intercal \boldsymbol{A} \tilde{\boldsymbol{P}}_k &= \left(\boldsymbol{V}_k \boldsymbol{L}_k^{-\intercal} \right)^\intercal \boldsymbol{A} \left(\boldsymbol{V}_k \boldsymbol{L}_k^{-\intercal} \right) \\ &= \boldsymbol{L}_k^{-1} \left(\boldsymbol{V}_k^\intercal \boldsymbol{A} \boldsymbol{V}_k \right) \boldsymbol{L}_k^{-\intercal} \\ &= \boldsymbol{L}_k^{-1} \left(\boldsymbol{T}_k \right) \boldsymbol{L}_k^{-\intercal} \\ &= \boldsymbol{L}_k^{-1} \left(\boldsymbol{L}_k \boldsymbol{D}_k \boldsymbol{L}_k^\intercal \right) \boldsymbol{L}_k^{-\intercal} \quad (33) \\ &= \boldsymbol{D}_k \end{aligned}$$

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

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

where the elements of $\star$ are of no importance. 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. We can begin by computing a recurrance for $\tilde{\boldsymbol{y}}_k$ as follows

$$\tilde{\boldsymbol{y}}_k = \|\boldsymbol{r}_0\| \boldsymbol{D}_k^{-1} \boldsymbol{L}_k^{-1} \boldsymbol{e}_1^k$$

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

where $\boldsymbol{e}_i^k$ is the i^{th} unit vector with k dimensions. 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 $(k-1) \times (k-1)$ submatrix of $\boldsymbol{L}_k^{-\intercal}$. In effect, $\tilde{\boldsymbol{P}}_{k-1}$ is identical to the leading $k-1$ columns of

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

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

$$
\tilde{\boldsymbol{p}}_k = \boldsymbol{v}_k - l_{k-1}\tilde{\boldsymbol{p}}_{k-1}. \tag{34}
$$

Finally we can use

$$
\boldsymbol{x}_k = \boldsymbol{x}_0 + \tilde{\boldsymbol{P}}_k \tilde{\boldsymbol{y}}_k = \boldsymbol{x}_0 + \left[\tilde{\boldsymbol{P}}_{k-1}, \tilde{\boldsymbol{p}}_k\right] \begin{bmatrix} \tilde{\boldsymbol{y}}_{k-1} \\ \eta_k \end{bmatrix} = \boldsymbol{x}_0 + \tilde{\boldsymbol{P}}_{k-1}\tilde{\boldsymbol{y}}_{k-1} + \eta_k \tilde{\boldsymbol{p}}_k = \boldsymbol{x}_{k-1} + \eta_k \tilde{\boldsymbol{p}}_k \tag{35}
$$

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

$$
\boldsymbol{r}_k = b - \boldsymbol{A}\boldsymbol{x}_k = b - \boldsymbol{A}\left(\boldsymbol{x}_{k-1} + \eta_k \tilde{\boldsymbol{p}}_k\right) = (b - \boldsymbol{A}\boldsymbol{x}_{k-1}) - \eta_k \boldsymbol{A}\tilde{\boldsymbol{p}}_k = \boldsymbol{r}_{k-1} - \eta_k \boldsymbol{A}\tilde{\boldsymbol{p}}_k \tag{36}
$$

Altogether we are left with recurrences for $\boldsymbol{v}_k$ from Lanczos, $\tilde{\boldsymbol{p}}_k$ (34), the residual $\boldsymbol{r}_k$ (34), and for the approximate solution $\boldsymbol{x}_k$ (35). However, futher simplifications can be made to bring about a more efficient algorithm. Recall from Section 3.3 that $\boldsymbol{A}\boldsymbol{V}_k = \boldsymbol{V}_k \boldsymbol{T}_k + \boldsymbol{v}_{k+1}\boldsymbol{t}_k^{\intercal}$ where $\boldsymbol{t}_k = [0, 0, \ldots, 0, \beta_k]^{\intercal} \in \mathbb{R}^k$ meaning

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

This tells us that $\boldsymbol{r}_k$ is parallel to $\boldsymbol{v}_{k+1}$ and orthogonal to all $\boldsymbol{v}_i,\ 1 \leq i \leq k$. This further implies that $\boldsymbol{r}_k$ is orthogonal to all $\boldsymbol{r}_i,\ 1 \leq i \leq k-1$ since they are just $\boldsymbol{v}_i$ scaled by some constant factor. So replacing $\boldsymbol{r}_{k-1}$ with $\boldsymbol{v}_k/\eta_k$ 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 Lemma 23 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 $\langle \boldsymbol{r}_i, \boldsymbol{r}_j \rangle = 0,\ i \neq j$. Note that from our recurrence for $\boldsymbol{p}_k = \boldsymbol{r}_{k-1} + \beta_k \boldsymbol{p}_{k-1}$ we have

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

We can now find an expression for α_k as

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

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

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

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

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

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

Equating the two expressions for α_k affords

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

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 10.

Algorithm 10: 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 $\boldsymbol{x}_k$

The next few theorems pertain to the rate of convergence of the CG algorithm.

Theorem 24. *Let the CG iteration (Algorithm 10) be applied to a symmetric positive definite matrix problem $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. As long as the iteration has not yet converged (that is, $\boldsymbol{r}_{n-1} \neq 0$), the algorithm proceeds without divisions by zero, and we have the following identities of subspaces:*

$$\mathcal{K}_n = \text{l. s}\left\{\boldsymbol{x}_i\right\}_{i=1}^{n} = \text{l. s}\left\{\boldsymbol{p}_i\right\}_{i=0}^{n-1} = \text{l. s}\left\{\boldsymbol{r}_i\right\}_{i=0}^{n-1} = \text{l. s}\left\{\boldsymbol{A}^i\boldsymbol{b}\right\}_{i=0}^{n-1}. \tag{37}$$

Moreover the residuals are orthogonal,

$$\langle \boldsymbol{r}_n, \boldsymbol{r}_j \rangle = 0 \quad (j < n)$$

and the search directions are $\boldsymbol{A}-$conjugate [Tre97, page 295].

Proof. The orthogonality of the residuals has already been shown within the derivation of the CG algorithm. Seeing that the search directions are $\boldsymbol{A}-$conjugate is a very straight forward application of Lemma 23. To show (37) we can induct on k. From the initial guess $\boldsymbol{x}_0$ and the expression $\boldsymbol{x}_k = \boldsymbol{x}_{k-1} + \alpha_k \boldsymbol{p}_k$, using an induction argument it follows that $\boldsymbol{x}_k$ belongs to $\text{l. s}\left\{\boldsymbol{p}_i\right\}_{i=0}^{k-1}$. Similarly, from $\boldsymbol{p}_k = \boldsymbol{r}_{k-1} + \beta_k \boldsymbol{p}_{k-1}$ it follows that this belongs to $\text{l. s}\left\{\boldsymbol{r}_i\right\}_{i=0}^{n-1}$. Finally, from $\boldsymbol{r}_k = \boldsymbol{r}_{k-1} - \alpha_k \boldsymbol{A}\boldsymbol{p}_k$ that this belongs to $\text{l. s}\left\{\boldsymbol{A}^i\boldsymbol{b}\right\}_{i=0}^{n-1}$. □

It is fairly straight forward to confirm that CG minimizes error at each step.

Theorem 25. *Let the CG iteration be applied to a symmertic positive definite matrix $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. If the iteration has not already converged (that is, $\|\boldsymbol{r}_{k-1}\| > 0$), then $\boldsymbol{x}_k$ is the unique point in $\mathcal{K}_k$ that minimizes $\|\boldsymbol{x}^\star - \boldsymbol{x}_k\|_{\boldsymbol{A}}$. The convergence is the monotonic,*

$$\|\boldsymbol{x}^\star - \boldsymbol{x}_k\|_{\boldsymbol{A}} \leq \|\boldsymbol{x}^\star - \boldsymbol{x}_{k-1}\|_{\boldsymbol{A}}$$

and $\|\boldsymbol{x}^\star - \boldsymbol{x}_k\|_{\boldsymbol{A}} = 0$ is achieved for some $k \leq n$ [Tre97, page 296].

Proof. From Theorem 24 we know that $\boldsymbol{x}_k$ belongs to $\mathcal{K}_k$. To show that it is the unique point in $\mathcal{K}_k$ that minimizes the error, consider an arbitrary point $\boldsymbol{z} = \boldsymbol{x}_k - \Delta\boldsymbol{z} \in \mathcal{K}_k$, with error $\boldsymbol{e} = \boldsymbol{x}_\star - \boldsymbol{z} = \boldsymbol{e}_k + \Delta\boldsymbol{x}$. Note in this context $\boldsymbol{e}_k = \boldsymbol{x}_\star - \boldsymbol{x}_k$, and does *not* represent standard basis vectors. We calculate

$$\begin{aligned} \|\boldsymbol{e}\|_{\boldsymbol{A}}^2 &= (\boldsymbol{e}_k + \Delta\boldsymbol{x})^\intercal \boldsymbol{A} (\boldsymbol{e}_k + \Delta\boldsymbol{x}) \\ &= \boldsymbol{e}_k^\intercal \boldsymbol{A}\boldsymbol{e}_k + (\Delta\boldsymbol{x})^\intercal \boldsymbol{A} (\Delta\boldsymbol{x}) + 2\boldsymbol{e}_k^\intercal \boldsymbol{A} (\Delta\boldsymbol{x}). \end{aligned}$$

The final term of this expression is $2\boldsymbol{r}_k^\intercal (\Delta\boldsymbol{x})$, an inner product of $\boldsymbol{r}_k$ with a vector in $\mathcal{K}_k$, and by Theorem 25, any such inner product is zero. Effectively, this means

$$\|\boldsymbol{e}\|_{\boldsymbol{A}}^2 = \boldsymbol{e}_k^\intercal \boldsymbol{A}\boldsymbol{e}_k + (\Delta\boldsymbol{x})^\intercal \boldsymbol{A} (\Delta\boldsymbol{x}).$$

Only the second of these terms depends on $\Delta\boldsymbol{x}$, since $\boldsymbol{A}$ is positive definite, that term is greater than or equal to 0, attaining the value of 0 if and only if $\Delta\boldsymbol{x} = \boldsymbol{0}$, that is, $\boldsymbol{x}_k = \boldsymbol{x}$. Thus $\|\boldsymbol{e}\|_{\boldsymbol{A}}$ is minimal if and only if $\boldsymbol{x}_k = \boldsymbol{x}$, as claimed. □

As seen in (19) Krylov subspace methods, in some sense, aim to find a polynomial $p_n \in P_n$ which minimizes $\|p_n(\boldsymbol{A})\| = \|\sum_{k=0}^{n-1} \alpha_{k+1} \boldsymbol{A}^k \boldsymbol{r}_0\|$, where P_n is the set of polynomials with degree less than or equal to n with $p(\boldsymbol{0}) = \mathbb{1}$. In the context of the CG algorithm, this is equivalent to finding a polynomial $p_n \in P_n$ which instead minimizes

$$\|p_n(\boldsymbol{A})\, \boldsymbol{e}_0\|_{\boldsymbol{A}}. \tag{38}$$

From Theorem 24, we can derive the following convergence theorem.

Theorem 26. *If the CG iteration has not already converged before step k (that is, $\|\boldsymbol{r}_{k-1}\| > 0$), then (38) has a unique solution $p_n \in P_n$, and the iterate $\boldsymbol{x}_k$ has error $\boldsymbol{e}_k = p_n(\boldsymbol{A})\,\boldsymbol{e}_0$ for this same polynomial p_n. Consequently we have*

$$\frac{\|\boldsymbol{e}_k\|_{\boldsymbol{A}}}{\|\boldsymbol{e}_0\|_{\boldsymbol{A}}} = \inf_{p\in P_n} \frac{\|p(\boldsymbol{A})\,\boldsymbol{e}_0\|_{\boldsymbol{A}}}{\|\boldsymbol{e}_0\|_{\boldsymbol{A}}} \leq \inf_{p\in P_n} \max_{\lambda\in\Lambda(\boldsymbol{A})} |p(\lambda)| \tag{39}$$

[Tre97, page 298].

Proof. From Theorem 24 it follows that $\boldsymbol{e}_k = p(\boldsymbol{A})\,\boldsymbol{e}_0$ for some $p \in P_n$. The equality in (39) is a consequence of this and Theorem 25. As for the inequality in (39), if $\boldsymbol{e}_0 = \sum_{j=1}^m a_j \boldsymbol{v}_j$ is an expansion of $\boldsymbol{e}_0$ in orthonormal eigenvectors of $\boldsymbol{A}$, then we have $p(\boldsymbol{A})\,\boldsymbol{e}_0 = \sum_{j=1}^m a_j p(\lambda_j)\,\boldsymbol{v}_j$ and thus

$$\|\boldsymbol{e}_0\|_{\boldsymbol{A}}^2 = \sum_{j=1}^m a_j^2 \lambda_j, \qquad \|p(\boldsymbol{A})\,\boldsymbol{e}_0\|_{\boldsymbol{A}}^2 = \sum_{j=1}^m a_j^2 \lambda_j \left(p(\lambda_j)\right)^2.$$

These identities indicate $\|p(\boldsymbol{A})\,\boldsymbol{e}_0\|_{\boldsymbol{A}}^2 / \|\boldsymbol{e}_0\|_{\boldsymbol{A}}^2 \leq \max_{\lambda\in\Lambda(\boldsymbol{A})} |p(\lambda)|^2$, which implies the inequality in question. □

3.6. **Minimum Residual.** In contrast to CG, MINRES is applicable to a wider range of linear systems, namely symmertic indefinite systems. From Section 3.4 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

$$\boldsymbol{x}_k = \operatorname*{arg\,min}_{\boldsymbol{x}\in\boldsymbol{x}_0+\mathcal{K}_k} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2. \tag{40}$$

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, similar to Theorem 22. So under the conditions of MINRES, using a similar argument for CG, $\boldsymbol{x}_k$ can be expressed as $\boldsymbol{x}_k = \boldsymbol{x}_0 + \boldsymbol{V}_k\boldsymbol{y}_k$. Employing (29) produced by the Lanczos algorithm in 3.3 we can manipulate our optimality condition (40) as follows

$$\begin{aligned}
\boldsymbol{x}_k &= \operatorname*{arg\,min}_{\boldsymbol{x}\in\boldsymbol{x}_0+\mathcal{K}_k} \|\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}\|_2 \\
\iff \boldsymbol{y}_k &= \operatorname*{arg\,min}_{\boldsymbol{y}\in\mathbb{R}^k} \|\boldsymbol{A}(\boldsymbol{x}_0 + \boldsymbol{V}_k\boldsymbol{y}) - \boldsymbol{b}\|_2 \\
&= \operatorname*{arg\,min}_{\boldsymbol{y}\in\mathbb{R}^k} \|-(\boldsymbol{b} - \boldsymbol{A}\boldsymbol{x}_0) + \boldsymbol{A}\boldsymbol{V}_k\boldsymbol{y}\|_2 \\
&= \operatorname*{arg\,min}_{\boldsymbol{y}\in\mathbb{R}^k} \|\boldsymbol{V}_k\boldsymbol{T}_{k+1,k}\boldsymbol{y} - \boldsymbol{r}_0\|_2 \\
&= \operatorname*{arg\,min}_{\boldsymbol{y}\in\mathbb{R}^k} \left\|\boldsymbol{T}_{k+1,k}\boldsymbol{y} - \boldsymbol{V}_k^\intercal\boldsymbol{r}_0\right\|_2 \\
&= \operatorname*{arg\,min}_{\boldsymbol{y}\in\mathbb{R}^k} \|\boldsymbol{T}_{k+1,k}\boldsymbol{y} - \beta_0\boldsymbol{e}_1\|_2
\end{aligned}$$

[Gre97, page 43]. Under the general project method Algorithm 9, this affords the following high-level description of the MINRES algorithm [Tre97, page 268].

Algorithm 11: High-Level MINRES

output: An approximation of $\boldsymbol{x}^*$, $\boldsymbol{x}_k$.

for $k = 1, \ldots$ **until** *convergence* **do**
  Lanczos-Step $(\boldsymbol{A}, \boldsymbol{v}_k, \boldsymbol{v}_{k-1}, \beta_k) \to \alpha_k, \beta_{k+1}, \boldsymbol{v}_{k+1}$
  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 an inexpensive $\mathcal{O}(n)$ Householder reflection [Tre97, page 268]. Computing the QR factorisation of $\boldsymbol{T}_{k,k-1}$ yields

$$\boldsymbol{Q}_k \boldsymbol{T}_{k,k+1} = \begin{bmatrix} \boldsymbol{R}_k \\ 0 \end{bmatrix} = \begin{bmatrix} \gamma_1^{(1)} & \delta_2^{(1)} & \varepsilon_3^{(1)} & & & \\ & \gamma_1^{(2)} & \delta_3^{(2)} & \varepsilon_4^{(1)} & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & \varepsilon_k^{(1)} \\ & & & & \ddots & \gamma_k^{(2)} \\ & & & & & \delta_k^{(2)} \\ & & & & & 0 \end{bmatrix} \tag{41}$$

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

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

[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 an alternative way of writing $\boldsymbol{Q}_k \boldsymbol{T}_{k,k+1}$ is $\boldsymbol{Q}_{k,k+1} \cdot \boldsymbol{Q}_{2,3}\boldsymbol{Q}_{1,2}\boldsymbol{T}_{k,k+1}$. 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 product

$$\begin{bmatrix} c_k & s_k \\ s_k & -c_k \end{bmatrix} \begin{bmatrix} \gamma_k^{(1)} & \delta_{k+1}^{(1)} & 0 \\ \beta_{k+1} & \alpha_{k+1} & \beta_{k+1} \end{bmatrix}. \tag{42}$$

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 (42) 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

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

which can be used to formulate subproblems [Cho07, page 25]. From the new optimality condition it becomes 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

$$\boldsymbol{R}_k^\intercal \boldsymbol{D}_k^\intercal = \boldsymbol{V}_k^\intercal \tag{43}$$

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

$$\begin{aligned} \boldsymbol{x}_k &= \boldsymbol{V}_k \boldsymbol{y}_k = \boldsymbol{D}_k \boldsymbol{R}_k \boldsymbol{y}_k = \boldsymbol{D}_k \boldsymbol{t}_k \\ &= [\boldsymbol{D}_{k-1}, \boldsymbol{d}_k] \begin{bmatrix} \boldsymbol{t}_{k-1} \\ \tau_t \end{bmatrix} \\ &= \boldsymbol{x}_{k-1} + \tau_k \boldsymbol{d}_k. \end{aligned} \tag{44}$$

The d_j in (44) 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 (43) [CHO11, page 4]. The final form of the MINRES in presented in Algorithm 12.

Algorithm 12: 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 $\boldsymbol{x}_k$

4. Applications and Results

So far we have discussed a number of different approximation techniques that can be to speed up some of the bottle necks within our Gaussian Process regression and classification algorithms. In this chapter, we shall see how they can be used within the naive implementations from **??** to provide faster processing, at the expense of a small amount of accuracy. We shall also go through an extensive treatment of how the various methods reviewed in previous chapters were implemented and how experiments were set up so that comparisons between them can be made and we can determine the most successful model.

4.1. Gaussian Processes Prediction Reviewed.

In Chapter **??**, a naive implementation for Gaussian Process prediction was presented in **??**. While this does provide a simple and convenient way to produce predictions for a regression task, it is not all smooth sailing after this. Unfortunately, there are a number of problems with this algorithm in terms of scalability. For convenience, this algorithm has been restated below but this time with the various bottlenecks highlighted in red.

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 $\mathbb{V}[f_\star]$.

1 $\boldsymbol{L} = \text{cholesky}\left(\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}} + \sigma_n^2 \mathbb{1}_{n\times n}\right)$

2 $\boldsymbol{\alpha} = \text{lin-solve}\left(\boldsymbol{L}^\intercal, \text{lin-solve}\left(\boldsymbol{L}, \boldsymbol{y}\right)\right)$

3 $\overline{f_\star} = \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}} \boldsymbol{\alpha}$

4 $\boldsymbol{v} = \text{lin-solve}\left(\boldsymbol{L}, \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{X}}\right)$

5 $\mathbb{V}[f_\star] = \boldsymbol{K}_{\boldsymbol{x}_\star \boldsymbol{x}_\star} - \boldsymbol{v}^\intercal \boldsymbol{v}$

6 **return** $\overline{f_\star}, \mathbb{V}[f_\star]$

To start computing the kernel matrix, $\boldsymbol{K}_{\boldsymbol{X}\boldsymbol{X}}$, on line 1 carries an $\mathcal{O}\left(n^2\right)$ runtime since $\mathcal{O}\left(n^2\right)$ pairwise kernel evaluations must be made. Moreover, solving linear systems using a Cholesky decomposition (seen on lines $1, 2$ and 4) will incur a runtime of $\mathcal{O}\left(n^3\right)$. While this quadratic scaling is not a significant issue for smaller datasets, this will scale very poorly as most modern desktops are unable to feasibly perform training with datasets having more than 10^5 samples [WJMaSKaAGW21, page 2]. The Gaussian Process classifier from **??** suffers for the exact same reasons as GPR. To mitigate the computational burden of these bottle necks, we can replace these procedures with their inexact counterparts discussed in Chapters 1,2 and 3.

To start, computing the kernel matrix can be done using either the Nystrom method or the RFF technique, both of which provide better asymptotic runtimes. Similarly, the Cholesky decomposition and linear solves can be replaced with either CG or MINRES to improve runtime performance. This makes sense when approximating the kernel matrix as the Cholesky decomposition provides an almost-exact solution to solving these linear system, which is rather counterintuitive if approximations are used earlier on within the algorithm.

4.2. Experimental Setup.

Each method was implemented in python3 and run using the python3.8.5 interpreter distributed by the official python website. Various scientific computing python libraries, such

as numpy, pandas and scipy, were used to implement methods that enhanced runtime performance. A just-in-time compiler was also employed to improve the performance of the FWHT and Krylov-Subspace methods as fast runtime is critically important for the success of both these methods. Experiments were carried out on a variety of different datasets listed in Table 3.

TABLE 3. Descriptions and sources for each of the datasets used in experiments.

Name	*Description*	d	n	*Source*
3DSN	The 3D Spatial Network (3DSN) dataset was constructed by adding elevation information to a 2D road network in North Jutland, Denmark (covering a region of $185 \times 135 km^2$).	2	2000	UCI
Abalone	Physical measurements of abalones.	7	4177	UCI
magic04	Simulated registration of high energy gamma particles in a ground-based atmospheric Cherenkov gamma telescope.	10	19020	UCI
Wine	Chemical measurements of wine.	11	4898	UCI
Temp	Weather station data from rural Queensland.	1	11324	Qld Gov
Stocks	Daily stock prices spanning 2000 to 2019.	4	4904	tiskw
quadratic	A dataset artificially constructed where samples where drawn from a Gaussian process with a mean function of $(x, y) \mapsto x^2 + y^2$ with a RBF kernel.	2	5000	NA

4.2.1. *Kernel Martix Approximation Testing.* To test the various kernel matrix approximation techniques, each method was used to construct an estimate of the actual kernel matrix with varying sample sizes. The definition of samples changes depending of which family of approximation technique is considered. For the Nystrom technique, the number of samples refers to s, that is the number of columns sampled. In the case of the RFF technique the samples refer to D, that being the dimension of the constructed feature space. The Nystrom technique was implemented according to [PDaMWM05] and RFF according to [Rah08] and [Liu21]. Methods were then made to compute approximations for kernel matrices for each of the above data sets for a fixed σ and sample size. This was repeated 30 times for each dataset. For every method the relative Frobenius error $\left\|\boldsymbol{K_{XX}} - \widehat{\boldsymbol{K}}_{\boldsymbol{XX}}\right\|_F / \|\boldsymbol{K_{XX}}\|_F$ and relative infinity error $\left\|\boldsymbol{K_{XX}} - \widehat{\boldsymbol{K}}_{\boldsymbol{XX}}\right\|_\infty / \|\boldsymbol{K_{XX}}\|_\infty = \left\|\boldsymbol{K_{XX}} - \widehat{\boldsymbol{K}}_{\boldsymbol{XX}}\right\|_\infty$ (in the case of the RBF kernel) was recorded, using the notation that $\widehat{\boldsymbol{K}}_{\boldsymbol{XX}}$ represents an approximated kernel matrix. The values of σ employed here were 0.1, 1.0 and 10.0. A σ value of 2.1 was also used to compare with results from [PDaMWM05]. Additionally, computation time and memory usage for each kernel approximation was also recorded. It should be noted, however, that the time and memory spent on constructing probabilites for the Nystrom methods are *not* included in the kernel construction time and have instead been recorded separately in Figures 35-40. This is because the probabilites are typically known prior to kernel matrix construction and are simply used to provide an approximation to the kernel matrix instead on needing to save the entire data kernel matrix for a new prediction. Thus, they act as an efficient means to quickly save and

rebuild the data kernel matrix. The errors, time and memory were averaged across the repetitions for each experiment.

To understand how well each kernel matrix approximation method performs in terms of providing predictions, each method was required to train a Gaussian process with $4/5^{ths}$ of each data set and then predict the remaining $1/5^{th}$. Predictions for each data set were repeated 15 times across various sample sizes. The mean square error (MSE) and classification error was captured for regression and classification tasks respectively. To make comparisons between different approximation methods as fair as possible, the matrix $\left(\widehat{\boldsymbol{K}}_{\boldsymbol{XX}} + \sigma_n^2 \mathbb{1}_{n\times n}\right)^{-1}$ was directly inverted to eliminate any error that an in-exact linear system solver might introduce. To do this efficiently, since both the Nystrom and RFF methods produce decompositions to their approximations of the form $\widehat{\boldsymbol{K}}_{\boldsymbol{XX}} = \boldsymbol{U}\boldsymbol{W}^{\dagger}\boldsymbol{U}^{\intercal}$ we can make use of the matrix inversion lemma (see **??**) to compute

$$\begin{aligned}\left(\widehat{\boldsymbol{K}}_{\boldsymbol{XX}} + \sigma_n^2 \mathbb{1}_{n\times n}\right)^{-1} &= \left(\boldsymbol{U}\boldsymbol{W}\boldsymbol{U}^{\intercal} + \sigma_n^2 \mathbb{1}_{n\times n}\right)^{-1} \\ &= \sigma_n^{-2}\mathbb{1}_{n\times n} + \sigma_n^{-2}\mathbb{1}_{n\times n}\boldsymbol{U}\left(\boldsymbol{W}^{\dagger} + \boldsymbol{U}^{\intercal}\sigma_n^{-2}\mathbb{1}_{n\times n}\boldsymbol{U}\right)^{-1}\boldsymbol{U}^{\intercal}\sigma_n^{-2}\mathbb{1}_{n\times n}.\end{aligned}$$

Only a single value of σ was used to create predictions for each dataset. The value of σ chosen corresponds to the best value of σ out of $0.1, 2.1, 1.0$ and 10.0 when an exact GP algorithm was used to provide predictions (a value of 2.1 was used to help compare with the results from [PDaMWM05]). The time and memory of performing the matrix inverse using the above procedure was recorded. Again, errors, time and memory were averaged across the repetitions for each different sample.

4.2.2. *Krylov Subspace Methods Approximation Testing.* Similar to the kernel matrix approximation setup, to see how well CG and MINRES compare in providing predictions, each method was used to solve the linear system

$$\left(\boldsymbol{K}_{\boldsymbol{XX}} + \sigma_n^2 \mathbb{1}_{n\times n}\right)\boldsymbol{\alpha} = \boldsymbol{y}$$

within the GPR algorithm and

$$\left(\boldsymbol{K}_{\boldsymbol{XX}} + \boldsymbol{W}^{-1}\right)\boldsymbol{\alpha} = \boldsymbol{K}_{\boldsymbol{x}_\star\boldsymbol{X}}^{\intercal}$$

within the GPC algorithm in terms of $\boldsymbol{\alpha}$. For consistency, predictions for each data set was repeated 15 times across various samples. The mean square error (MSE) and classification error was captured for regression and classification tasks respectively. In an effort to provide a fair comparison between the different linear solvers, the kernel matrix was constructed in a manner that ensured no error was introduced through any other part of the prediction process. As with the kernel matrix approximation prediction set up, only a single value of σ was used to create predictions for each dataset, that being the value of σ that provided the best predictions results using exacts methods. Each method was used to train a Gaussian process with $4/5^{ths}$ of each data set and required to predict the remaining $1/5^{th}$ over all the different datasets and was repeated 15 times for each dataset with a varying number of maximum iterations. The usual metrics such as prediction error, time and memory usuage were recorded for every experiment and averaged across repetitions for each dataset and method.

4.3. **Discussion.**

4.3.1. *Kernel Matrix Approximation.* Starting with the performance of the Nystrom methods, assessing Figures 23 to 34, overall the rls method is superior for sampling distributions as it is virtually always among the top three methods for any combination of dataset, k or σ. Interestingly, non-uniform sampling techniques performed better with datasets for which the spectrum of the corresponding kernel matrix was also non-uniform. For example, the spectrum for the kernel matrices of the magic04 and Stocks dataset (see Figures 7 and 8) are relatively uniform and from Figures 27 and 29 we find that the non-uniform sampling techniques give rather poor approximations and generally behave just as poorly, or even worse, than the naive uniform sampling distribution. In contrast, non-uniform methods very much shine in the case of the 3DSN and Wine datasets where some of the better methods could provide almost exact kernel matrix approximations (refer to Figures 23, 24, 33 and 34) after a few thousand samples. This phenomena is likely due to the fact that the spectrum of a matrix decays faster when it is less uniform, meaning that the matrix can be better expressed as a low-rank approximation. Evidently, the majority of non-uniform sampling distributions are intelligent enough to select columns that are best representative in this low rank approximation, most of the time. What is especially interesting about the Abalone dataset is, despite the rls sampling distribution being almost uniform (compared to other sampling distributions, see Figures 12 and 18), this method seems to perform significantly better than uniform selection, shown in Figure 25. The small amount of variation in the rls sampling distribution presumably gives it an edge over the uniform method in selecting columns that better provides information to the lower rank approximation.

Turning our attention to the RFF methods, we find that the ORF and SORF methods do not really live up to their acclaimed theoretical error bounds. While the ORF and SORF methods do occasionally produce better errors over the standard transformation matrix sampling (see Figures 53 to 64), it is hard to justify using either of these methods as neither offer any obvious advantage in terms of time and memory saving (see Figures TODO). This may come as a surprise given the SORF method, even with a JIT compiler, was used to speed up various parts of the algorithm. This dissonance between the theoretical bounds presented in Table 2 and these "real world" experiments is that generating Gaussian variables is still, in practice, quicker than the asymptotically more efficient construction of SORF's transform matrix. When fed large enough datasets, the SORF method is likely to overtake the "basic" sampling method proposed by Rami and Recht. Unfortunately, due to the memory constraints of the computers used for this project's experimental, using such large data sets could not be conducted to test this hypothesis.

To make a fair comparison in kernel matrix approximation between the RFF and Nystrom methods, the time that each method used to build an approximation was graphed alongside the corresponding relative error. In this setup, we can think about each method as having some sort of time budget and that its task is to produce the best approximation it can within this limit, regardless of how many samples it requires or how big of a feature space it uses. Comparing the methods in this manner the Nystrom family is superior in lowering the Frobenius error in the approximations it produces, while the RFF methods provide approximations with smaller infinity errors (see Figures 71 to 82). In other words, Nystrom methods produce approximations where entries are, *on average*, closer to the true matrix compared to RFF methods. RFF methods, on the otherhand, produce approximations where entries are *in the worst case* closer to the true matrix compared to the Nystrom methods. This makes sense since much of the theory in the Nystrom methods was aimed toward lowering Frobenius errors, while RFF methods focuses on

lowering infinity errors. One final point worth mentioning is that RFF methods seemed to require much larger amounts of memory to produce their approximations, sometimes double or even triple the amount of memory the of Nystrom methods used (refer to Figures TODO).

Next, two methods from the Nystrom class and one method from the RFF class, that we deemed to be the best among their respective families, were used within the Gaussian Process algorithm (replacing exact kernel matrix creation) to form predictions. The rls and data columns methods were chosen to represent the best of the Nystrom family while the Rami and Recht's "basic" RFF method was selected to represent the best of the RFF techniques. Similar to relative error comparisons, the time each method required to complete an approximation was graphed along side the corresponding prediction error. The results, suggest that there was no clear winner. Generally speaking, Rami and Recht's RFF method delivered the best predictions with smaller time budgets were the rls generally caught up and overtook it when a larger time budget was allowed. The data columns method, tends to yield the poorest prediction errors among the selected methods. Thus, the RFF method would probably be the best method to use in practice for two main reasons. First, if one was to select an inexact method for kernel matrix production, it most likely means that there is not a large time budget. In this case RFF's ability to produce the best approximation within shorter timeframes makes it the best option. Second, the time taken to construct probability distributions used in the Nystrom methods are not included in Figure TODO (and can instead be found in Figures 35-40) making it harder to justify using the rls method if the probability distribution is not already available since a much larger time budget would be required to deliver the same level of prediction accuracy.

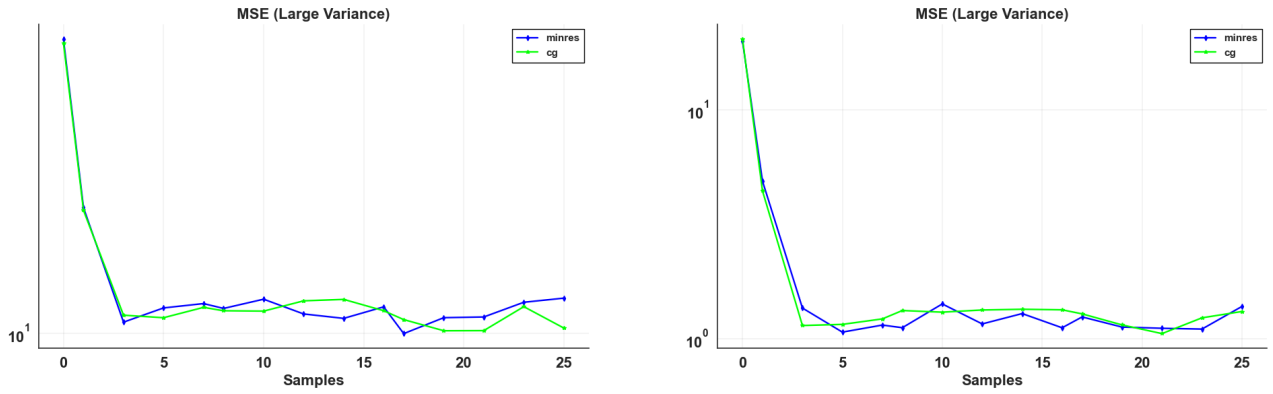
4.3.2. *Krylov Subspace Methods.* The CG and MINRES methods where also used within the Gaussian Process algorithm to see which one of the two would provide better predictions. Historically CG is preferred over MINRES for Gaussian process prediction since it is typically a superior linear system solver when dealing with PSD matrices. There are, however, a number of circumstances where MINRES can actually outperform CG. Take for example the work of Pleiss etal. [Ple20] which leverages a variant of MINRES to assist in the computation kernel matrix square roots. We were also wondering if MINRES is better suited as a linear solver within the context of Gaussian processes. In our experimentation, the number of iterations that each method used to form an approximation was graphed along side the corresponding prediction error produced. Differences in execution time per iteration between the two methods were nominal. Looking at the results in Figures TODO, it appears that the MINRES method performs just as well and, occasionally, even better than CG. This was apparent with the prediction errors produced from the abalone and 3DSN datasets. In particular, the MINRES method affords better results even on the quadratic dataset. The quadratic dataset is an artificial dataset constructed using a Gaussian process with a quadratic mean function $x^2 + y^2$ and some amount of variance. Its main purpose was to ascertain if the applied method was prone to overfitting. The fact that, overall, MINRES yielded lower errors indicates that it is also more robust against overfitting compared to CG. To understand why MINRES may produce smaller errors for regression tasks, recall that the true empirical mean square error is computed as

$$\|\boldsymbol{K}_{\boldsymbol{X}_\star \boldsymbol{X}} \boldsymbol{\rho} - \boldsymbol{y}_\star\|_2^2$$

where $\boldsymbol{\rho}$ is our best estimate for a solution to the linear system $\left[\boldsymbol{K_{XX}} + \sigma_n^2 \mathbb{1}_{n\times n}\right] \boldsymbol{\rho} = \boldsymbol{y}$, up to a scaling constant. The MINRES method seeks to solve this linear system by solving an empirical version of the MSE up to a noise value of sigma, that is, $\boldsymbol{\rho}$ is chosen so that

$$\left\| \left[\boldsymbol{K_{XX}} + \sigma_n^2 \mathbb{1}_{n\times n}\right] \boldsymbol{\rho} - \boldsymbol{y} \right\|_2^2$$

is minimized. In contrast, CG seeks vectors that successively minimizes the power norm of $\boldsymbol{\rho}$; something which we are not so interested in when comparing and contrasting test accuracies across different methods. While both MINRES and CG will eventually yield the same solution, the success of MINRES is likely due to its ability find early estimates of $\boldsymbol{\rho}$ that more directly lowers the empirical MSE. For small values of σ_n^2, will also correspond in a lower true MSE. If the above analysis is correct, then MINRES is really only advantageous when the value of sigma is small so that less weight is placed on the model's prior. Indeed, when the variance was artificially increased for both the abalone and quadratic dataset prediction, the errors between the CG and MINRES where much closer.



(A) (B)

FIGURE 4. Large variances place greater emphasis on the regression model's prior. This makes the true and empirical MSE much more dissimilar causing MINRES to lose its edge over CG. The effect is best seen on the abalone (right panel) and the quadratic dataset (left panel).

4.4. **Conclusion.** The aim of this thesis was to explore how various approximation techniques could be used within the Gaussian Process algorithm to speed up prediction time. Chapter 1 focuses on what kernel reproducing Hilbert spaces and how the kernel trick can be employed as a clever means to afford non-linear estimations while avoiding non-linear solvers. Next, Gaussian processes were defined and shown how they could be used as an instrument for machine learning to facilitate classification and regression tasks. However, when implemented naively, Gaussian process scale very poorly with the number of samples on account of two predominate bottlenecks within the prediction process. The first bottleneck arises from having to produce the kernel matrix and the second from solving linears involving said kernel matrix.

The remainder of the thesis focuses on popular estimation techniques to accelerate the prediction process. Chapter 2 looked at the Nystrom method to speed up kernel matrix approximation. Specifically, a

common sampling technique variant was studied in which various sampling distributions where used to construct a sketching matrix. We also devised novel sampling distributions that depended only on the data matrix and did not require any information about the kernel matrix. In Chapter 3, another (and more recent) approach to kernel matrix approximation was reviewed, that being the Random Fourier Feature (RFF) method. In this chapter various transformation matrices were considered for the construction of a Gaussian random matrix. Finally, in Chapter 4, a number of Krylov subspace methods were surveyed to seek alternative ways of solving linear systems within Gaussian process predictions. The particular Krylov subspace methods studied were the conjugate gradient (CG) and minimum residual (MINRES) algorithms.

Chapter 5, summarized our extensive experimental analysis to determine which methods delivered the best results. For the kernel estimation techniques, each method was required to create a its own approximation for the kernel matrices of various datasets where the relative infinity and Frobenius error was captured to measure the quality of approximations. Moreover, selected kernel methods were used within the Gaussian Process algorithm to provided predictions for the same datasets. Similarly, Krylov subspace methods were used in place of the Cholesky decomposition in Algorithm **??** to afford a comparative prediction.

When looking at a kernel matrix construct in isloation, 5 different Nystrom column sampling distributions where compared with 3 RFF transformation matrix construction methods. For Nystrom methods, it was found that some of the more sophisticated sampling distributions provided superior results for kernel matrices with non-uniform spectrums. Surprisingly, for RFF methods, the use of sophisticated techniques used to construct the transformation did not provide much of an advantage over basic methods. In terms of Gaussian process predictions, RFF methods delivered better errors with small time budgets over Nystrom methods.

When comparing CG and MINRES, we found that MINRES was consistently more competitive, if not better, than CG, even when paired with RFF for regression tasks. This may seem surprising given the ubiquitous preference of CG over any other linear solver in many Gaussian process software libraries. We believe MINRES's superior performance is due to its ability to more directly lower the Euclidean distance between the predictions and the true outputs.

Looking forward, some of the findings discovered in this thesis have, to our knowledge, not been reported elsewhere. As a result, we intend to publish these findings in a reputable journal given their obvious appeal to the wider scientific computing community. In terms of research, many of the datasets used in this thesis were far too small to observe the asymptotic benefits of using approximation methods over most of the exact methods. It would be interesting to determine how well these techniques scale on very large datasets. In particular, our discoveries are likely to benefit the agricultural sector as more data is collected over the forthcoming years to perform crop analysis on. Another, direction future research could be taken is the application of the approximation techniques applied to multi-output or multi-task Gaussian process models. In many machine learning scenarios we may want to predict multiple outputs using the same set of inputs. As an example, remote sensing researchers attempt to predict the intensity of multiple bands of light reflecting off farm land to give an indication of crop growth. Different versions of the Gaussian processes algorithm exist to predict multiple outputs simultaneously [Bon07];

however, they also suffer from the same bottlenecks as single output Gaussian processes. It would be intriging to learn whether the approximation techniques studied in this thesis could potentially improve the prediction time of multi-output Gaussian Processes.

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.

[NdF13] Nando de Freitas, *University of British Columbia CPSC 540, Lecture notes in Machine Learning*, University of British Columbia, 2013.

[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.

[WJMaSKaAGW21] Wesley J. Maddox and Sanyam Kapoor and Andrew Gordon Wilson, *When are Iterative Gaussian Processes Reliably Accurate?*, CoRR **abs/2112.15246** (2021), available at 2112.15246.

[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).

[Ala15] Ahmed and Mahoney Alaoui Michael W, *Fast Randomized Kernel Ridge Regression with Statistical Guarantees*, Advances in Neural Information Processing Systems, 2015.

[Kar10] Noureddine El Karoui, *THE SPECTRUM OF KERNEL RANDOM MATRICES*, The Annals of statistics **38** (2010), no. 1, 1–50 (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.

[Hor13] Roger A and Johnson Horn Charles R, *Matrix analysis / Roger A. Horn, Charles R. Johnson.*, 2nd ed., Cambridge University Press, New York, 2013 (eng).

[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).

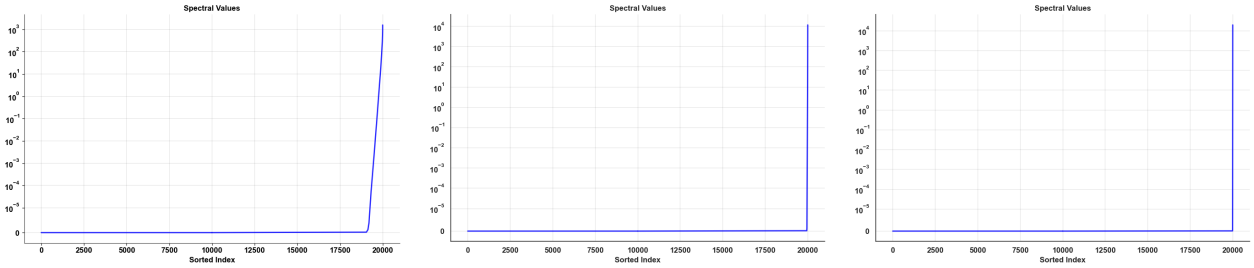
[Ple20] Geoff and Jankowiak Pleiss Martin and Eriksson, *Fast Matrix Square Roots with Applications to Gaussian Processes and Bayesian Optimization*, arXiv, 2020.

[Bon07] Edwin V and Chai Bonilla Kian and Williams, *Multi-task Gaussian Process Prediction* (J. Platt and D. Koller and Y. Singer and S. Roweis, ed.), Vol. 20, Curran Associates, Inc., 2007.

Appendix A. Supplementary Results

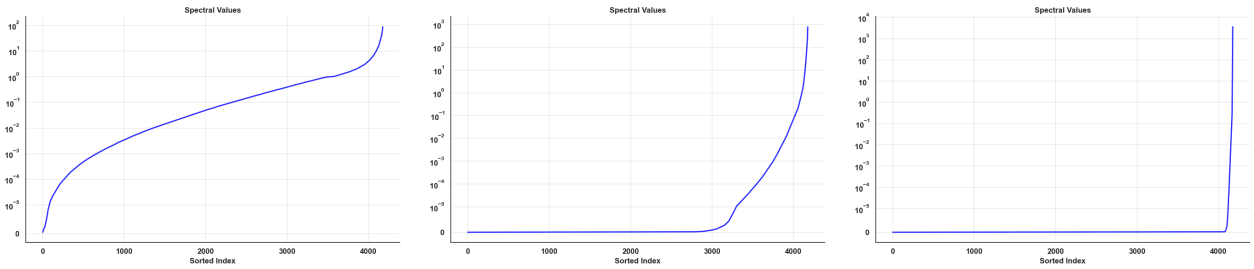
Additional results that may have not been included in the main text for conciseness.

A.1. **Gram Matrix Spectral Values.**



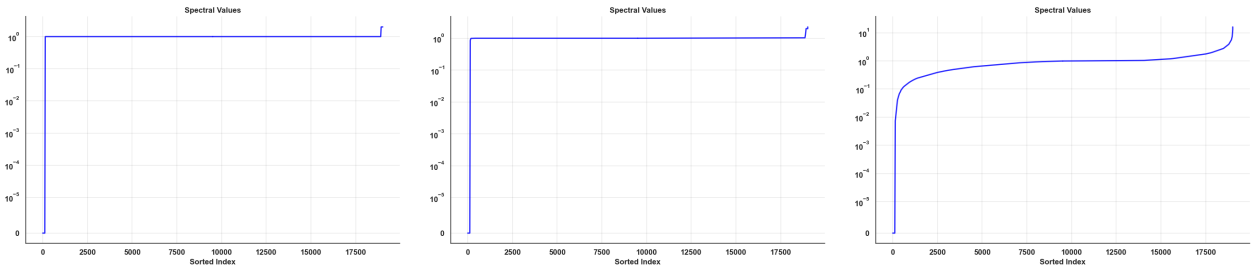
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 5. Spectral values for 3D-spatial network data.



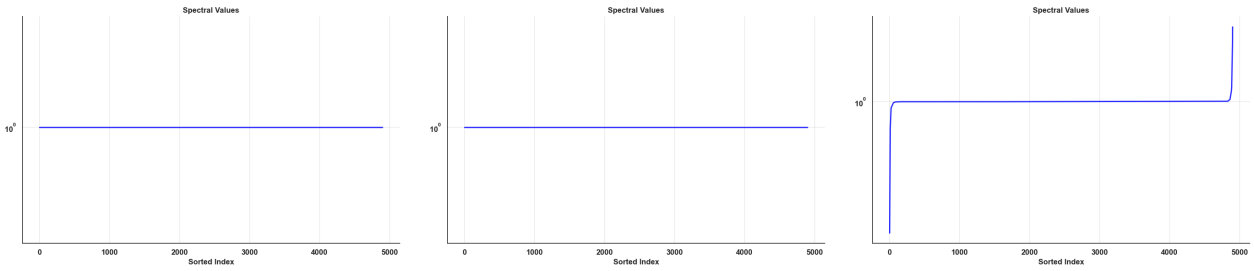
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 6. Spectral values for abalone data.



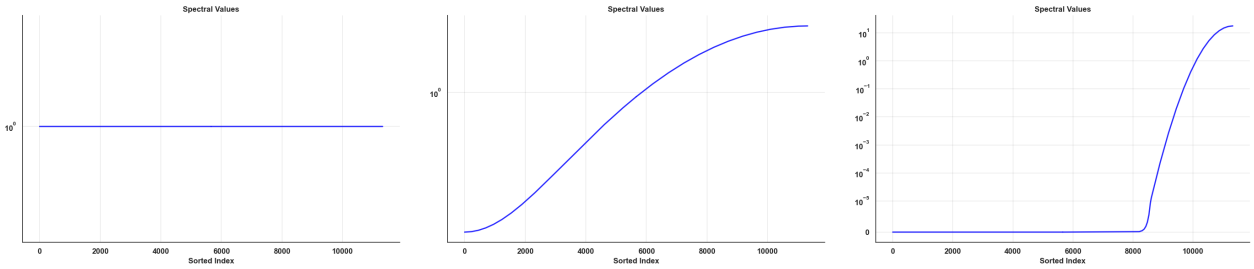
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 7. Spectral values for magic04 data.



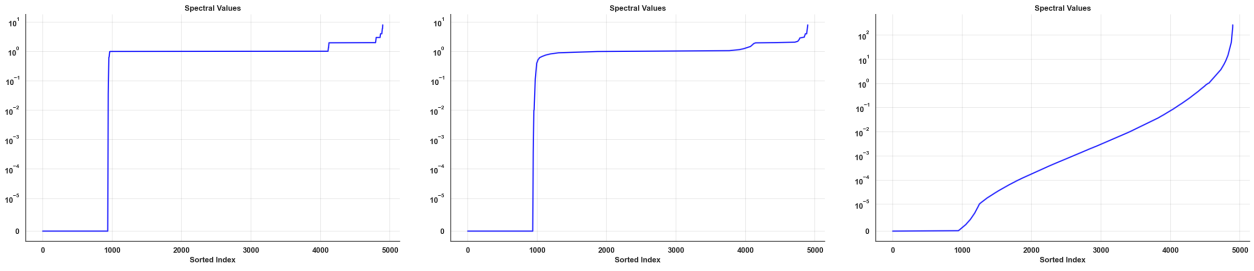
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 8. Spectral values for stock market data.



(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

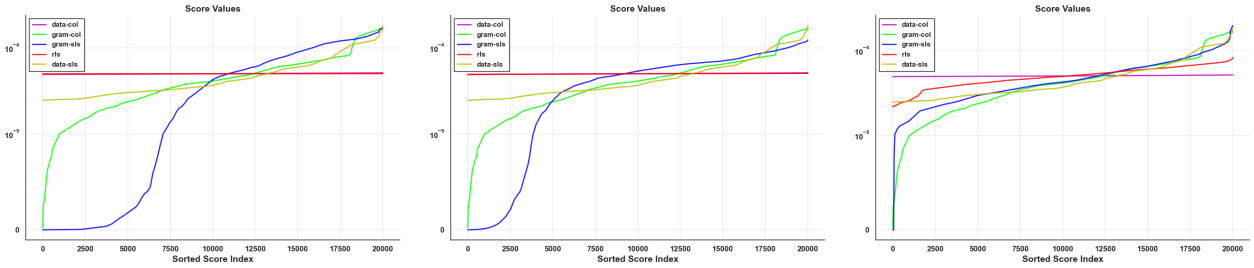
FIGURE 9. Spectral values for temperature data.



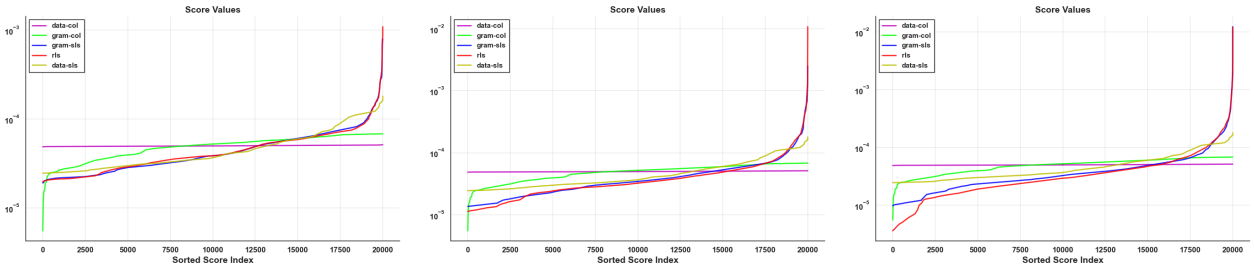
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 10. Spectral values for wine data.

A.2. **Nystrom Scores.**

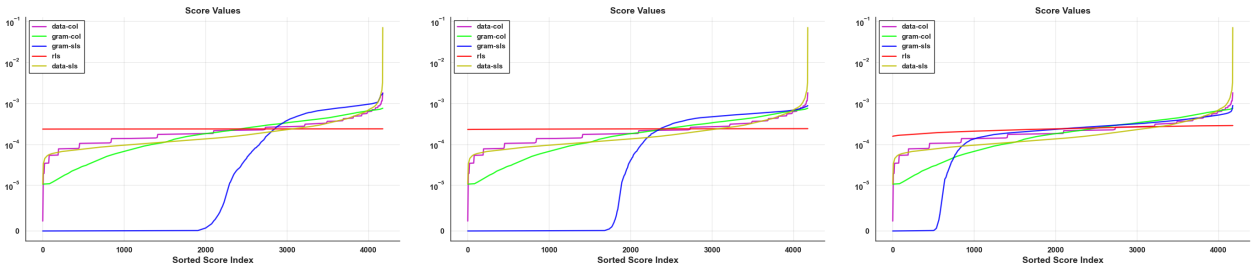


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$



(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

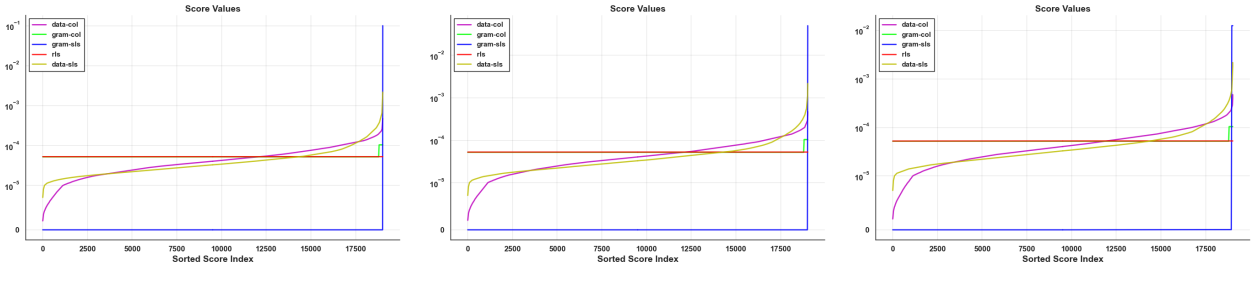
FIGURE 11. Nystrom scores for the 3D-spatial network data set.



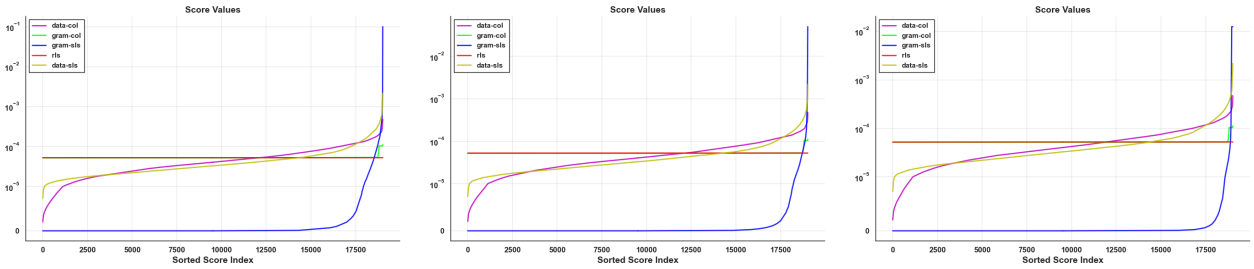
(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 12. Nystrom scores for the Abalone data set.



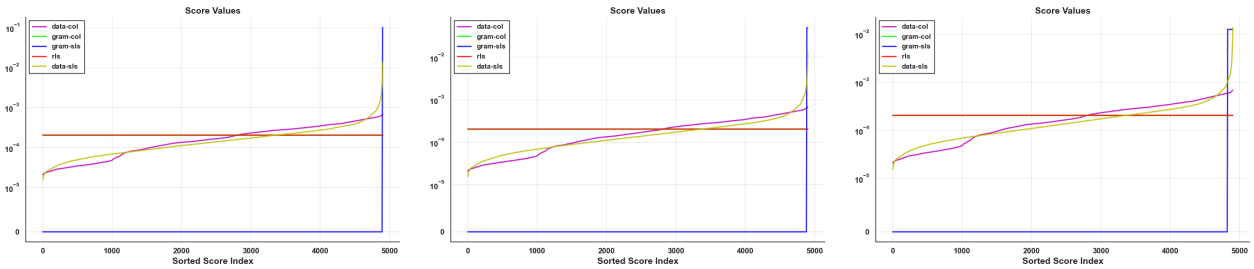
(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$



(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

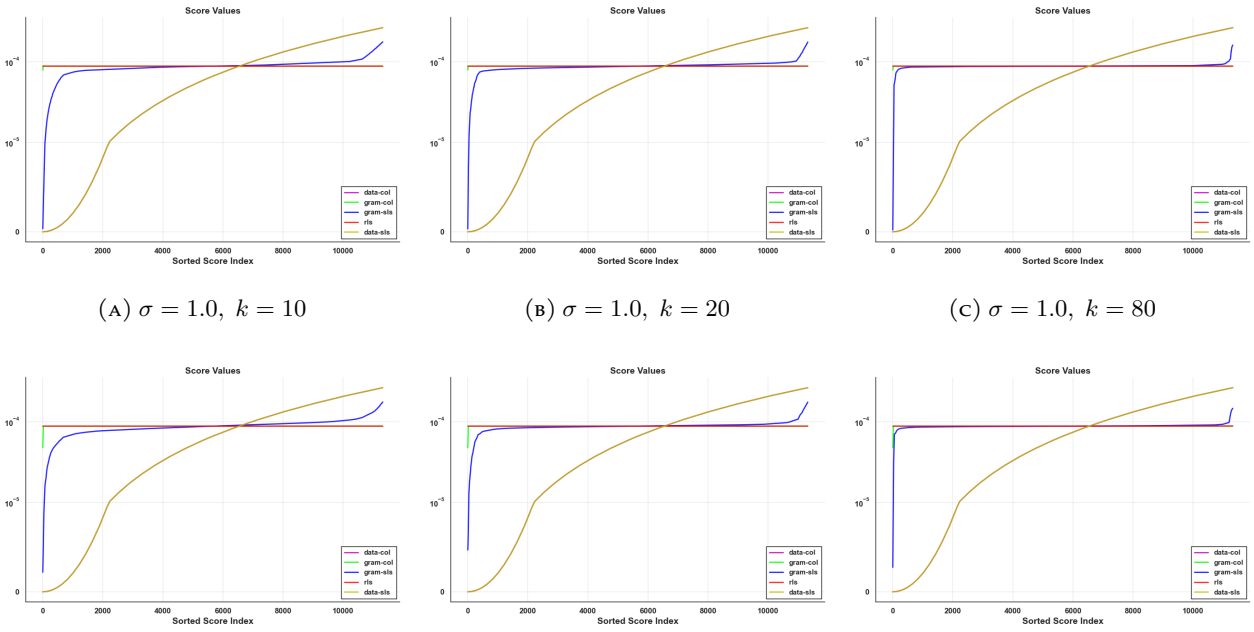
FIGURE 13. Nystrom scores for the Magic data set.

(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$



(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

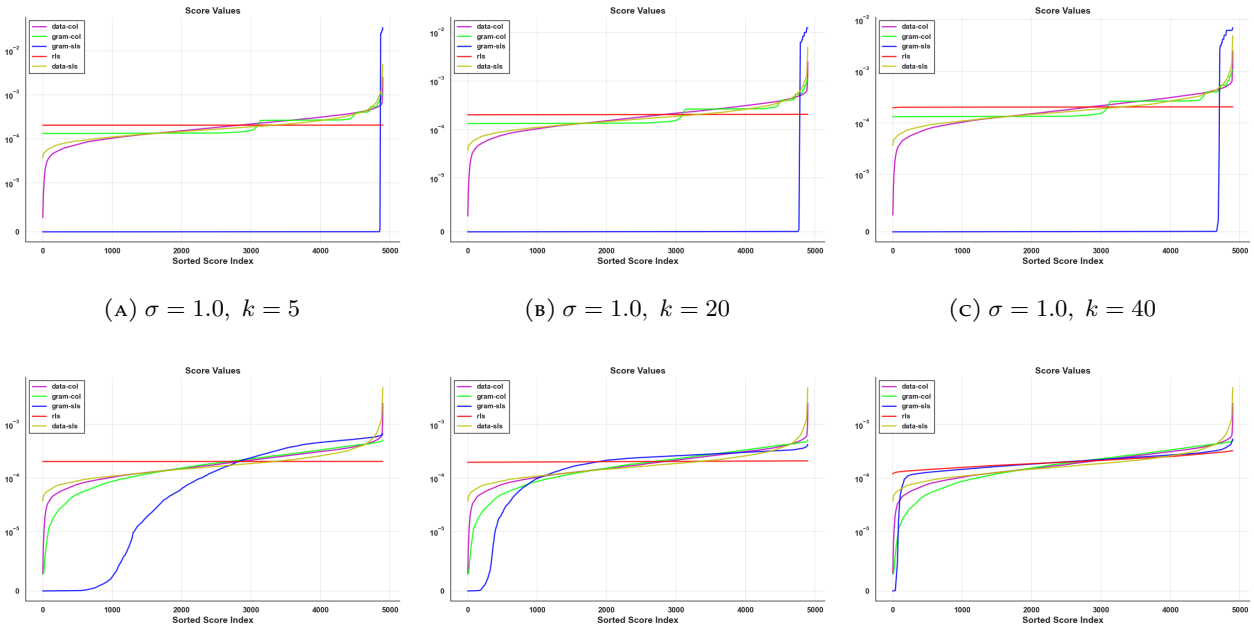
FIGURE 14. Nystrom scores for the stock market data set.



(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 15. Nystrom scores for the temperature data set.

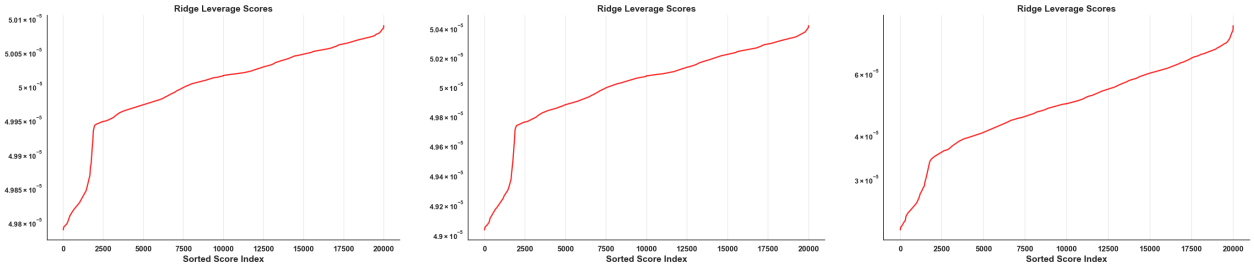


(A) $\sigma = 1.0,\ k = 5$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 40$

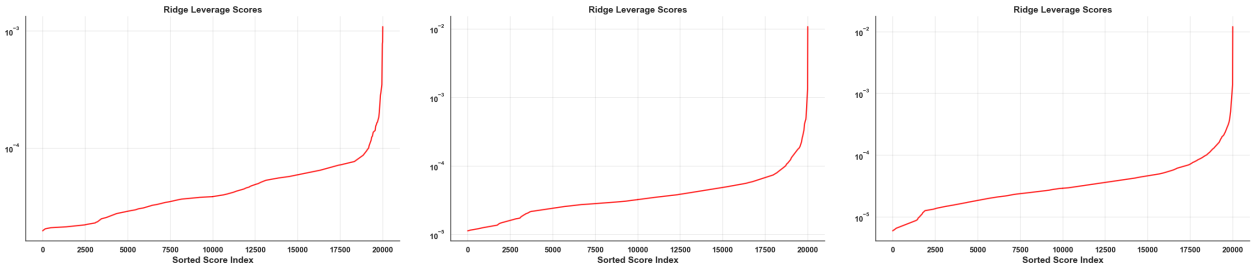
(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 16. Nystrom scores for the wine data set.

A.3. **Ridge Leverage Scores.**

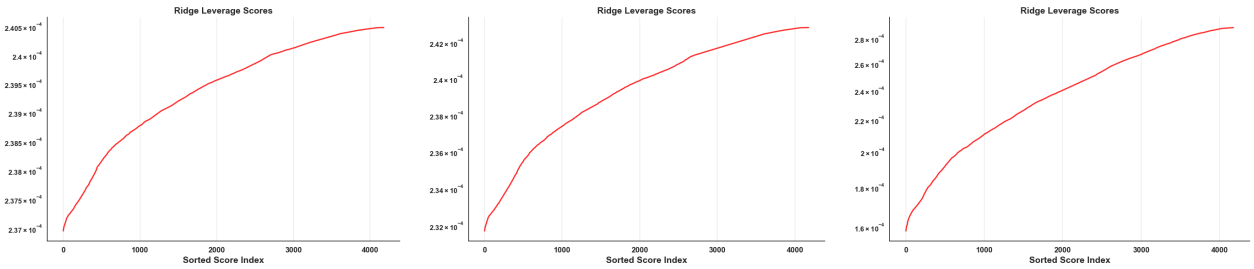


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

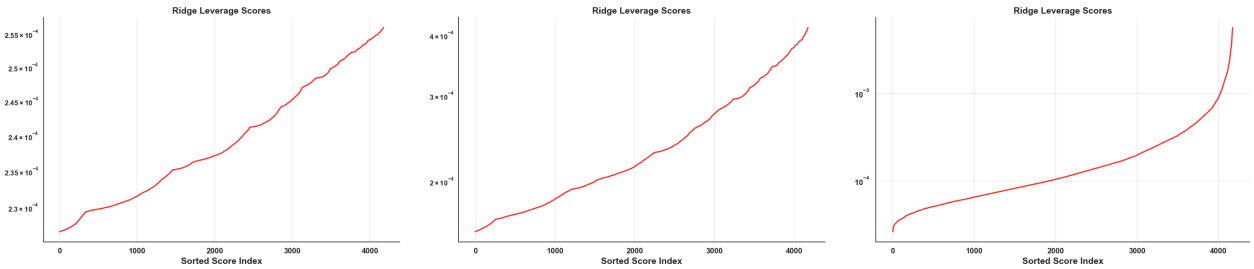


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

FIGURE 17. Ridge Leverage scores for the 3D-spatial network data set.

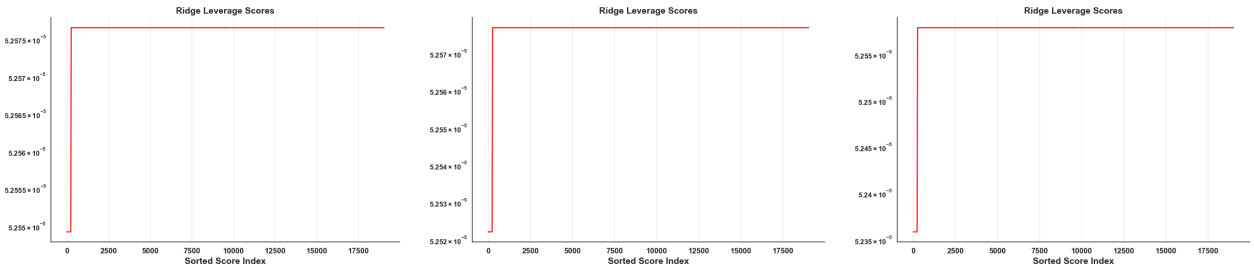


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

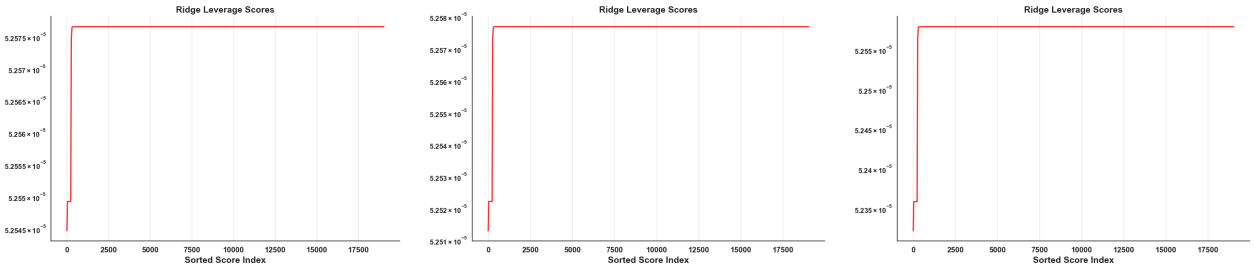


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 18. Ridge Leverage scores for the Abalone data set.

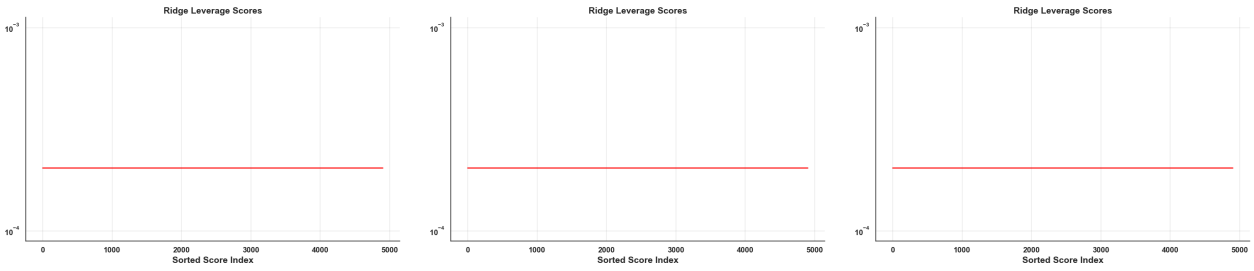


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

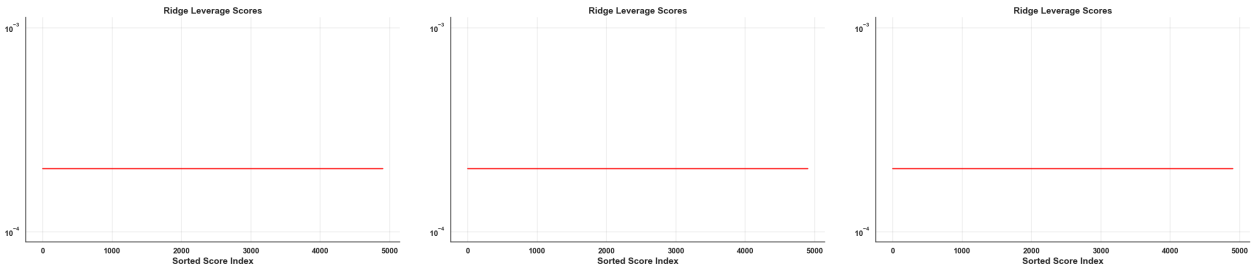


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 19. Ridge Leverage scores for the Magic data set.

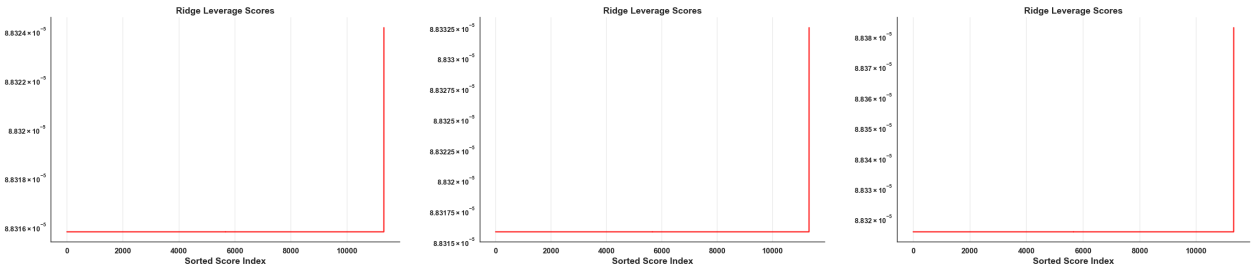


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

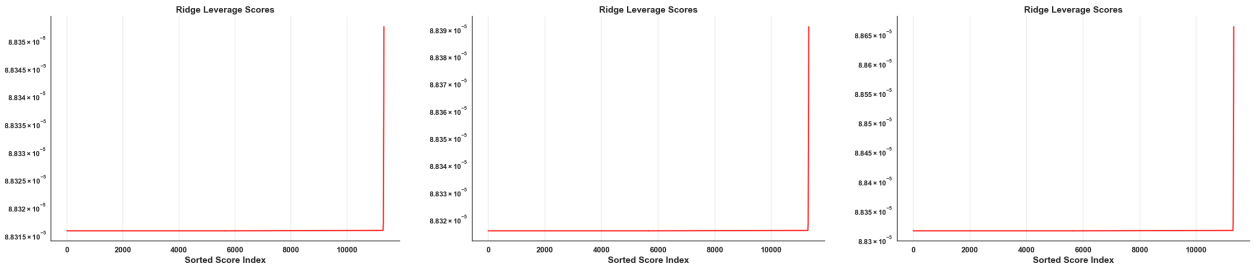


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 20. Ridge Leverage scores for the stock market data set.

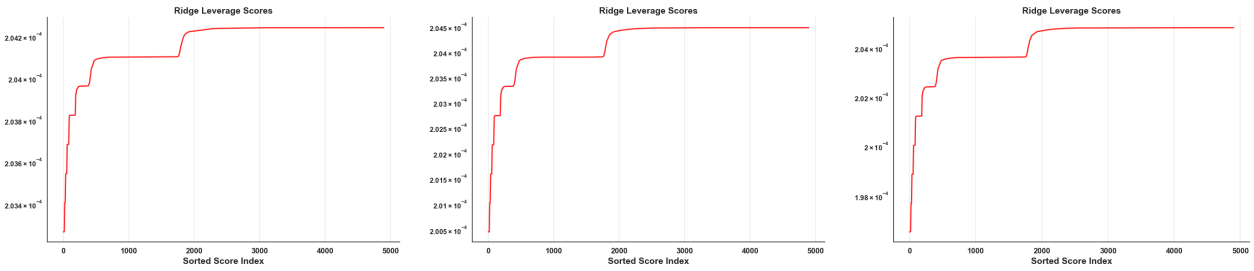


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

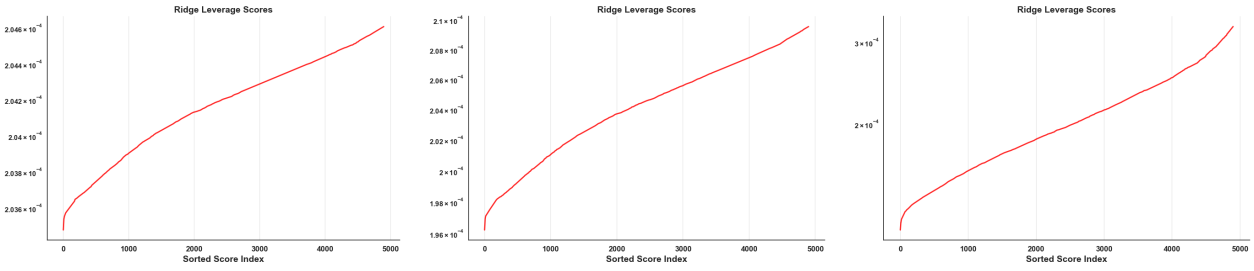


(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 21. Ridge Leverage scores for the temperature data set.



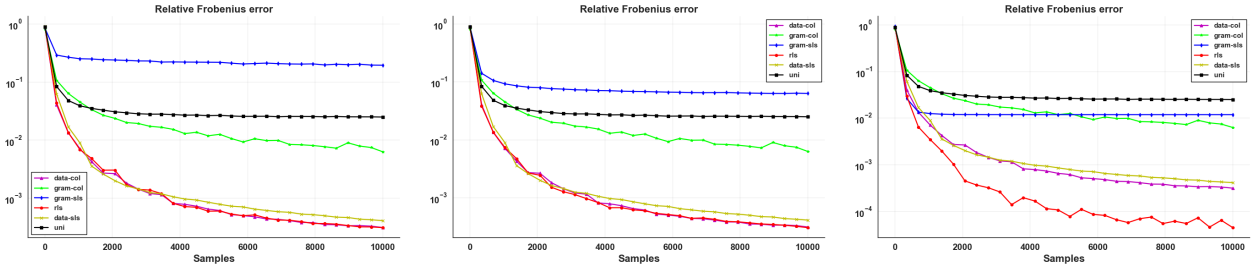
(A) $\sigma = 1.0,\ k = 5$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 40$



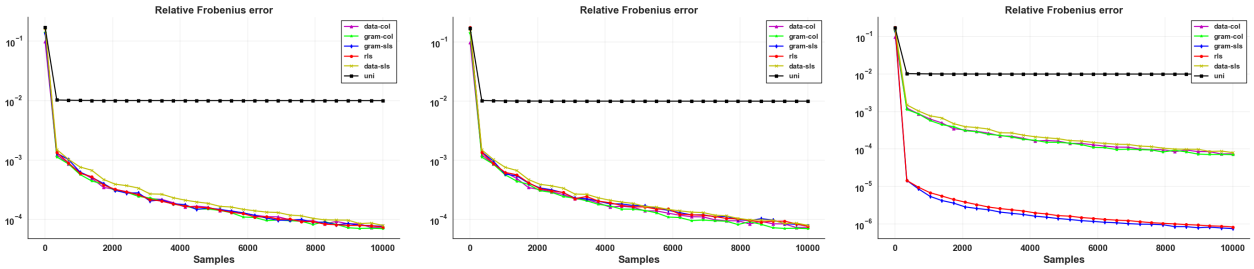
(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 22. Ridge Leverage scores for the wine data set.

A.4. **Nystrom Errors.**

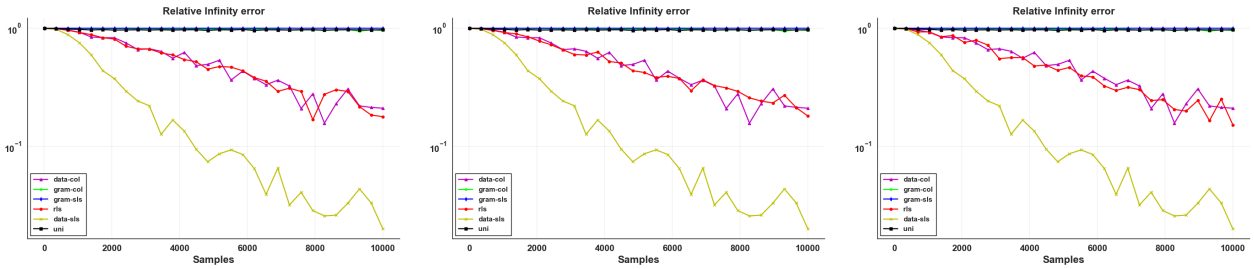


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

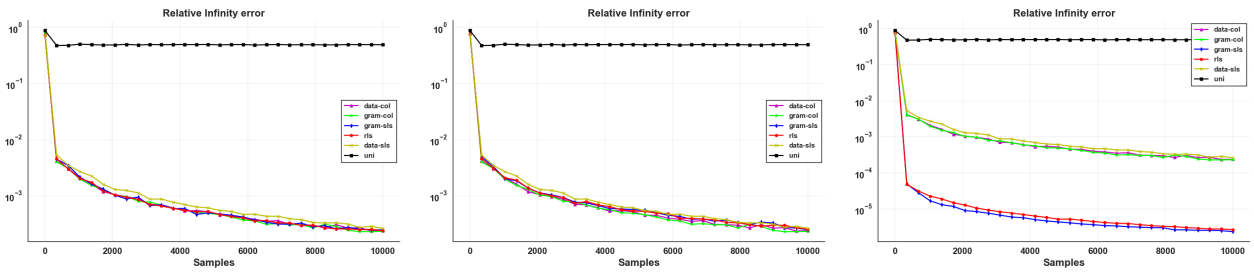


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

FIGURE 23. Nystrom Frobenius errors for the 3DSN data set.

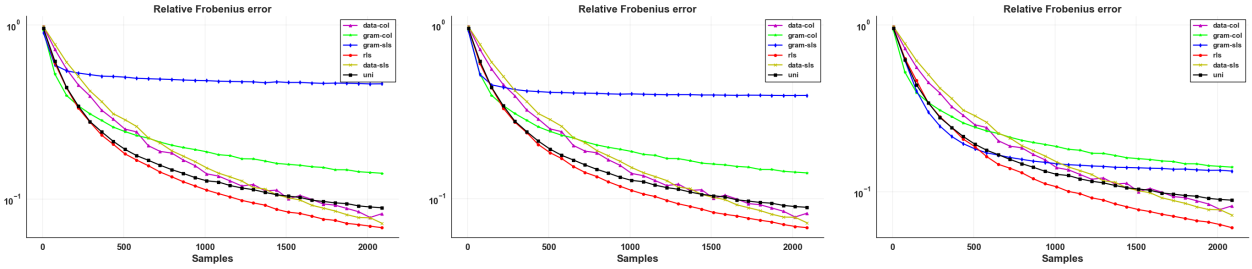


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 40$

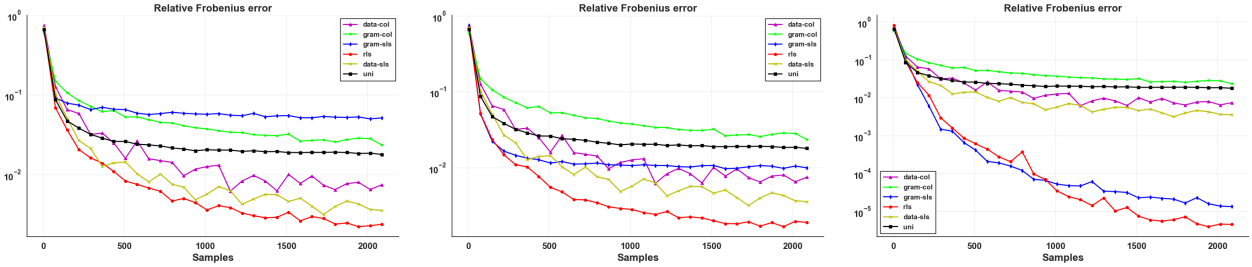


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

FIGURE 24. Nystrom infinity errors for the 3DSN data set.

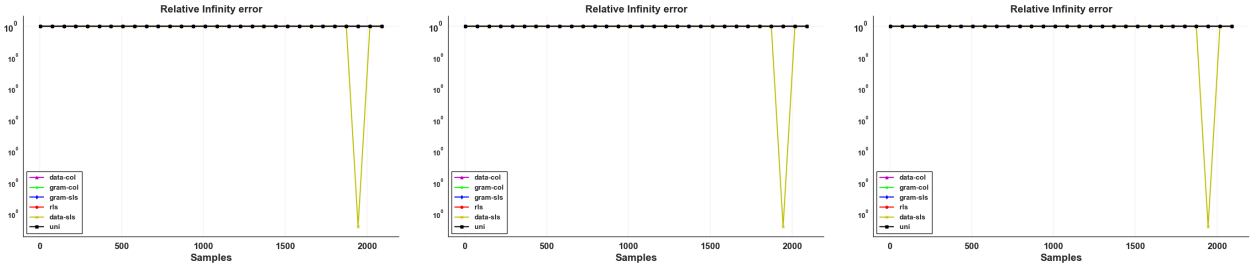


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$



(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

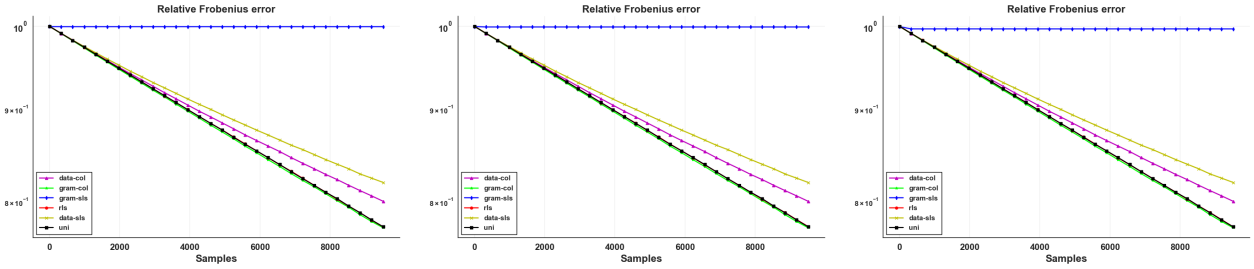
FIGURE 25. Nystrom Frobenius errors for the Abalone data set.



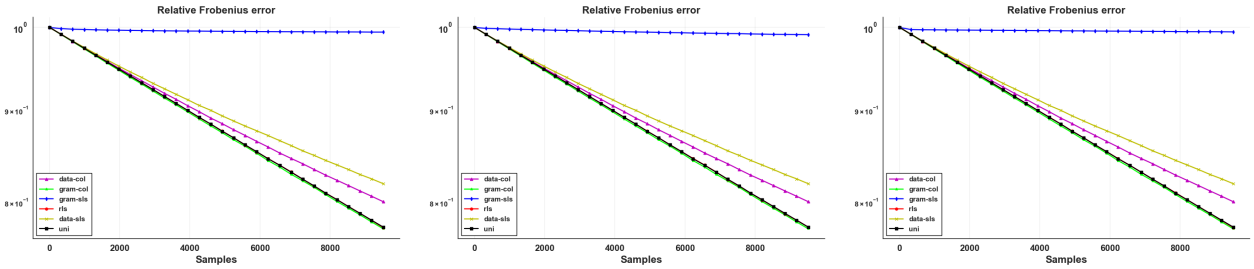
(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 26. Nystrom infinity errors for the Abalone data set.

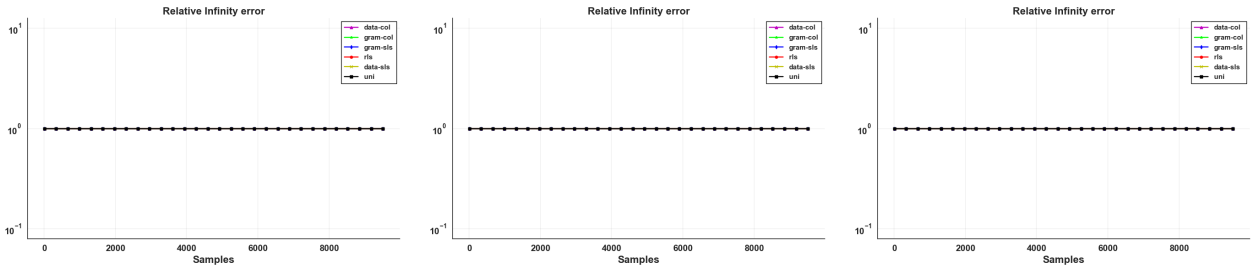


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

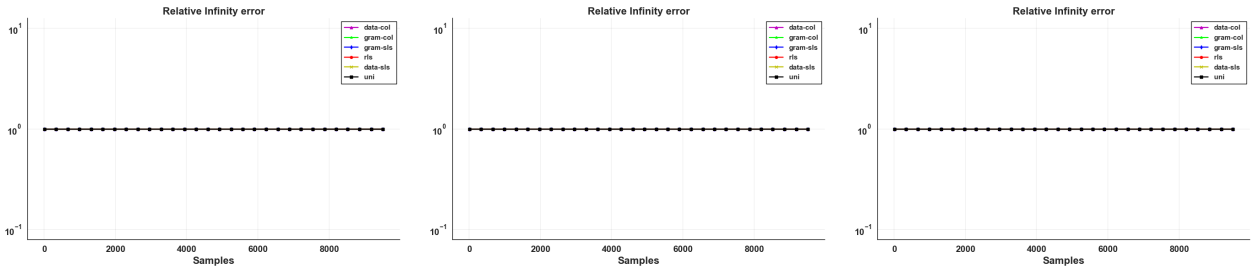


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 27. Nystrom Frobenius errors for the Magic data set.

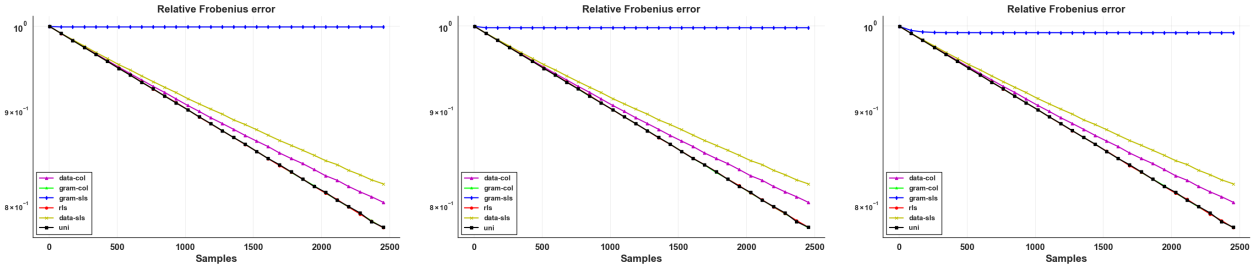


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

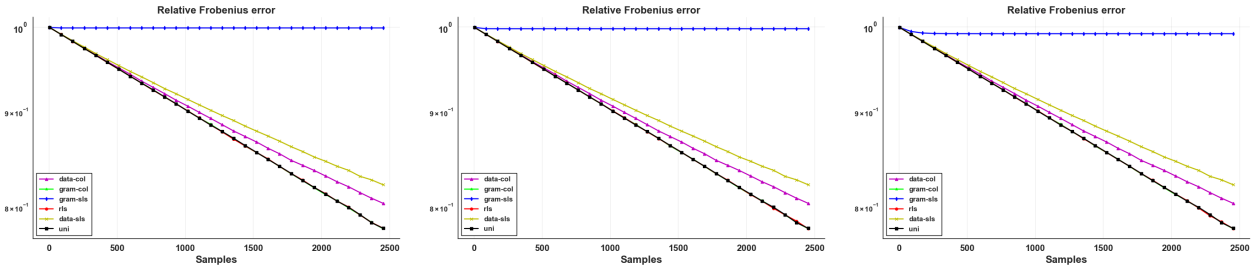


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 28. Nystrom infinity errors for the Magic data set.

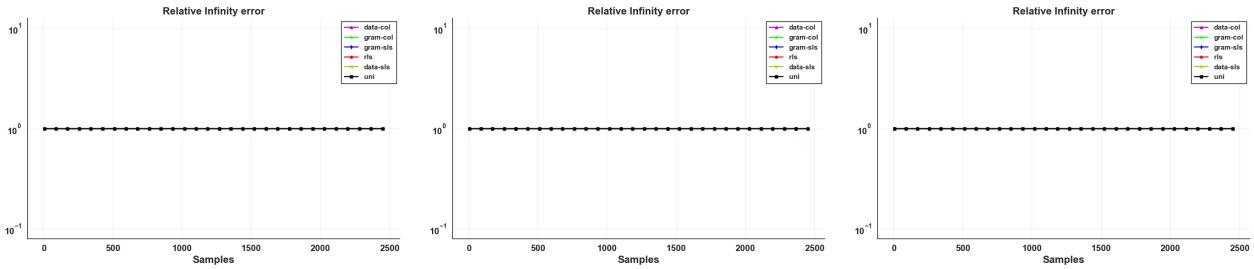


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

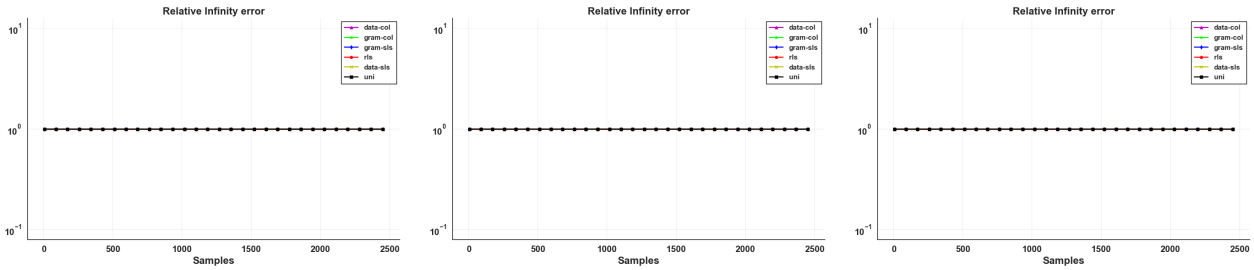


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 29. Nystrom Frobenius errors for the Stocks data set.

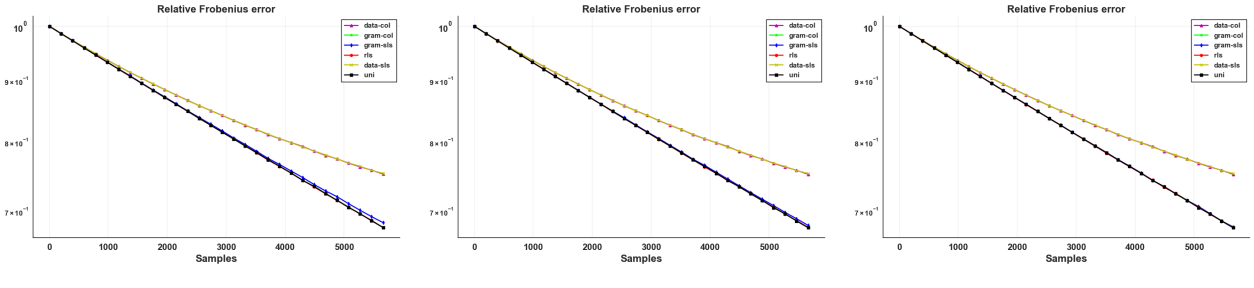


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

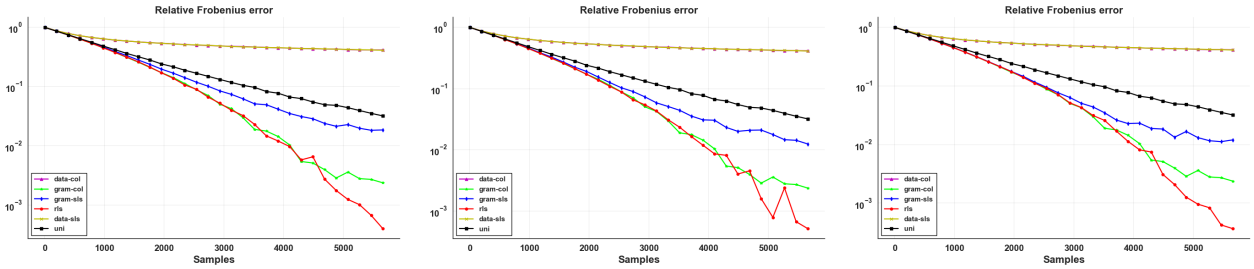


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 30. Nystrom infinity errors for the Stocks data set.

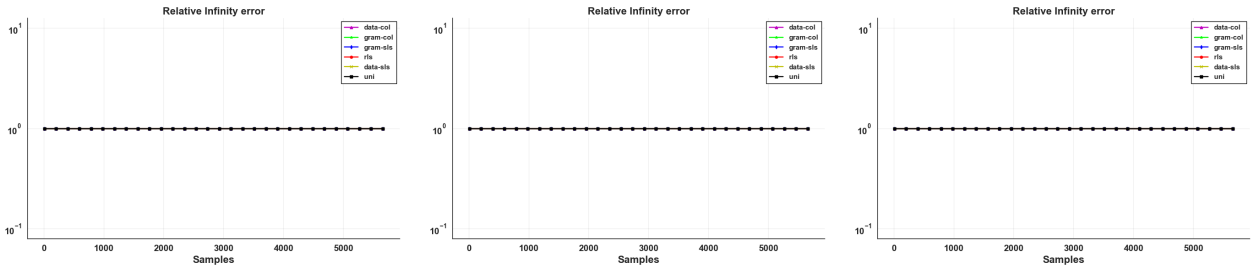


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

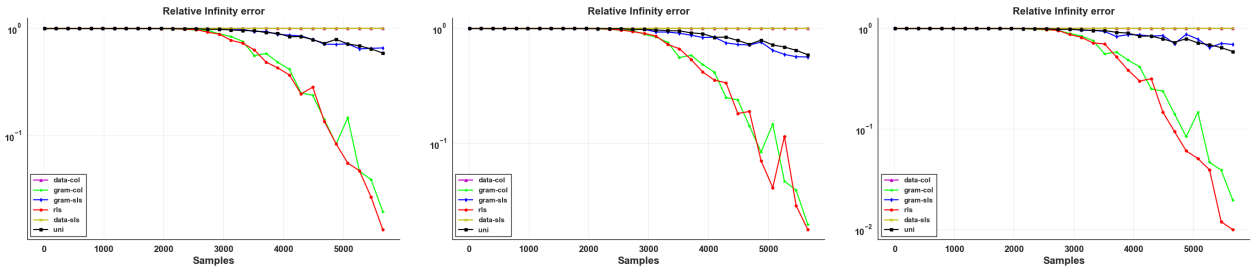


(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 31. Nystrom Frobenius errors for the Temp data set.

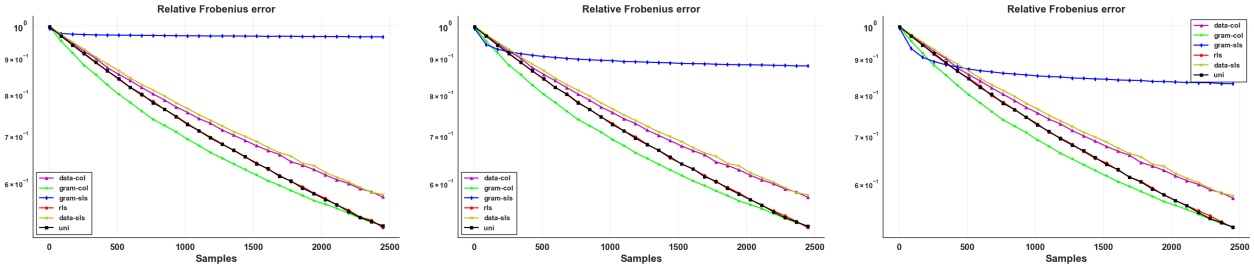


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

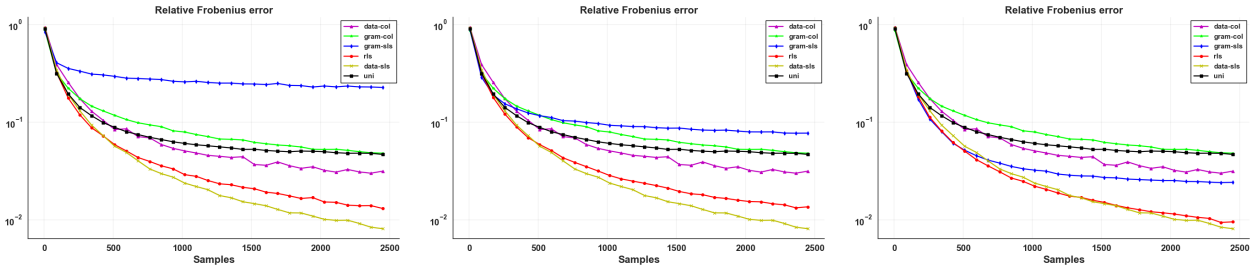


(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 32. Nystrom infinity errors for the Temp data set.

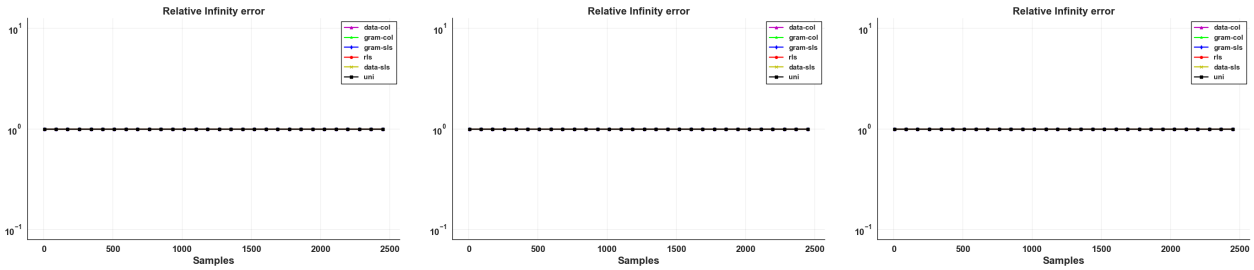


(A) $\sigma = 2.1,\ k = 5$ (B) $\sigma = 2.1,\ k = 20$ (C) $\sigma = 2.1,\ k = 40$

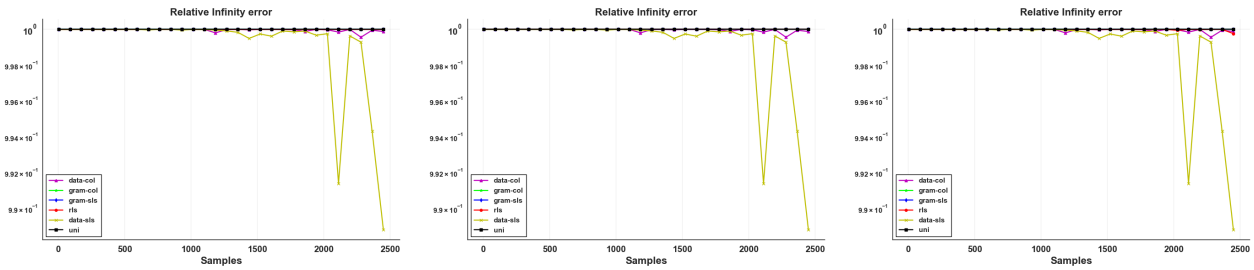


(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 33. Nystrom Frobenius errors for the wine data set.



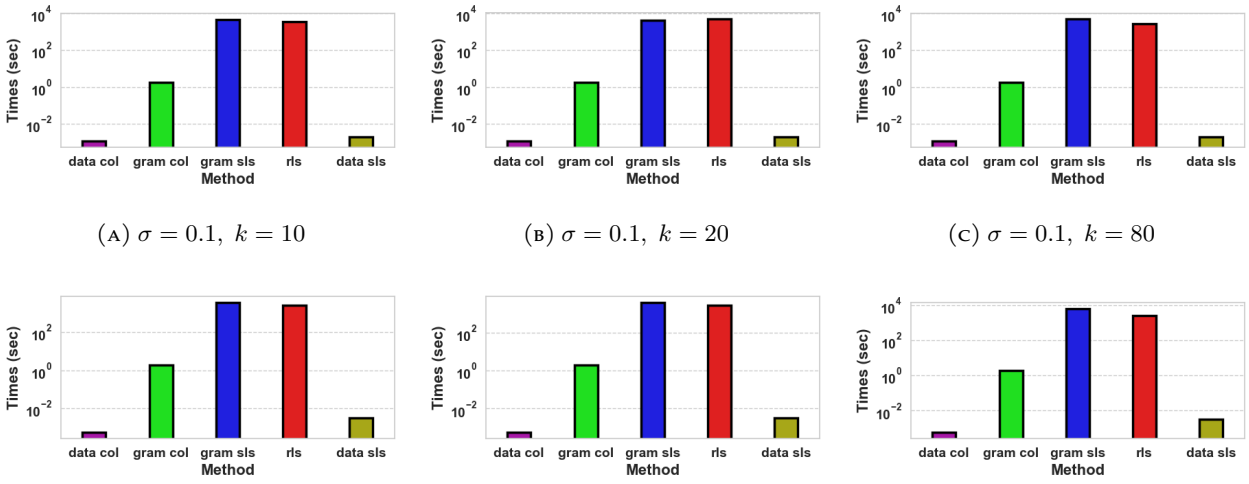
(A) $\sigma = 2.1,\ k = 5$ (B) $\sigma = 2.1,\ k = 20$ (C) $\sigma = 2.1,\ k = 40$



(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 34. Nystrom infinity errors for the wine data set.

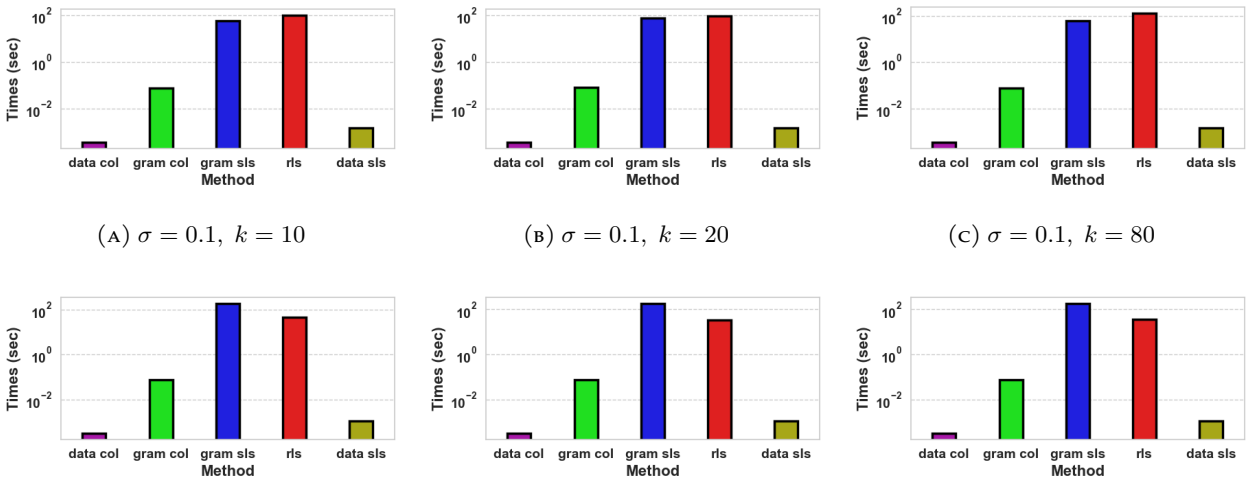
A.5. **Nystrom Sampling Distribution Construction Times.**



(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

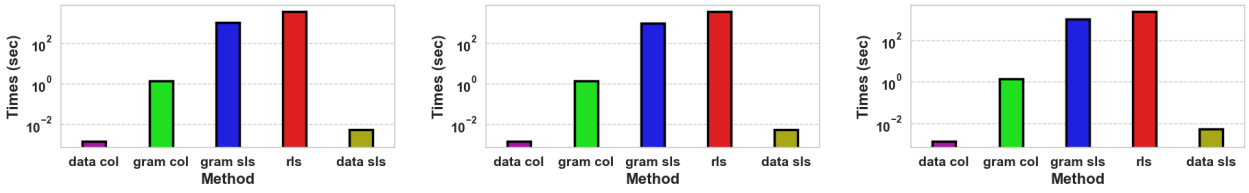
FIGURE 35. Nystrom sampling distribution construction times for the 3D-spatial network data set.



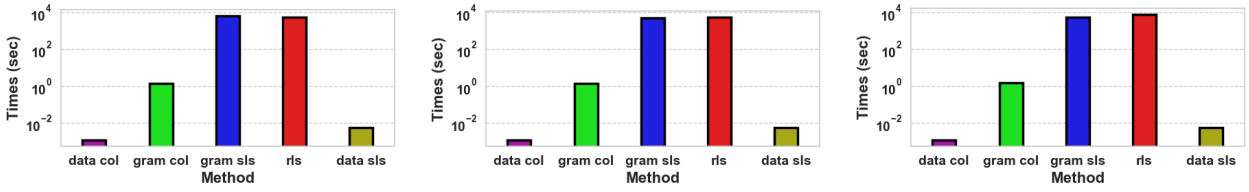
(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 36. Nystrom sampling distribution construction times for the Abalone data set.

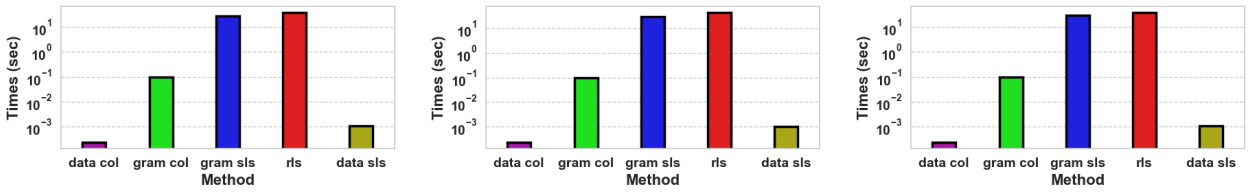


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

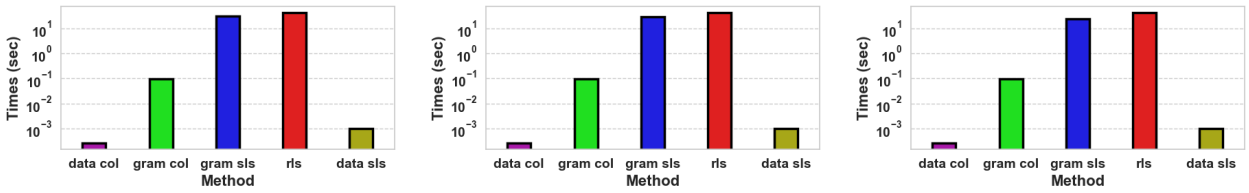


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 37. Nystrom sampling distribution construction times for the Magic data set.

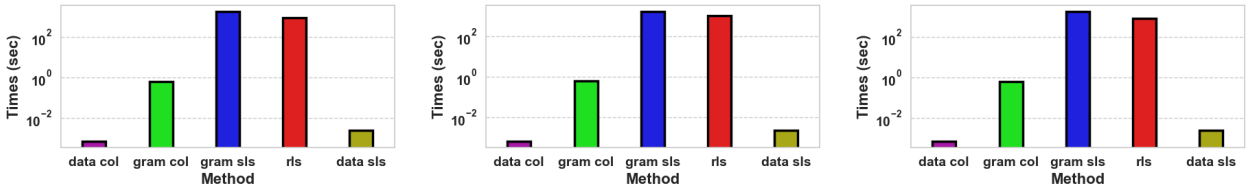


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

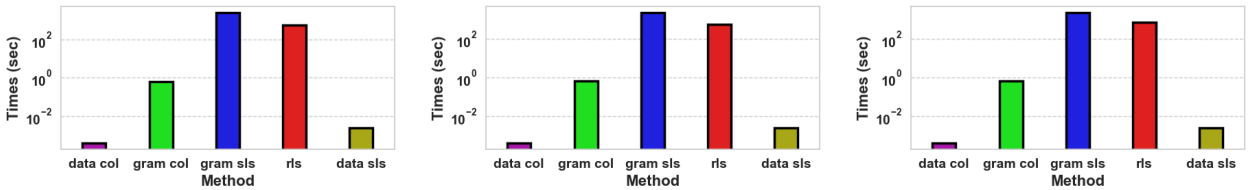


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 38. Nystrom sampling distribution construction times for the stock market data set.

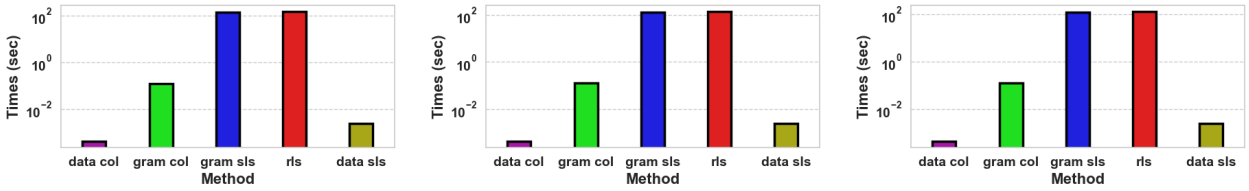


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

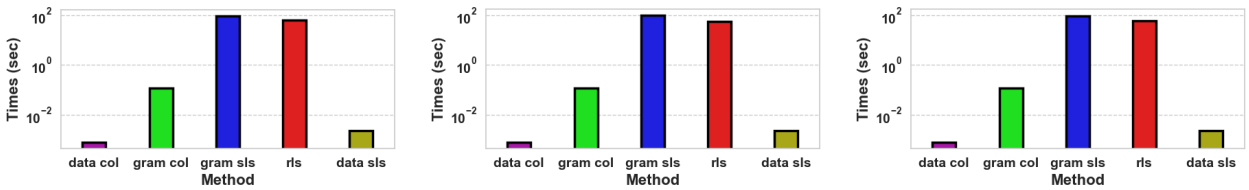


(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 39. Nystrom sampling distribution construction times for the temperature data set.



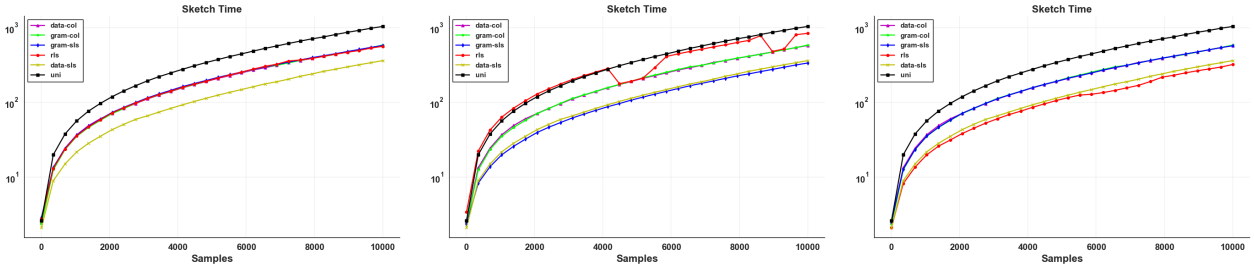
(A) $\sigma = 1.0,\ k = 5$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 40$



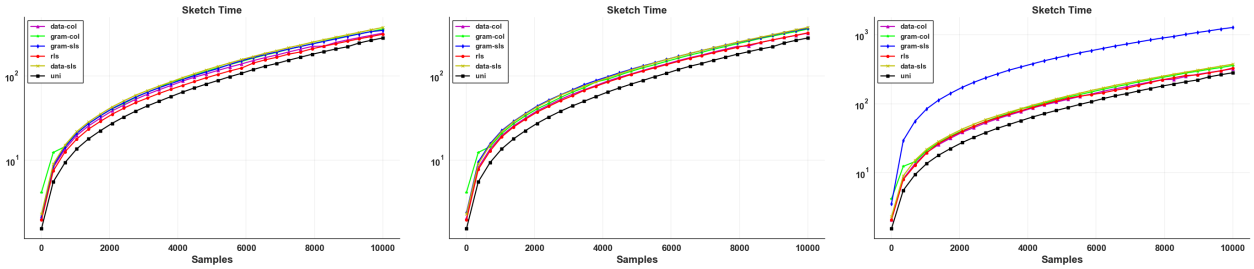
(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 40. Nystrom sampling distribution construction times for the wine data set.

A.6. **Nystrom Sketch Times.**

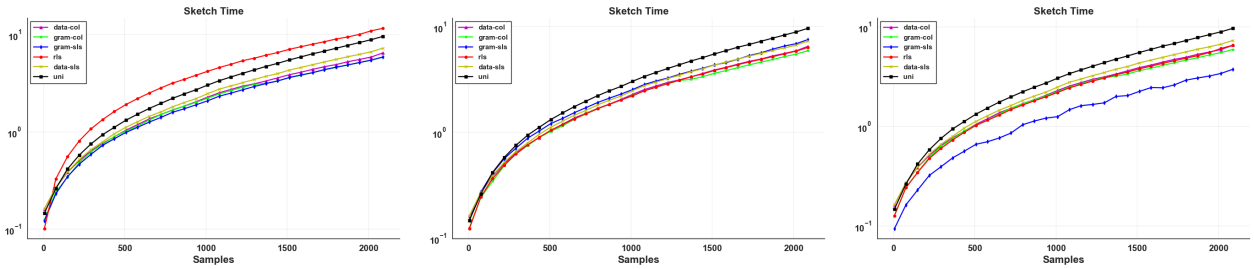


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

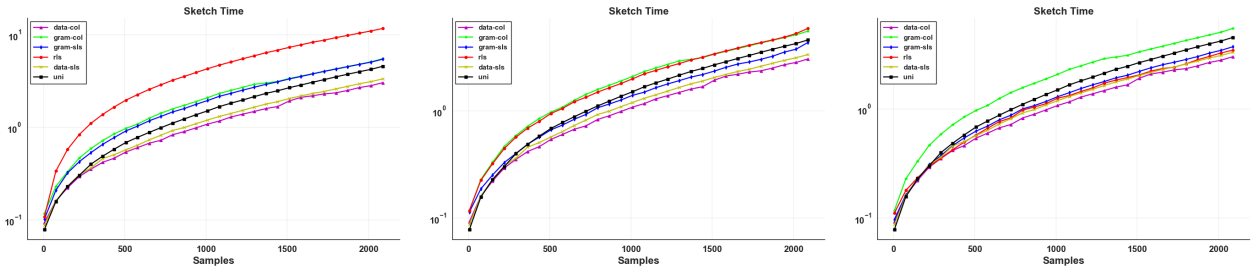


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

FIGURE 41. Nystrom sketch times for the 3D-spatial network data set.

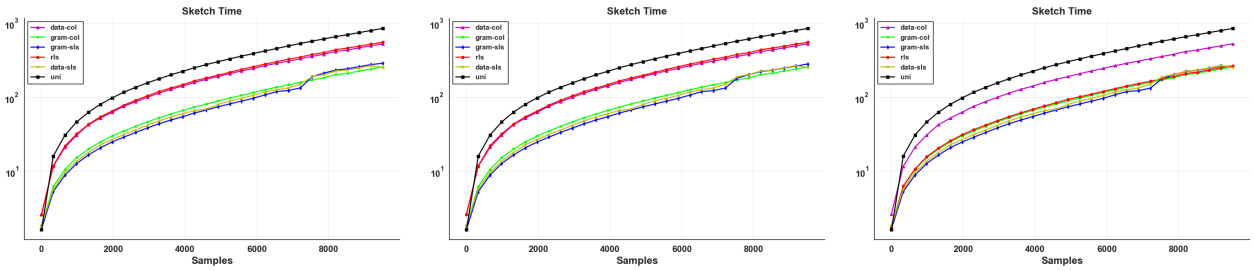


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

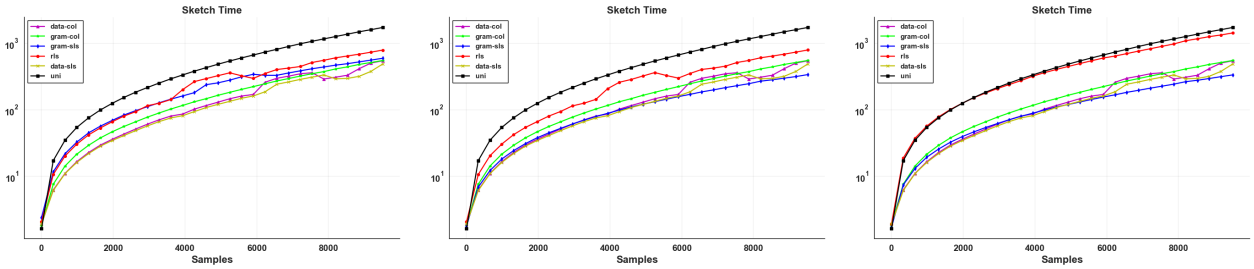


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 42. Nystrom sketch times for the Abalone data set.

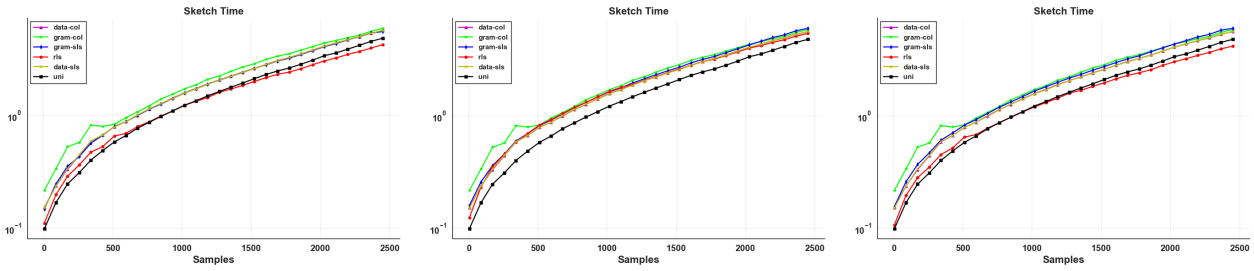


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

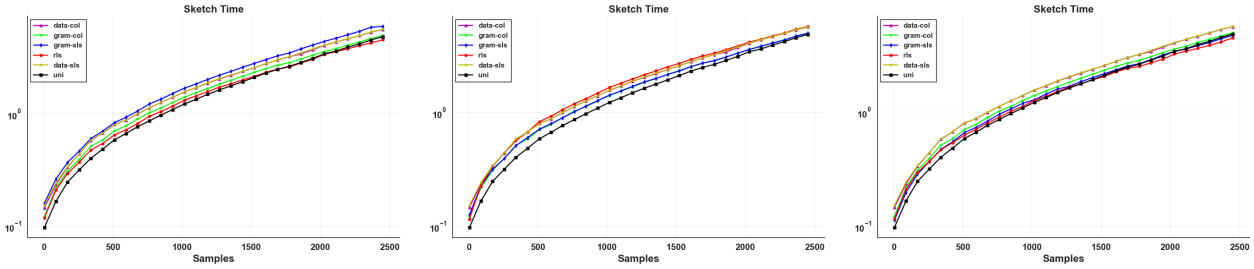


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 43. Nystrom sketch times for the Magic data set.

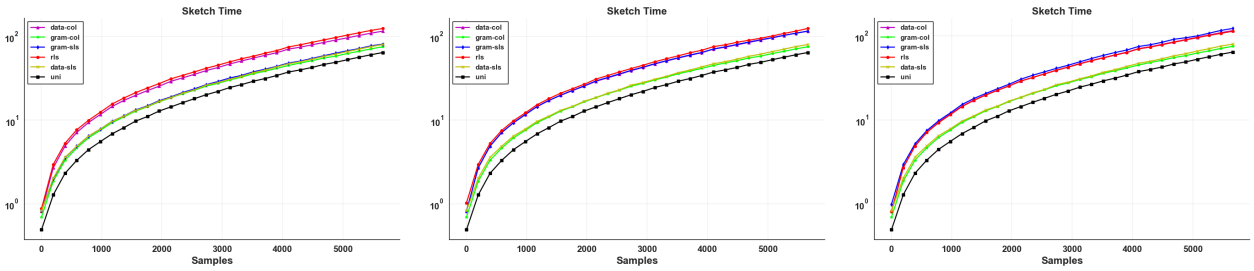


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

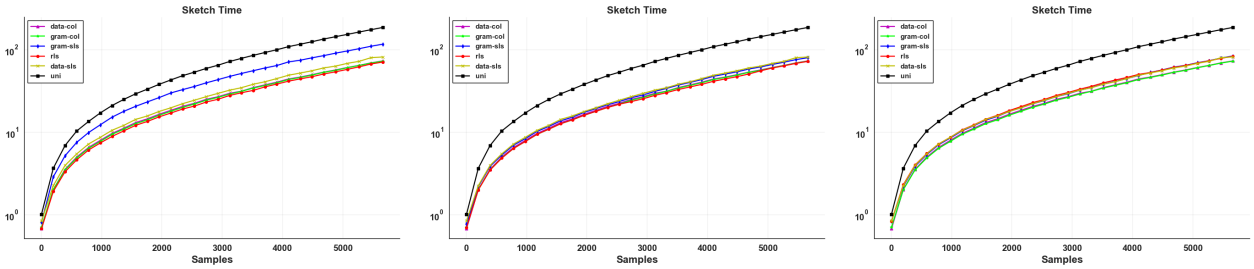


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 44. Nystrom sketch times for the stock market data set.

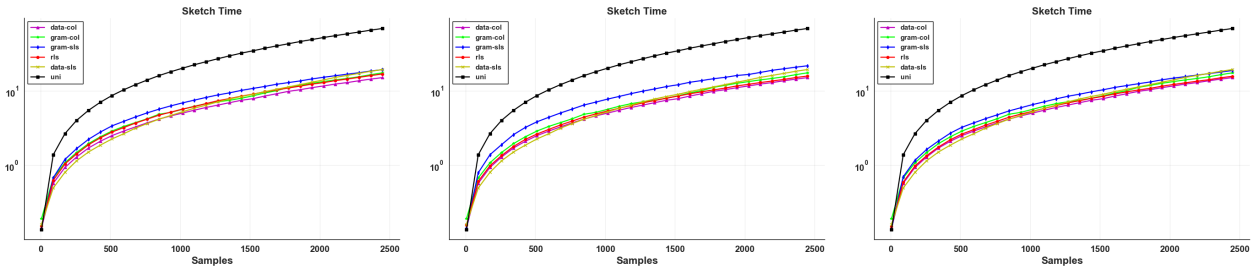


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

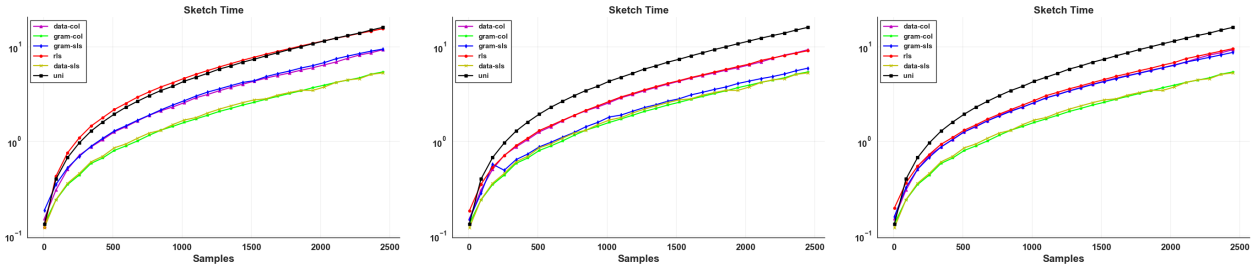


(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 45. Nystrom sketch times for the temperature data set.



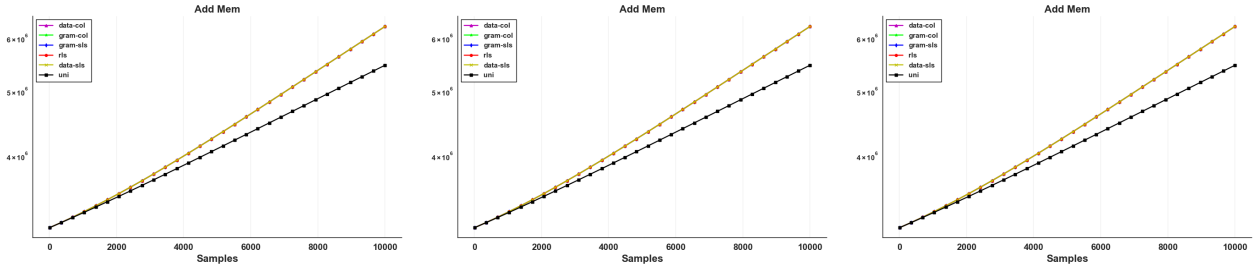
(A) $\sigma = 1.0,\ k = 5$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 40$



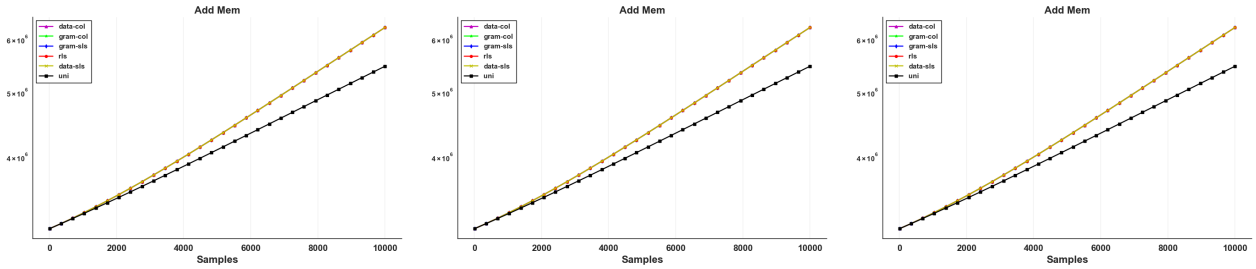
(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 46. Nystrom sketch times for the wine data set.

A.7. **Nystrom Memory Usage.**

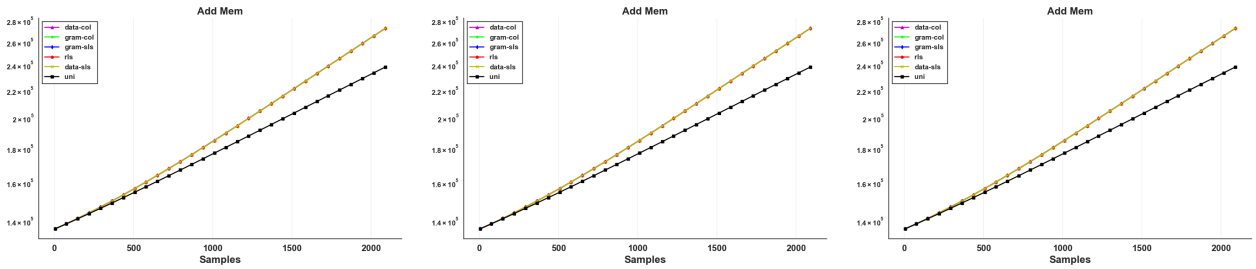


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

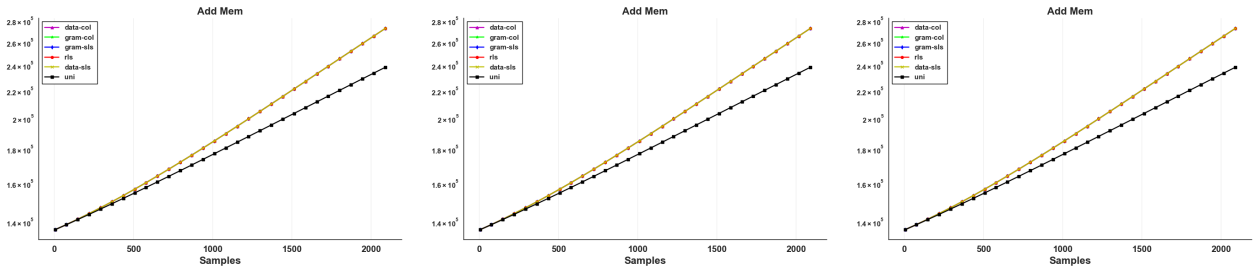


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

FIGURE 47. Nystrom sketch times for the 3D-spatial network data set.

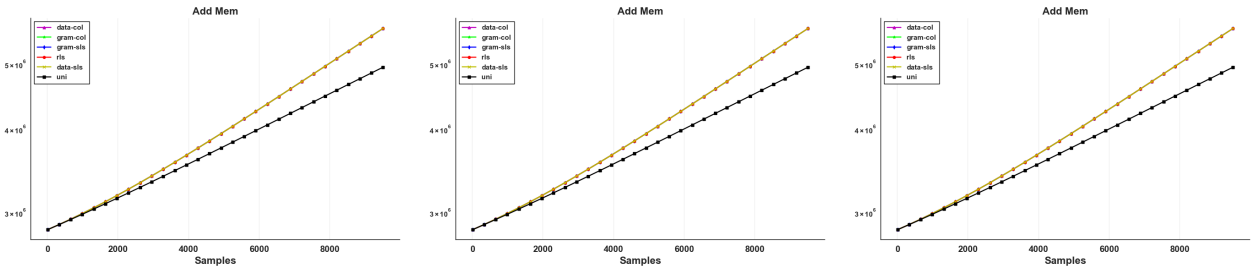


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

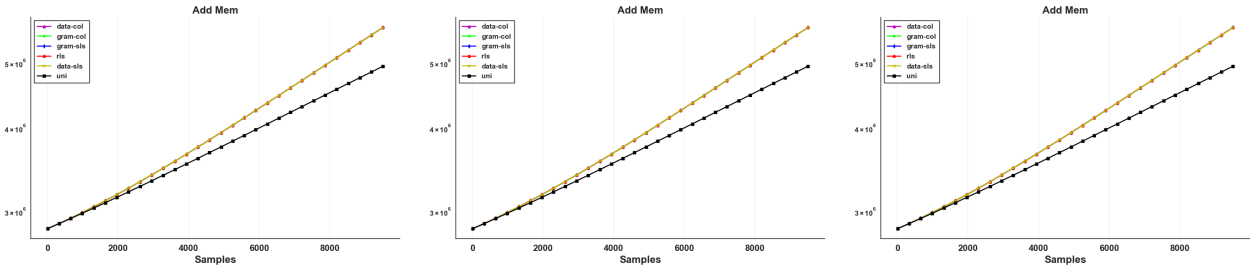


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 48. Nystrom sketch times for the Abalone data set.

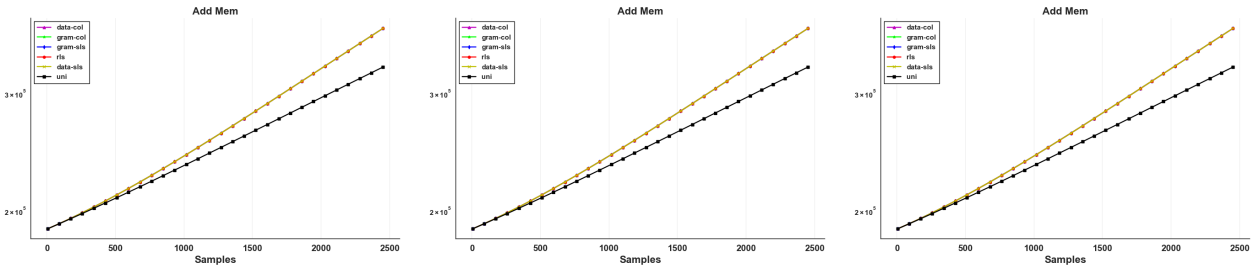


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

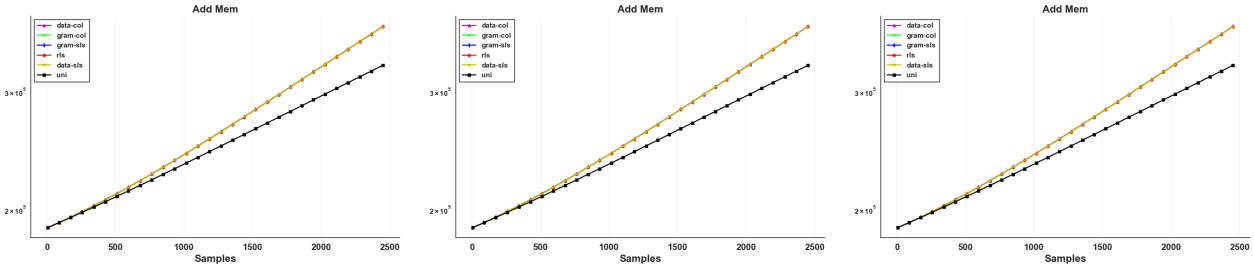


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 49. Nystrom sketch times for the Magic data set.

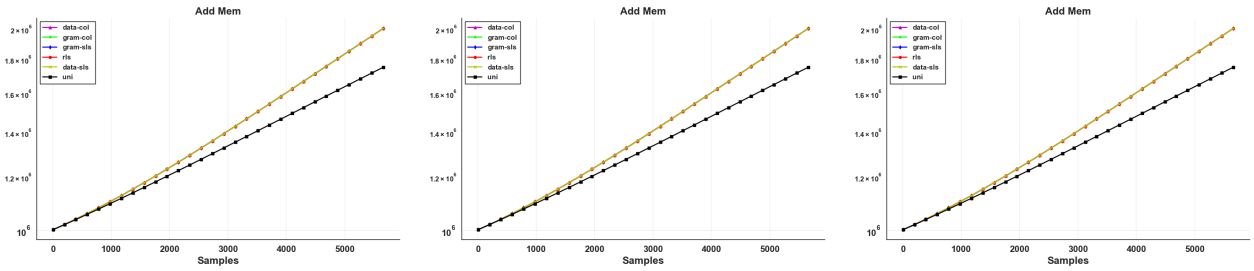


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

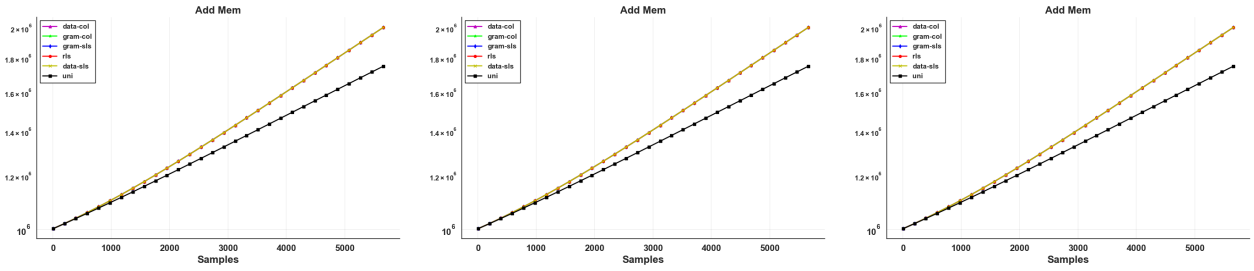


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 50. Nystrom sketch times for the stock market data set.

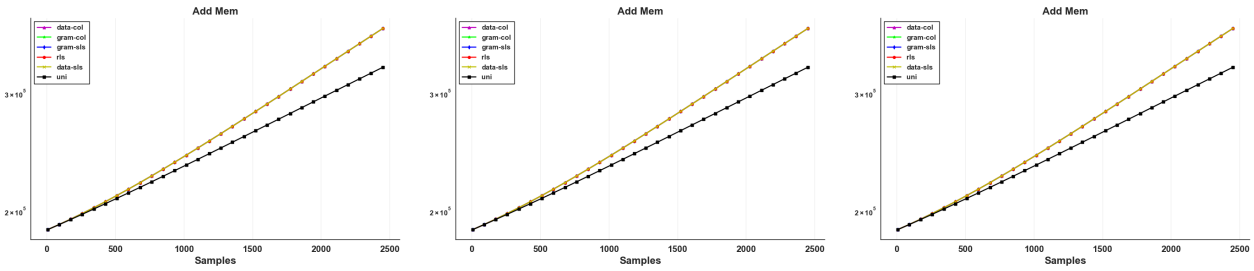


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

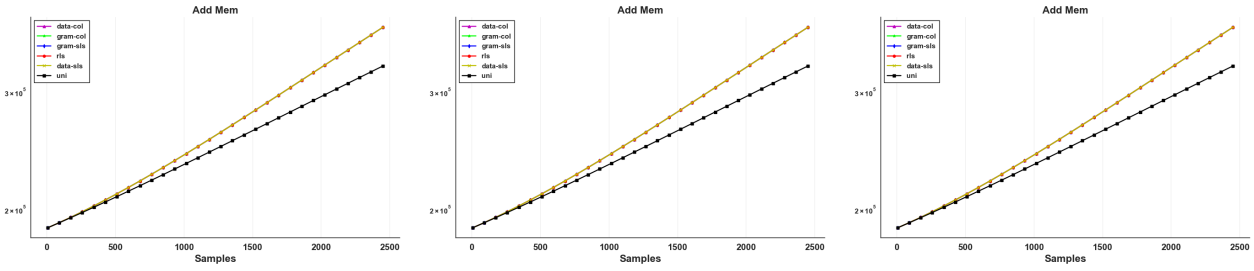


(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 51. Nystrom sketch times for the temperature data set.



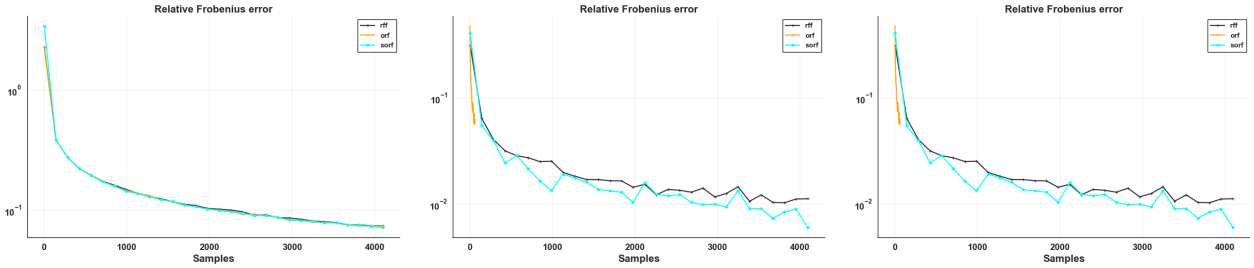
(A) $\sigma = 1.0,\ k = 5$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 40$



(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

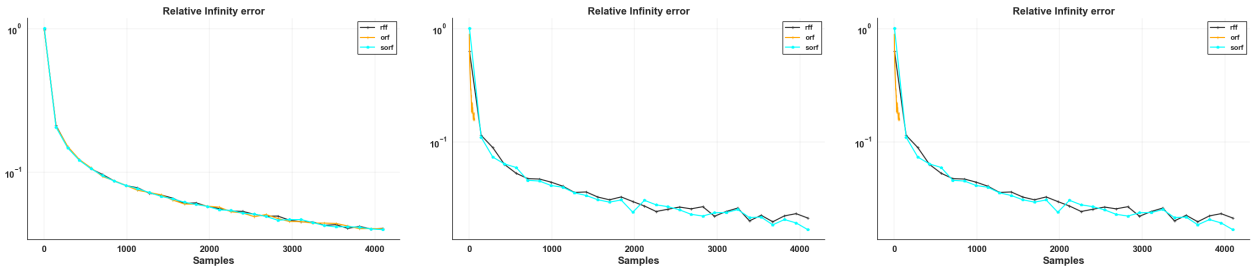
FIGURE 52. Nystrom sketch times for the wine data set.

A.8. **RFF Errors.**



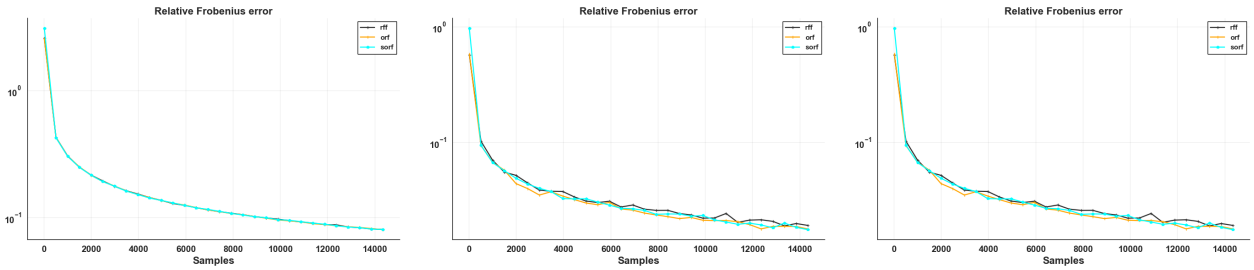
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 53. RFF Frobenius errors for the 3DSN data set.



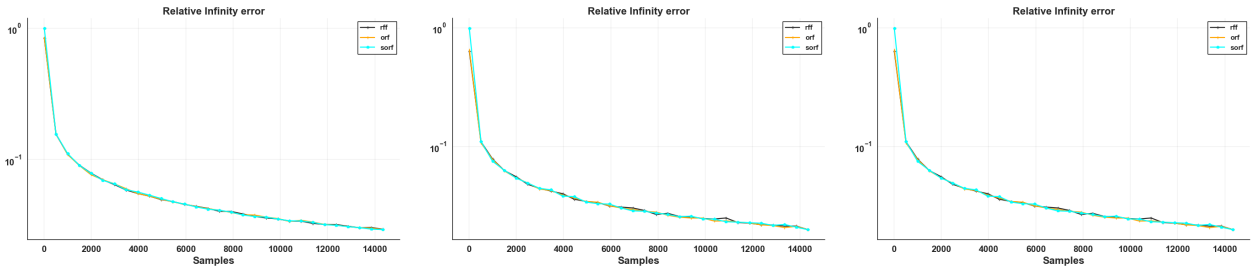
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 54. RFF absolute errors for the 3DSN data set.



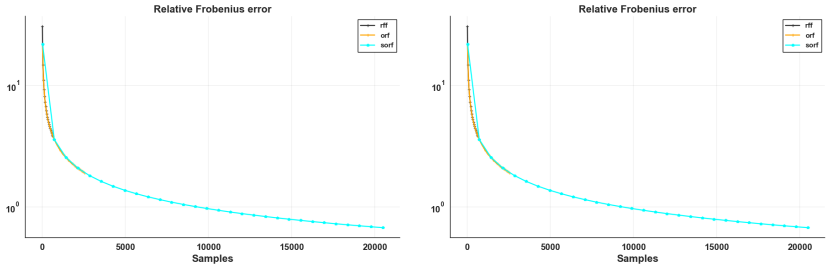
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 55. RFF Frobenius errors for the Abalone data set.



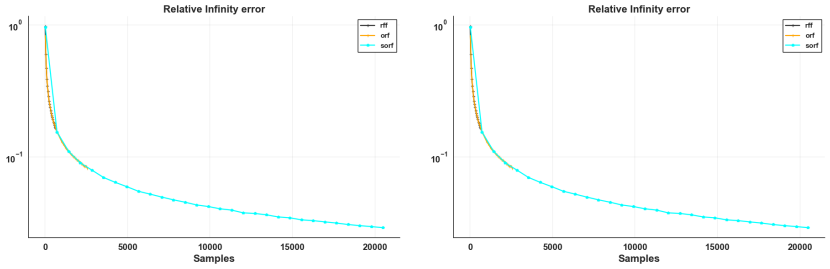
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 56. RFF infinity errors for the Abalone data set.



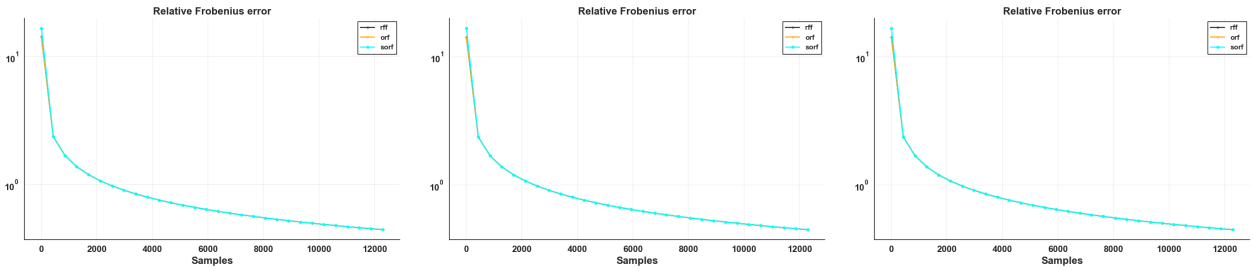
(A) $\sigma = 1.0$ (B) $\sigma = 10.0$

FIGURE 57. RFF Frobenius errors for the magic04 data set.



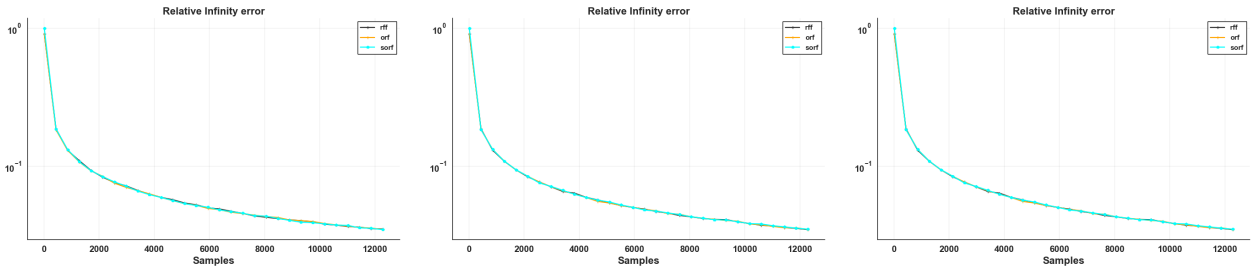
(A) $\sigma = 1.0$ (B) $\sigma = 10.0$

FIGURE 58. RFF infinity errors for the magic04 data set.



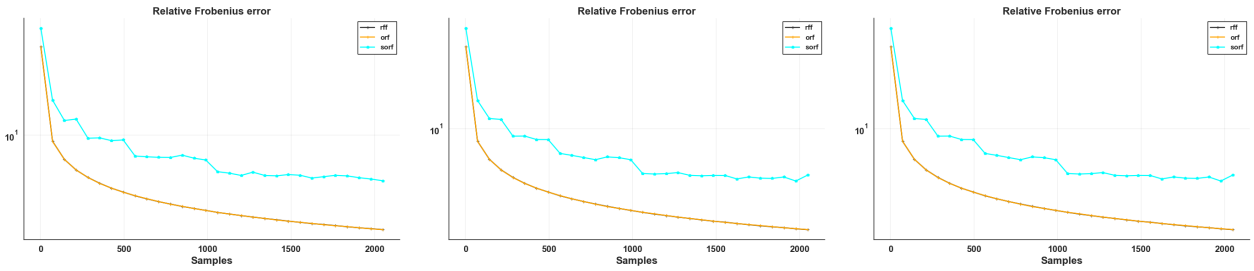
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 59. RFF Frobenius errors for the Stocks data set.



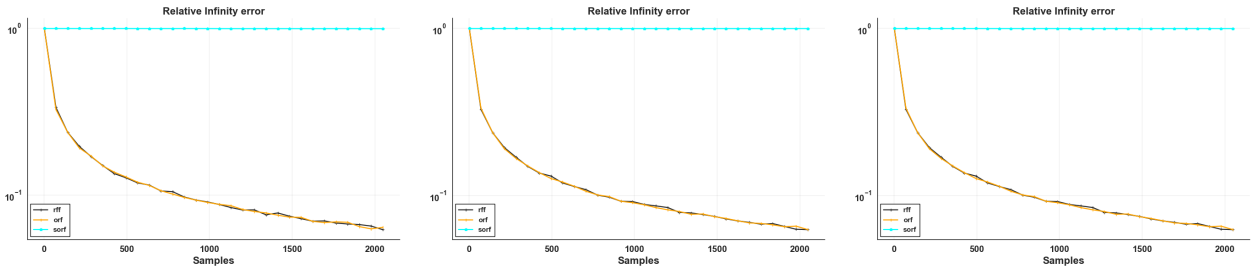
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 60. RFF infinity errors for the Stocks data set.



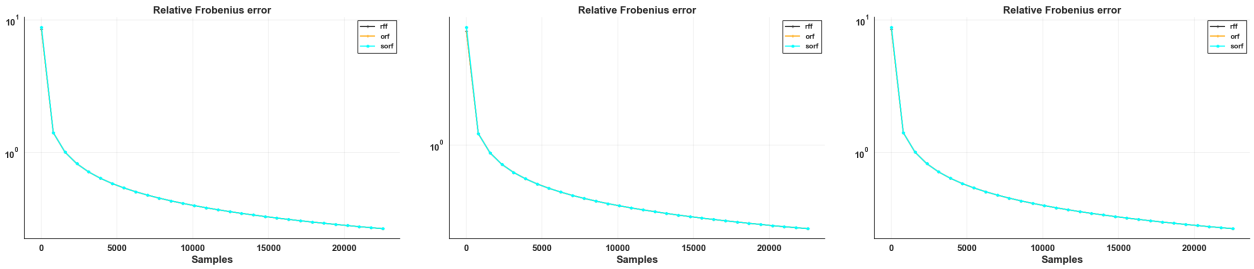
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 61. RFF Frobenius errors for the Temp data set.



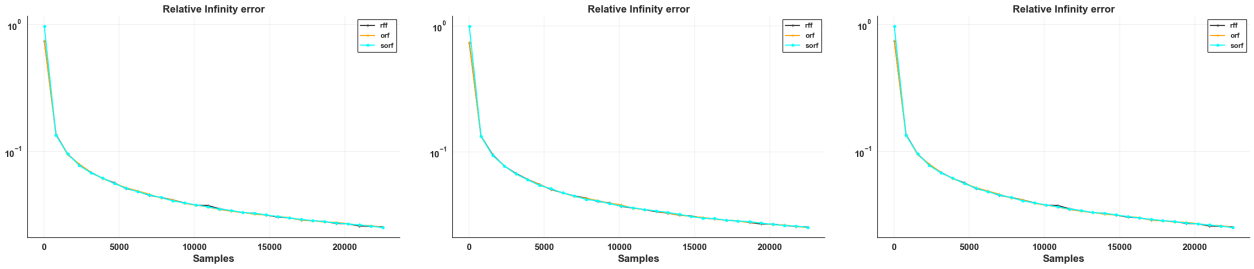
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 62. RFF infinity errors for the Temp data set.



(A) $\sigma = 1.0$ (B) $\sigma = 2.1$ (C) $\sigma = 10.0$

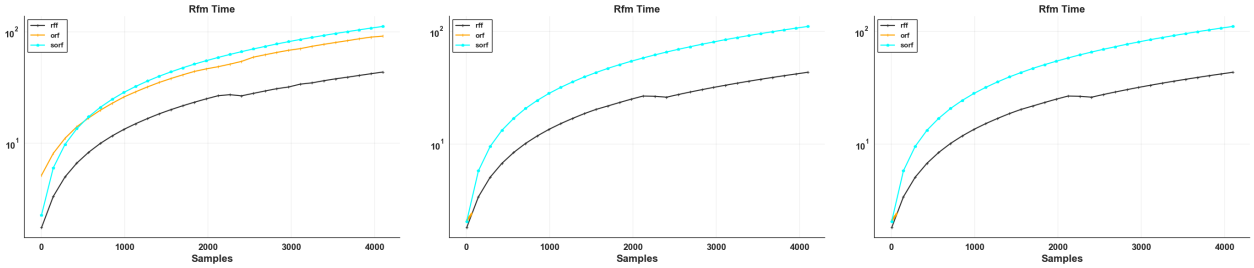
FIGURE 63. RFF Frobenius errors for the Wine data set.



(A) $\sigma = 1.0$ (B) $\sigma = 2.1$ (C) $\sigma = 10.0$

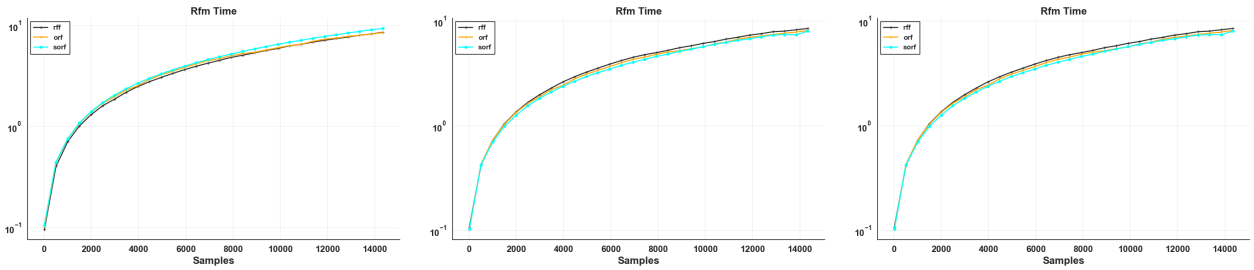
FIGURE 64. RFF infinity errors for the Wine data set.

A.9. **RFF Times.**



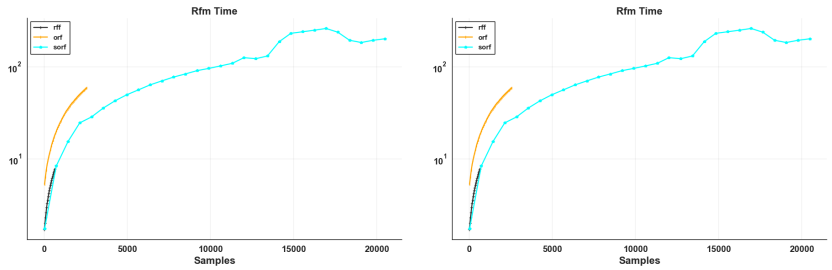
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 65. RFF approximation times for the 3DSN data set.



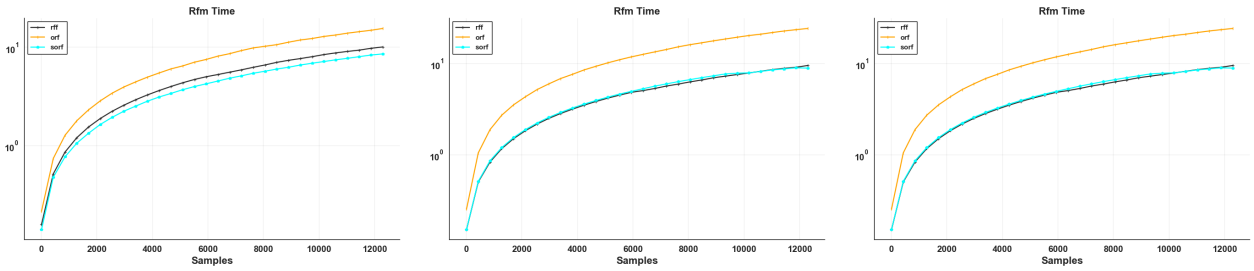
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 66. RFF approximation times for the Abalone data set.



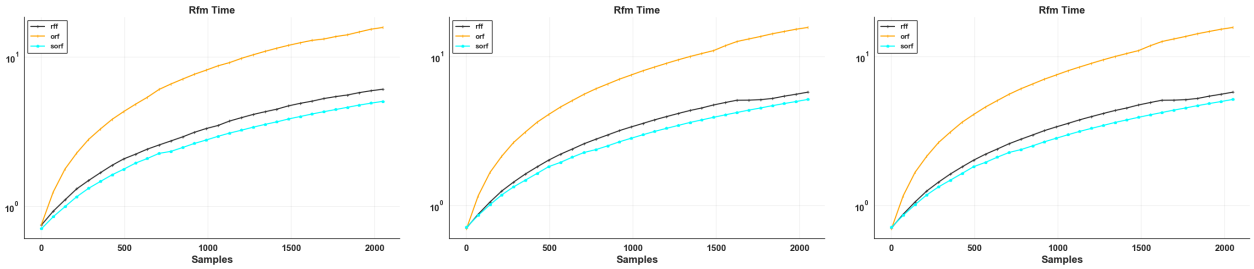
(A) $\sigma = 1.0$ (B) $\sigma = 10.0$

FIGURE 67. RFF approximation times for the magic04 data set.



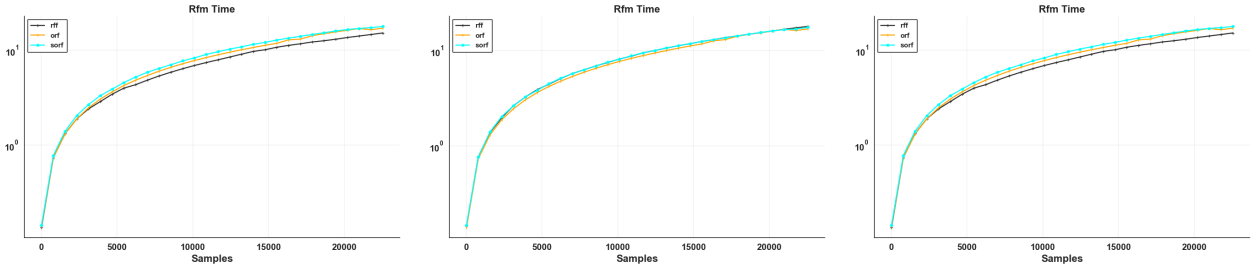
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 68. RFF approximation times for the Stocks data set.



(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

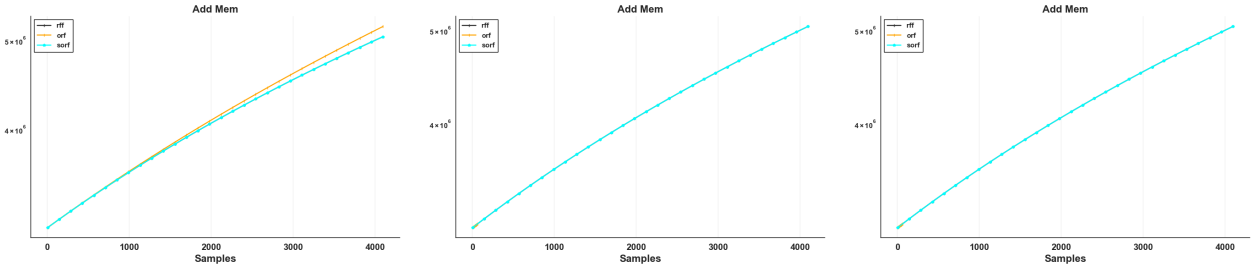
FIGURE 69. RFF approximation times for the Temp data set.



(A) $\sigma = 1.0$ (B) $\sigma = 2.1$ (C) $\sigma = 10.0$

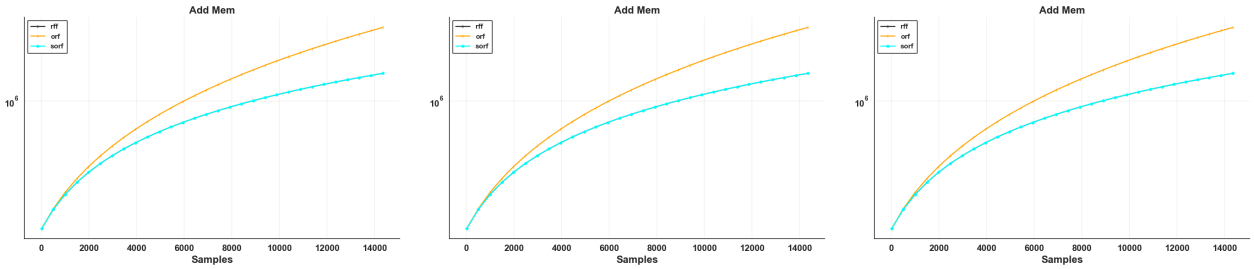
FIGURE 70. RFF approximation times for the Wine data set.

A.10. **RFF Additional Memory Usage.**



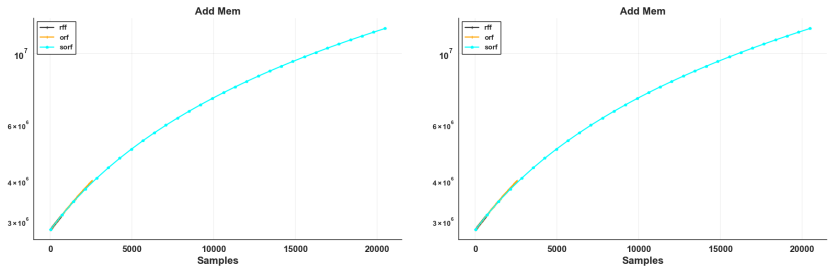
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 71. RFF additional memory usage for the 3DSN data set.



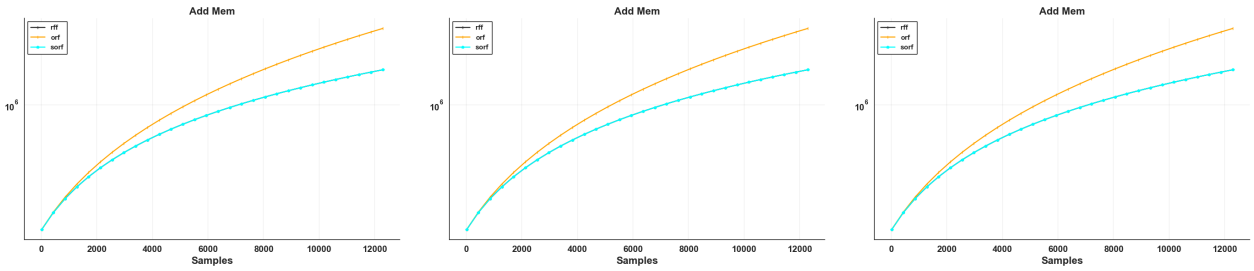
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 72. RFF additional memory usage for the Abalone data set.



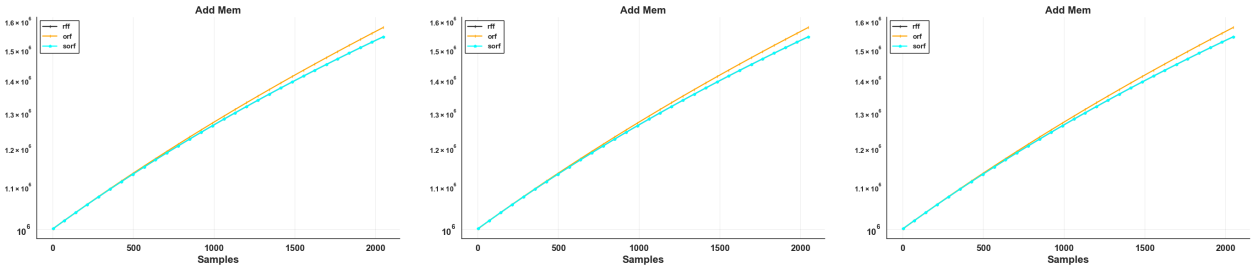
(A) $\sigma = 1.0$ (B) $\sigma = 10.0$

FIGURE 73. RFF additional memory usage for the magic04 data set.



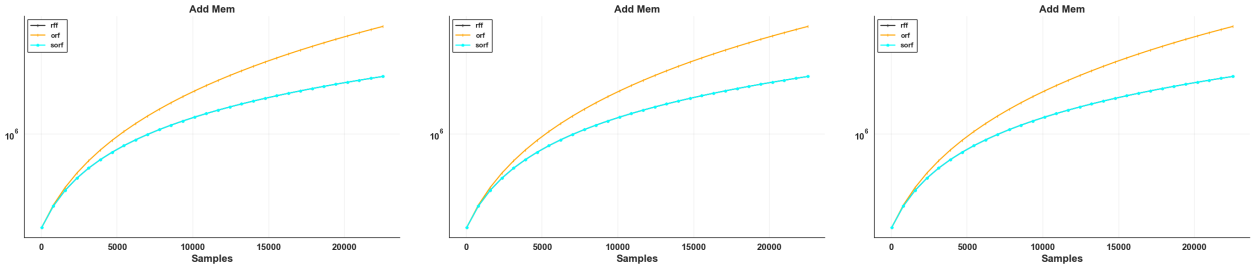
(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

FIGURE 74. RFF additional memory usage for the Stocks data set.



(A) $\sigma = 0.1$ (B) $\sigma = 1.0$ (C) $\sigma = 10.0$

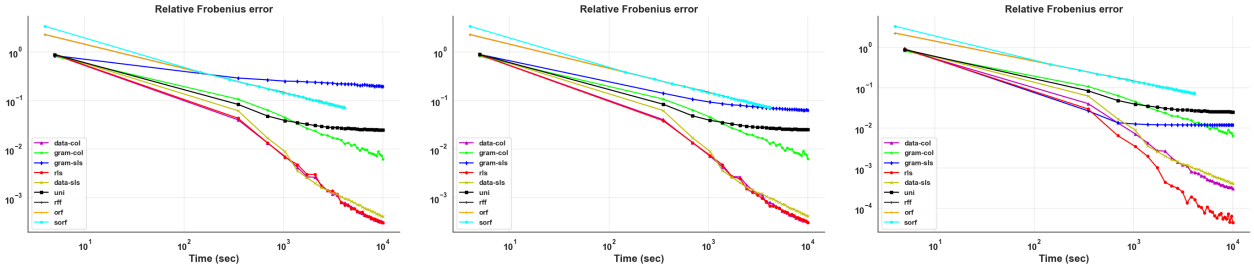
FIGURE 75. RFF additional memory usage for the Temp data set.



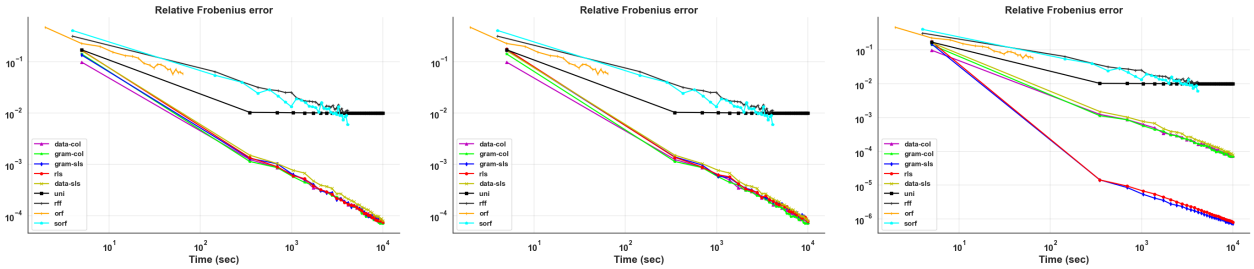
(A) $\sigma = 1.0$ (B) $\sigma = 2.1$ (C) $\sigma = 10.0$

FIGURE 76. RFF additional memory usage for the Wine data set.

A.11. **Kernel Comparisons.**

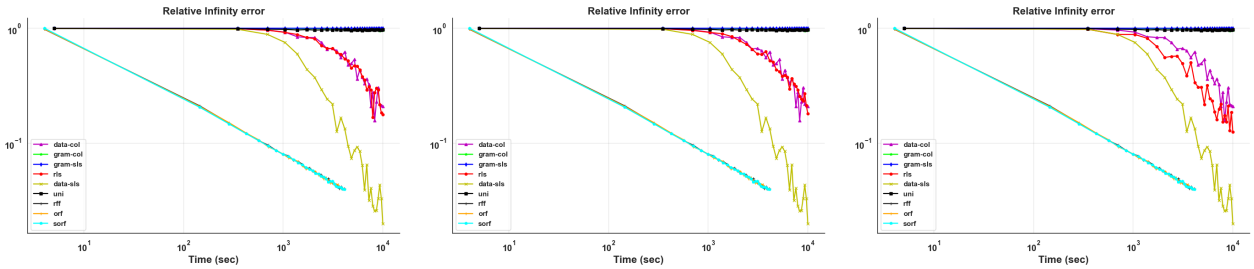


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

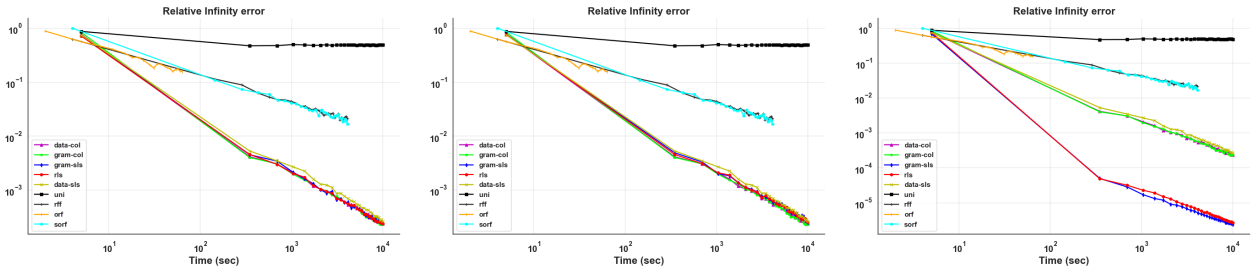


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

FIGURE 77. Comparing Frobenius errors of kernel methods for the 3DSN data set.

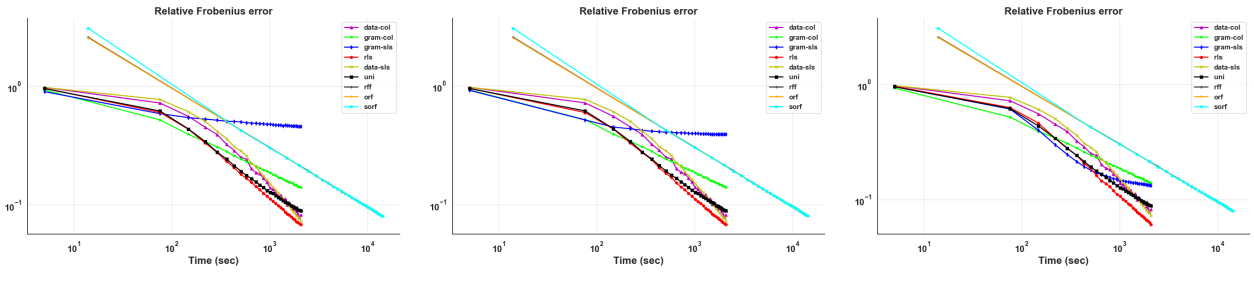


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

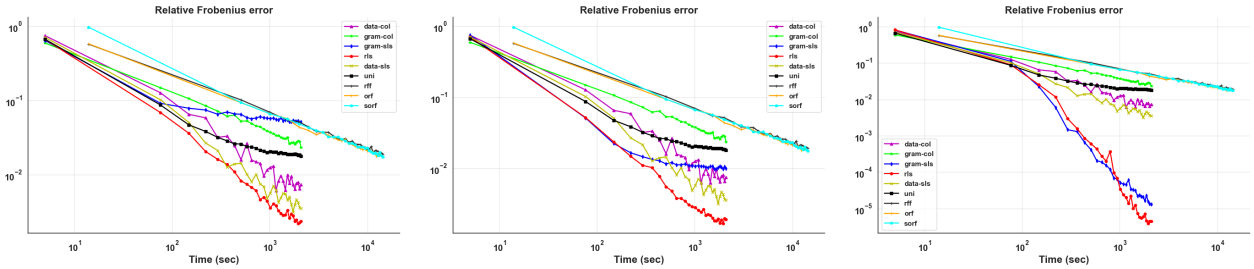


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 40$

FIGURE 78. Comparing infinity errors of kernel methods for the 3DSN data set.

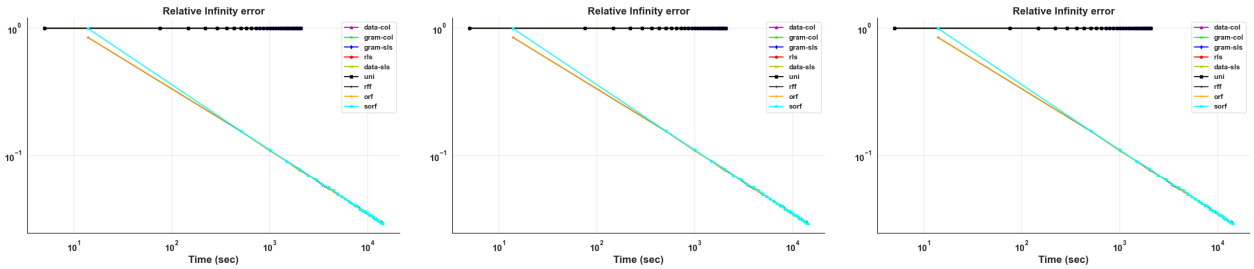


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

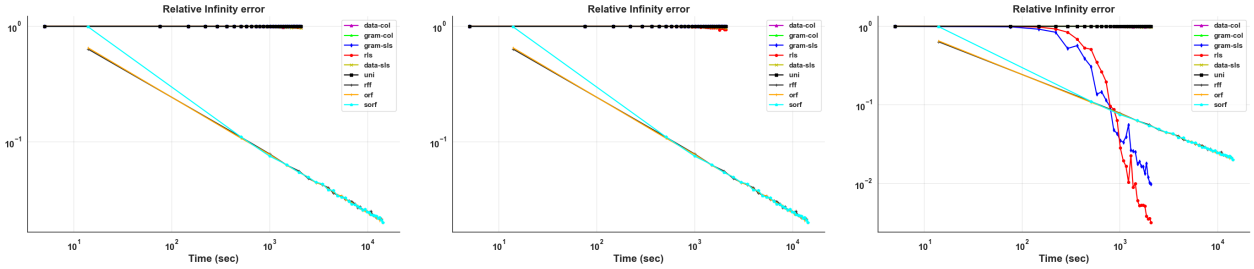


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 79. Comparing Frobenius errors of kernel methods for the Abalone data set.

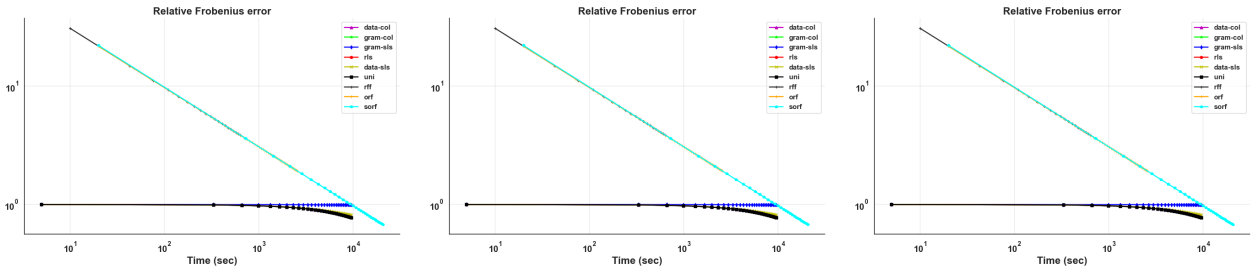


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$



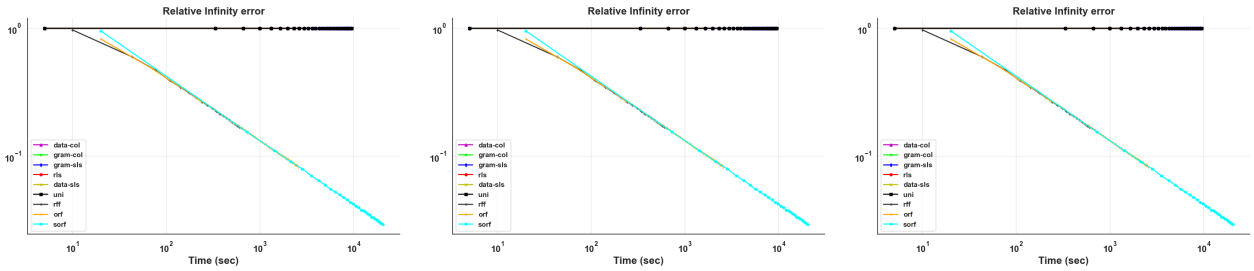
(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 80. Comparing infinity errors of kernel methods for the Magic data set.



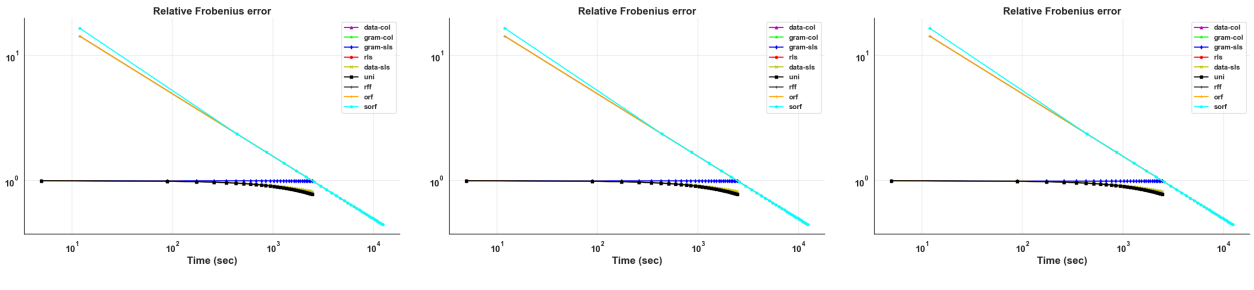
(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

FIGURE 81. Comparing Frobenius errors of kernel methods for the Magic data set.

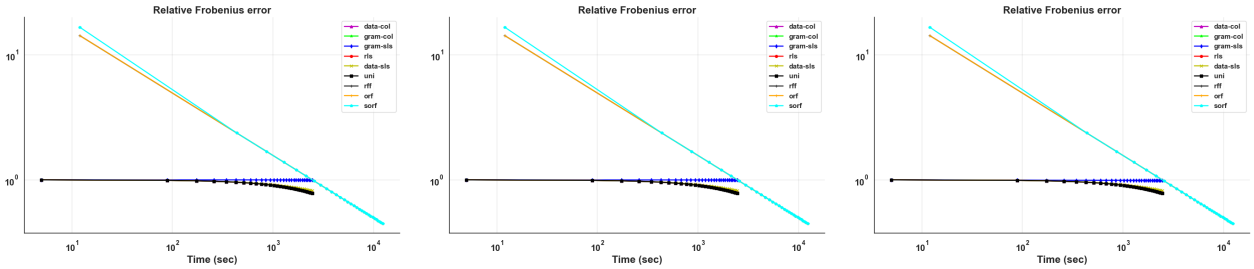


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

FIGURE 82. Comparing infinity errors of kernel methods for the Magic data set.

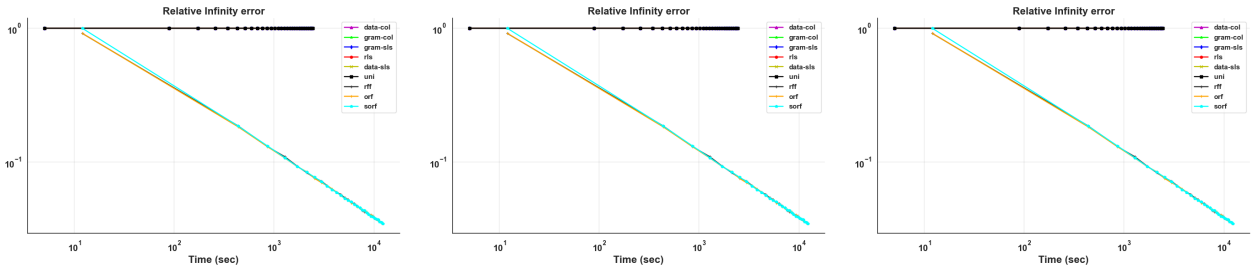


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

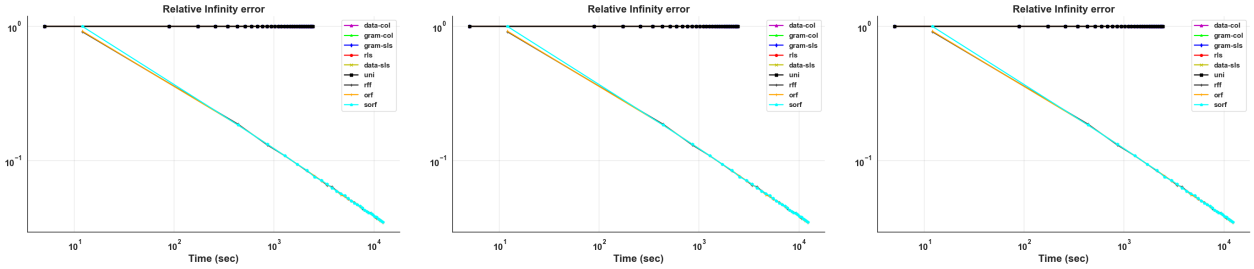


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 83. Comparing Frobenius errors of kernel methods for the Stocks data set.

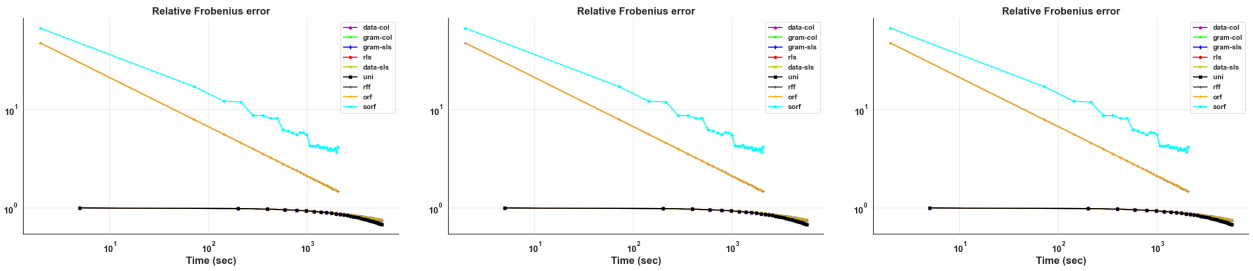


(A) $\sigma = 0.1,\ k = 10$ (B) $\sigma = 0.1,\ k = 20$ (C) $\sigma = 0.1,\ k = 80$

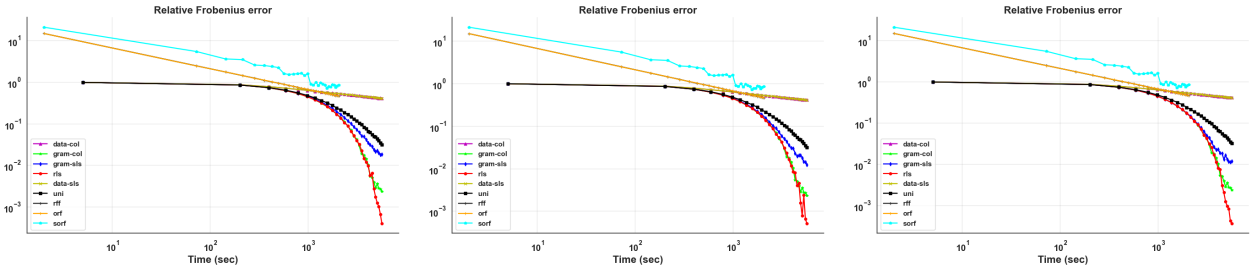


(D) $\sigma = 1.0,\ k = 10$ (E) $\sigma = 1.0,\ k = 20$ (F) $\sigma = 1.0,\ k = 80$

FIGURE 84. Comparing infinity errors of kernel methods for the Stocks data set.

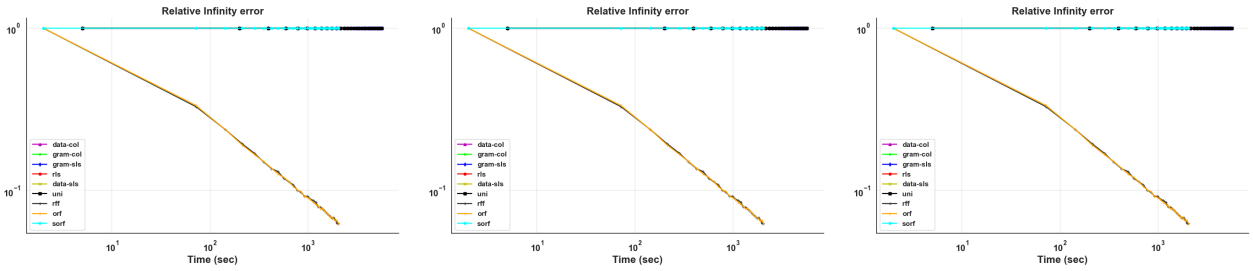


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

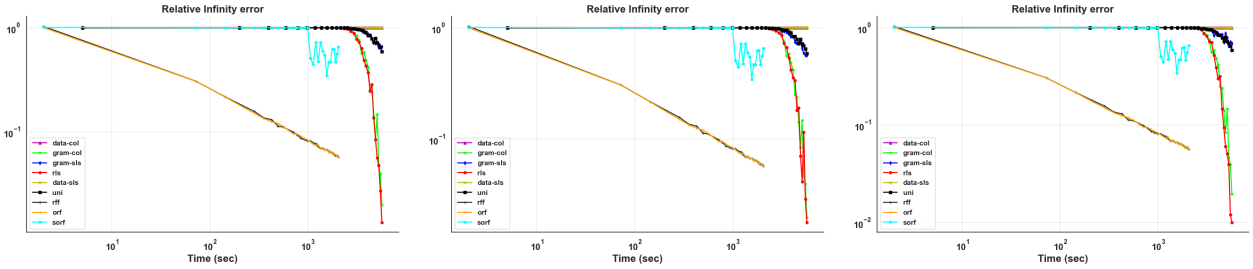


(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 85. Comparing Frobenius errors of kernel methods for the Temp data set.

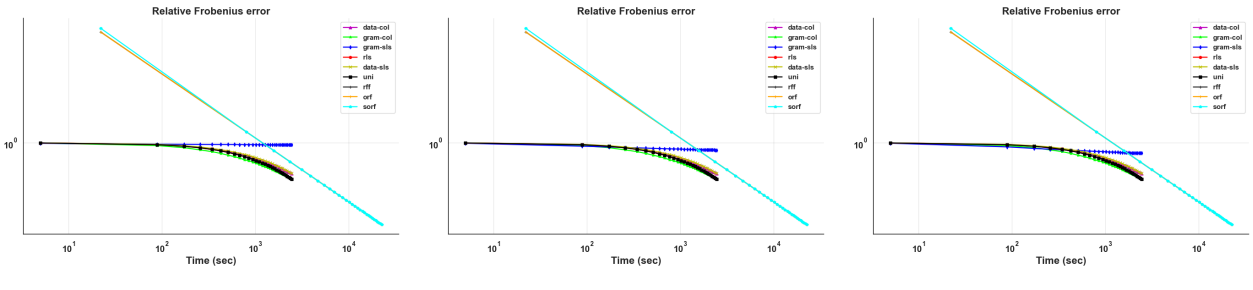


(A) $\sigma = 1.0,\ k = 10$ (B) $\sigma = 1.0,\ k = 20$ (C) $\sigma = 1.0,\ k = 80$

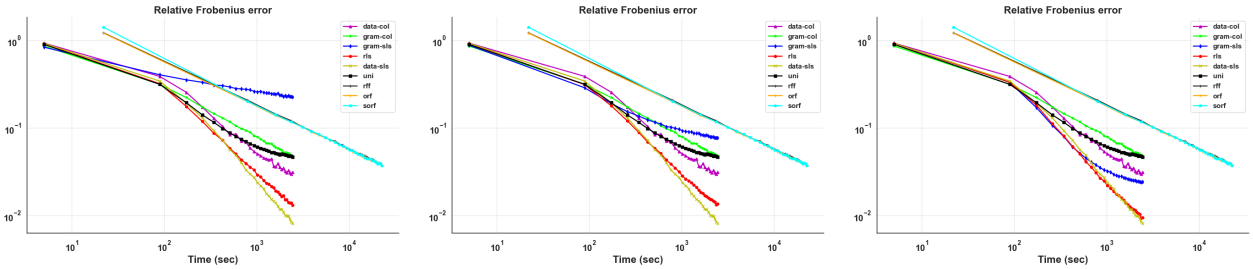


(D) $\sigma = 10.0,\ k = 10$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 86. Comparing infinity errors of kernel methods for the Temp data set.

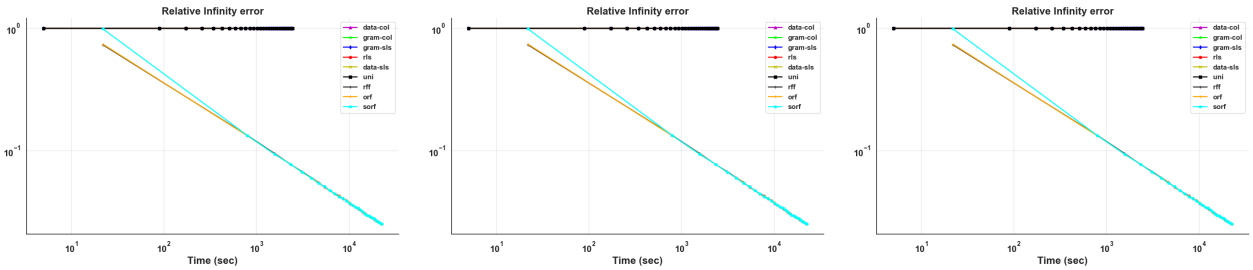


(A) $\sigma = 2.1,\ k = 5$ (B) $\sigma = 2.1,\ k = 20$ (C) $\sigma = 2.1,\ k = 40$

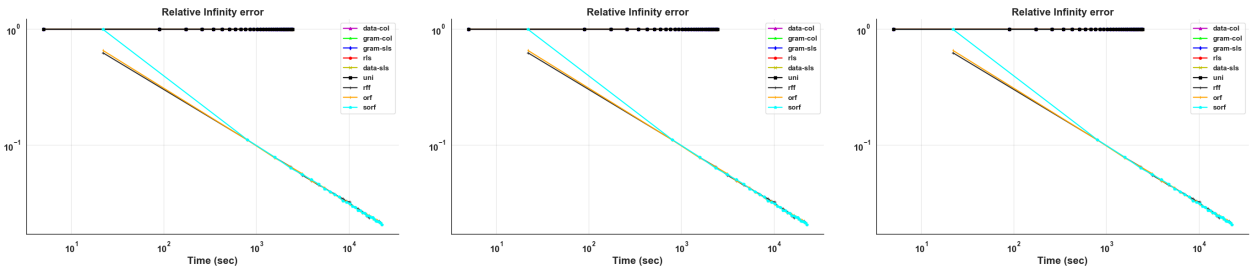


(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 87. Comparing Frobenius errors of kernel methods for the wine data set.



(A) $\sigma = 2.1,\ k = 5$ (B) $\sigma = 2.1,\ k = 20$ (C) $\sigma = 2.1,\ k = 40$



(D) $\sigma = 10.0,\ k = 5$ (E) $\sigma = 10.0,\ k = 20$ (F) $\sigma = 10.0,\ k = 80$

FIGURE 88. Comparing infinity errors of kernel methods for the wine data set.