Distribution Agreement

In presenting this thesis as a partial fulfillment of the requirements for an advanced degree from Emory University, I hereby grant to Emory University and its agents the non-exclusive license to archive, make accessible, and display my thesis in whole or in part in all forms of media, now or hereafter known, including display on the world wide web. I understand that I may select some access restrictions as part of the online submission of this thesis. I retain all ownership rights to the copyright of the thesis. I also retain the right to use in future works (such as articles or books) all or part of this thesis.

Wenxuan Cai April 9, 2024

Kernel and Lengthscale Selection on the Performance of the Sparse Cholesky Factorization Algorithm

By

Wenxuan Cai

Yuanzhe Xi
Advisor

Tianshi Xu
Advisor

Department of Mathematics

Yuanzhe Xi
Advisor

Tianshi Xu
Advisor

Julianne Chung
Committee Member

Alessandro Veneziani
Committee Member

2024

Kernel and Lengthscale Selection on the Performance of the Sparse Cholesky Factorization Algorithm

By

Wenxuan Cai

Yuanzhe Xi
Advisor

Tianshi Xu
Advisor

An abstract of
a thesis submitted to the Faculty of the Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Department of Mathematics

2024

Abstract

Kernel and Lengthscale Selection on the Performance of the Sparse Cholesky Factorization Algorithm
By Wenxuan Cai

In the realms of science and engineering, many challenges arise that demand the repeated solving of intricate systems of partial differential equations (PDEs) across various parameter values. Such scenarios are common in fields like molecular dynamics, micro-mechanics, and turbulent flows. Machine learning methods have shown promising capabilities in automating scientific computations, especially in PDE solving. Among these methods, Gaussian Process (GP) and kernel methods are notable for their interpretable and theoretically grounded function representation. In our study, we initially focused on the Sparse Cholesky-accelerated Gauss-Newton Algorithm [6]. However, we identified a need for further exploration regarding the choice of kernel and its parameters. Utilizing provided code, we conducted experiments to investigate the impact of different kernels and their lengthscales on accuracy, identifying optimal lengthscales. Moreover, we explored nonlinear elliptic PDEs, testing various solutions and observing limitations in achieving low relative error as high frequency terms became more significant, leading to non-convergence of some solutions. We also adjusted algorithm parameters and saw some accuracy improvements, although some questions still remain unanswered.

Kernel and Lengthscale Selection on the Performance of the Sparse Cholesky Factorization Algorithm

By

Wenxuan Cai

Yuanzhe Xi
Advisor

Tianshi Xu
Advisor

A thesis submitted to the Faculty of Emory College of Arts and Sciences
of Emory University in partial fulfillment
of the requirements of the degree of
Bachelor of Science with Honors

Department of Mathematics

2024

Acknowledgments

My thanks to Dr. Xi and Dr. Xu, my co-advisers, who guided me through the background knowledge and showed me the direction of the research when I got lost.

In addition to my other committee members, Julianne Chung and Alessandro Veneziani were very supportive throughout this project and gave me a lot of constructive suggestions, for which I am grateful.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Partial Differential Equations (PDEs) are mathematical expressions describing functions that rely on multiple independent variables. They incorporate these independent variables, the function itself, and partial derivatives of the function [12].

The most general form of a first-order PDE with two independent variables is expressed as [22]:

$$F(x, y, u(x, y), u_x(x, y), u_y(x, y)) = F(x, y, u, u_x, u_y) = 0$$

PDEs are prevalent across various disciplines such as mathematics, physics, chemistry, and biology, and find applications in numerous real-world scenarios.

Broadly, PDEs fall into two main categories: linear and non-linear. Within non-linear PDEs, there are various degrees of nonlinearity. Typically, tackling non-linear PDEs poses greater difficulty compared to their linear counterparts, particularly as the level of nonlinearity intensifies.

1.1 Linear PDE

Let $\mathcal{L}$ represent an operator. This implies that, for any function u, the expression $\mathcal{L}u$ denotes a new function. For instance, $\mathcal{L}$ could denote a partial derivative operation, such as $\mathcal{L} = \partial/\partial_x = u_x$, where it computes the partial derivative of u, or $\mathcal{L} = \partial_x + y\partial/\partial_z = u_x + yu_z$.

An operator $\mathcal{L}$ qualifies as linear if and only if the following two equations hold true for any functions u and v, and any constant c [22]:

$$\mathcal{L}(u+v) = \mathcal{L}u + \mathcal{L}v \qquad \mathcal{L}(cu) = c\mathcal{L}u$$

Linear operators possess several advantageous properties, including the superposition principle, the method of eigenfunctions, and transform methods.

The superposition principle states that if $u_1, u_2, \ldots, u_n$ are solutions to the homogeneous equation $\mathcal{L}u = 0$, then for arbitrary constants $c_1, c_2, \ldots, c_n$, the linear combination $c_1u_1 + c_2u_2 + \ldots + c_nu_n$ is also a solution.

1.2 Non-Linear PDE

Nonlinear partial differential equations (PDEs) pose greater challenges compared to their linear counterparts as they might fail to converge to a satisfactory solution without an adequate initial guess [11]. Unlike linear equations, nonlinear ones do not adhere to the superposition principle, rendering traditional techniques like eigenfunction methods and transform methods inapplicable.

PDEs can be classified according to their varying degrees of nonlinearity. At the lowest level are linear equations, which exhibit no nonlinearity. Beyond these are semi-linear equations, quasi-linear equations, and fully nonlinear equations.

1.2.1 Semi-linear equations

The most basic form of nonlinear PDEs is represented by semilinear equations. In semilinear equations, the nonlinear components do not manifest in the leading-order terms of the equation [21]. They can be expressed as follows:

$$\sum_{|\alpha|=k} c_\alpha(x) D^\alpha u + c_0(x, \nabla^{k-1} u, \ldots, \nabla u, u) = 0$$

An instance of such an equation is the reaction-diffusion equation:

$$u_t = u_{xx} + u^2$$

Here the leading order is u_{xx}, which is also linear, and u^2 is the nonlinear part, thus it is a semilinear equation.

1.2.2 Quasi-linear equations

Following semilinear equations are quasilinear equations, where the highest-order terms has the nonlinear coefficients. The constrain is that these coefficients cannot rely on the highest-order derivatives [21]:

$$\sum_{|\alpha|=k} c_\alpha(x, \nabla^{k-1} u, \ldots, \nabla u, u) D^\alpha u + c_0(x, \nabla^{k-1} u, \ldots, \nabla u, u) = 0$$

An example of such an equation is Burgers' equation:

$$u_t + u u_x = \nu u_{xx}$$

Here the leading order term is νu_{xx}, where ν is the diffusion coefficient or kinematic viscosity, which is generally nonlinear. Thus, it is a quasilinear equation.

1.2.3 Fully nonlinear equations

And finally, we have the hardest equations: fully nonlinear equations, which have the form:

$$\sum_{|\alpha|=k} c_\alpha(x, \nabla^k u, \ldots, \nabla u, u) D^\alpha u + c_0(x, \nabla^{k-1} u, \ldots, \nabla u, u) = 0$$

An example of such an equation is the Monge-Ampère equation [3]:

$$\det(D^2 u) = f, x \in (0,1)^2$$

Here, $D^2 u$ denotes the Hessian matrix of the function u, and its determinant is equal to f.

1.3 Classical numerical methods for solving nonlinear PDEs

There are several classical numerical methods for solving nonlinear PDEs, with three of the most notable being Finite-Difference Methods (FDM), Finite Element Methods (FEM), and Finite Volume Methods (FVM).

1.3.1 Finite-difference methods

Finite-difference methods (FDM) constitute a set of numerical approaches employed to solve differential equations through the estimation of derivatives using finite differences. These methods find applications in solving both ordinary differential equations (ODEs) and partial differential equations (PDEs). The approximations of derivatives stem from the Taylor series. For instance, considering a one-dimensional problem on a uniform grid with a mesh size h, we can express the Taylor series expansion at $y(x)$

as follows:

$$y(x+h) = y(x) + y'(x)h + \frac{1}{2}y''(\xi_1)h^2, \quad y(x-h) = y(x) - y'(x)h + \frac{1}{2}y''(\xi_2)h^2.$$

Thus, the first order derivative can be approximated as

$$\frac{\partial y}{\partial x} \approx \frac{y_{i+1} - y_{i-1}}{2h}.$$

Let's consider the following 1D model problem (ODE) as an example.

For this differential equation

$$y'' + (y')^2 + y = ln(x) \quad 1 \leq x \leq 2, \quad y(1) = 0, \quad y(2) = \ln 2.$$

with $u = 0$ at boundary in $[1, 2]$, and we want to approximate the actual solution for any value in this interval.Thus we use the discretization on a uniform mesh, which can be written as

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} + (\frac{y_{i+1} - y_{i-1}}{2h})^2 + y_i = \ln(1 + ih) \quad i = 1, 2, \ldots, n.$$

We can then solve this problem using Newton's method.

1.3.2 Finite-element methods

Finite element methods (FEM) offer an approach where the governing equations are approximated piecewise, assuming that a solution region can be represented analytically or by discretizing it into discrete elements [13].

In certain scenarios, FEM proves more suitable than the FD method. As illustrated in Figure 1.1 from [13], while a uniform finite difference mesh may adequately cover the blade, the boundaries need to be approximated by a series of horizontal and vertical

lines. In contrast, the finite element model provides a more accurate representation of the boundary shape.

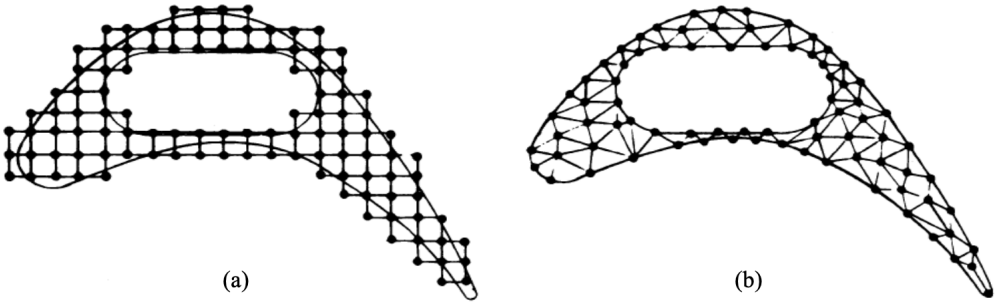


Figure 1.1: Demonstration of FD method on graph (a), and FEM method on graph (b). The solution region is the turbine blade profile. Derived from [13].

1.4 Machine learning methods

Those classical numerical methods of solving PDEs with computers are a great starting point. However, they have some drawbacks. It can take a very long time to calculate accurate answers for complex PDEs, especially if things are changing over time. Also, if we want to change the boundaries of the problem or the way we've divided it up, we often need to start the calculations all over again, which makes it hard to try out different ideas during product design.

Machine learning algorithms offer a potential alternative to address some of the limitations of classical numerical methods. A significant advantage of many machine learning algorithms is their mesh-free nature, which makes them more adaptable to changes in problem mesh. Additionally, they often provide a much faster time-to-solution compared to classical methods. However, it's important to note that machine learning-based PDE solvers can have limitations in terms of accuracy and may require long time to train.

There are two primary approaches to solving PDEs using machine learning techniques. Supervised learning methods operate akin to traditional function approximation, trained on a dataset consisting of PDEs and their corresponding solutions. These data-driven approaches aim to capture the intricate mapping between a PDE and its solution. Notable examples in this category include Fourier Neural Operators (FNOs) [16] and Deep Operator Networks (DeepONets) [17].

In contrast, unsupervised learning methods, such as certain physics-informed neural networks and Gaussian Process methods, leverage knowledge of the underlying physical laws inherent in the PDE itself. These techniques typically necessitate minimal or no datasets.

In this thesis, our focus lies on the unsupervised learning approach.

1.4.1 Neural Networks

One of the most widely recognized unsupervised learning algorithms based on neural networks is the Physics-Informed Neural Network (PINN). Recent advancements in PINN models, such as enhancing neural operators to learn mappings between function spaces [15] and devising a comprehensive framework for data-driven approximation of input-output mappings between infinite-dimensional spaces [2], have enabled the provision of expressive function representations.

However, compared to traditional solvers like the Finite Element Method (FEM), PINN typically requires more time and exhibits lower accuracy [9]. Consequently, efforts have been directed towards stabilizing and accelerating the training process [14, 23, 24, 8, 27].

1.4.2 GP and kernel methods

On the contrary, GP and kernel methods provide a more interpretable and theoretically grounded function representation rooted in the Reproducing Kernel Hilbert Space

(RKHS) theory [25, 1, 18], which integrates hierarchical kernel learning [26, 19, 5, 7]. However, the scalability of these methods is often limited by the dense kernel matrix.

Unlike other problems that primarily focus on point-wise values, in PDE problems, these kernel matrices may also encompass partial derivative values [4]. For the Sparse Cholesky factorization algorithm [6], computing such matrices incurs a complexity of $O(N \log^d(N/\epsilon))$ in space and $O(N \log^{2d}(N/\epsilon))$ in time.

1.5 Contributions

In this thesis, we review the state-of-the-art GP based algorithm for solving non-linear PDEs. Through extensive experimentation and analysis, we provided valuable insights into the influence of kernel and lengthscale choices, the significance of high-frequency terms, and the impact of the number of preconditioned conjugate gradient (pCG) steps on algorithm performance. These contributions advance our understanding of GP-based approaches to non-linear PDE solving and pave the way for further research and development in this domain.

The remaining chapters are organized as follows:

In Chapter 2, we introduces the framework for solving nonlinear PDEs using Gaussian Process [6]. The purpose of this thesis is then stated: to address three main questions. Firstly, whether the choice of kernel and lengthscale in the GP framework affects algorithm performance. Secondly, whether increasing the significance of high-frequency terms influences algorithm performance. Lastly, whether the number of Gauss-Newton iterations (GN steps) have an impact on algorithm performance. These questions are further expanded upon in Chapter 3.

To achieve this, experiments are conducted in Chapter 4, which is divided into three parts. Firstly, we investigate whether the selection of kernel and lengthscale influences accuracy, and determine a reasonable range for these parameters based

on sample functions provided in [6]. Next, we shift our focus to nonlinear elliptic PDEs, observing algorithm performance as we reduce the significance or truncate low-frequency terms. Finally, we evaluate whether parameters such as Gauss-Newton steps that can improve algorithm performance.

In Chapter 5, the discoveries and conclusions drawn from the experiments are summarized.

Chapter 2

Overall Process to solve non-linear PDE

2.1 Solving nonlinear PDEs via GPs

To solve a non-linear PDE, like nonlinear elliptic PDEs

$$\begin{cases} -\Delta u + \tau(u) = f & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega, \end{cases} \tag{2.1}$$

where Δ is Laplace Operator, Ω means the domain, and $\partial\Omega$ means the boundary.

The initial step involves sampling collocation points within the interior denoted by $x_\Omega = x_1, \ldots, x_{M_\Omega} \subset \Omega$, along with $M_{\partial\Omega}$ points on the boundary denoted by $x_{\partial\Omega} = x_{M_\Omega+1}, \ldots, x_M \subset \partial\Omega$.

After that, a Gaussian Process (GP) prior is assigned to the unknown function u, resulting in:

$$\begin{cases} \textbf{minimize}_{u\in\mathcal{U}}\|u\| \\ -\Delta u(x_m) + \tau(u(x_m)) = f(x_m) \quad \text{for } m = 1,\dots,M_\Omega, \\ u(x_m) = g(x_m) \qquad\qquad\qquad\quad \text{for } m = M_\Omega+1,\dots,M. \end{cases} \tag{2.2}$$

Here, $\|\cdot\|$ is the Reproducing Kernel Hilbert Space (RKHS) norm corresponding to the kernel/covariance function K.

Subsequently, leveraging a generalization of the Representer theorem [4], the minimizer of (2.2) takes the form:

$$u^\dagger(x) = K(x,\phi)K(\phi,\phi)^{-1}z^\dagger \tag{2.3}$$

where $z^\dagger$ is a solution of a finite-dimensional problem that satisfies

$$\begin{cases} \underset{z\in\mathbb{R}^{M+M_\Omega}}{minimize} & z^T K(\phi,\phi)^{-1}z \\ \text{s.t.} & -z_m^{(2)} + \tau(-z_m^{(1)}) = f(x_m), \text{ for } m = 1,\dots,M_\Omega \\ & z_m^{(1)} = g(x_m), \text{ for } m = M_\Omega+1,\dots,M \end{cases} \tag{2.4}$$

When consider the general PDEs, the methodology leads to the optimization problem

$$\begin{cases} \min\limits_{u\in\mathcal{U}} & \| u \| \\ \text{s.t.} & \text{PDE constraints at} \{x_1,\dots,x_M\} \in \bar{\Omega} \end{cases} \tag{2.5}$$

and the equivalent to this finite dimensional problem:

$$\begin{cases} \underset{z\in\mathbb{R}^N}{minimize} & z^T K(\phi,\phi)^{-1}z \\ \text{s.t.} & F(z) = y \end{cases} \tag{2.6}$$

As we discussed in the background, K is the kernel. Since we intend to deal with K^{-1}, computing K^{-1} with Gauss Elimination results in $O(N^3)$ space/time complexity.

They changed the algorithm, and in the end they get

$$K(\phi,\phi)^{-1} \approx P_{perm}^{T} U^{\rho} U^{\rho^{T}} P_{perm} \tag{2.7}$$

where P_{perm} is a permutation matrix, and U^{ρ} is a sparse matrix that approximates the Cholesky factorization of the kernel matrix after the permutation.

2.2 Sparse Cholesky factorization algorithm

To get the approximation (2.7), they outline the general algorithmic procedure in Algorithm 1.

Algorithm 1 Sparse Cholesky Factorization for $K(\phi,\phi)^{-1}$

1: **Input:** Measurements ϕ, kernel function K, sparsity parameters ρ, supernodes parameter λ
2: **Output:** U^{ρ}, P_{perm} such that $K(\phi,\phi)^{-1} \approx P_{perm}^{T} U^{\rho} U^{\rho^{T}} P_{perm}$
3: **Step 1:** Reorder measurements
4: Order Dirac measurements using maximin ordering.
5: Order derivative measurements arbitrarily.
6: Obtain permutation matrix P_{perm} such that $P_{perm}\phi = \tilde{\phi}$.
7: Find the lengthscales l for each measurement in $\tilde{\phi}$.
8: Construct aggregate sparsity pattern $S_{P,l,\rho,\lambda}$ using parameters
9: **Step 2:** Solve KL minimization
10: Solve (2.11) with $\Theta = K(\tilde{\phi},\tilde{\phi})$ to obtain U^{ρ}.
11: **Return:** U^{ρ}, P_{perm}

Overall, they initially establish the ordering of the measurements to acquire the permutation matrix P_{perm}, followed by determining which entries of the approximation U^{ρ} should be non-zero. Finally, they optimize the values of those non-zero entries.

2.2.1 Ordering of the measurement

In order to render formula (2.2) computationally tractable by transforming it into a finite-dimensional problem, specific notations are introduced for measurements [6]:

The measurement functions are defined as follows:

$$\phi_m^{(1)} = \delta_{x_m}, 1 \leq m \leq M \quad \text{and} \quad \phi_m^{(2)} = \delta_{x_m} \circ \Delta, 1 \leq m \leq M_\Omega,$$

where δ_x is the Dirac delta function centered at x. They are in $\mathcal{U}$, the dual space of $\mathcal{U}$, for sufficiently regular kernel functions. Those measurements contains local functions values and derivatives, and are used to train the physical-informed GP.

When arranging the measurements, the maximin ordering strategy [10] is employed to order non-derivative types of measurements.

The maximin ordering strategy selects the initial point at the center, which could be determined by the mean location or another measure of centrality. Subsequently, it iteratively selects the next point to maximize the minimum distance from all previously selected points, as described by [10]:

$$\begin{aligned} \tau(1) &= \arg \min_{i \in 1,\ldots,n} \parallel x_i - \bar{x} \parallel, \\ \tau(j) &= \arg \max_{i \notin \tau(1),\ldots,\tau(j-1)} \min_{k \in 1,\ldots,j-1} \parallel x_i - x_{\tau(k)} \parallel, \quad j > 1. \end{aligned} \tag{2.8}$$

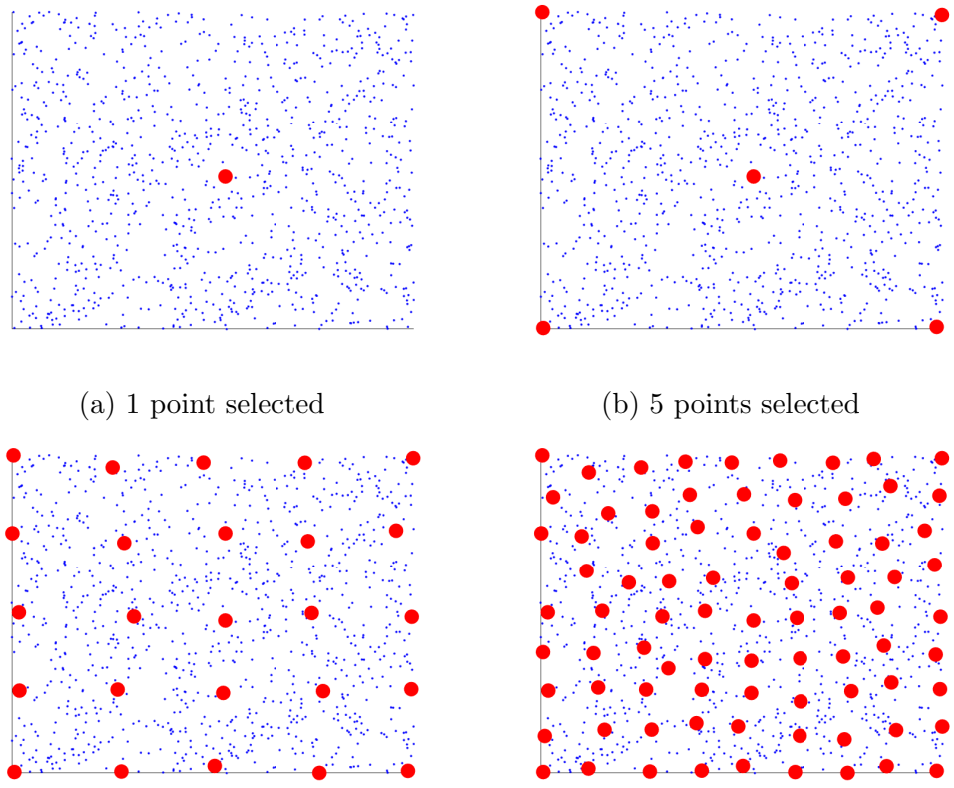
A Schematic diagram is shown in Figure 2.1, an advantage of this ordering the selected points always evenly distributed.

In [6], they want to select two groups of points: interior and boundary. They revised the strategy, and use the revised definition of the Maximin Ordering (2.2.1[6])

Definition. (Conditioned Maximin Ordering)

The Maximin ordering conditioned on a set A for points $x_i, i \in I$ is achieved by iteratively selecting the point x_i that is farthest from A and the previously chosen points. If A is an empty set, we designate an arbitrary index $i \in I$ as the starting point. Otherwise, we select the first index as

$$i_1 = \arg \max_{i \in I} dist(x_i, A).$$



(a) 1 point selected

(b) 5 points selected

(c) 25 point selected

(d) 81 point selected

Figure 2.1: Demonstration of maximum-minimum distance ordering. This graph is generated through maximum-minimum distance ordering. From graph (a) to graph (d), each subsequent graph is based on the previous one. Each point represents a sample point, and each red point indicates that it has been selected. At any given stage of the ordering process, the selected points exhibit homogeneity.

For the first q indices already chosen, we choose

$$i_{q+1} = \arg \max_{i \in I \setminus \{i_1, \ldots, i_q\}} \operatorname{dist}(x_i, \{x_{i+1}, \ldots, x_{i_q}\} \cup A). \tag{2.9}$$

In this scenario, they designate the set $A = \emptyset$ when choosing boundary conditions and $A = \partial\Omega$ when selecting interior points. Consequently, during the selection of interior points, the Conditioned Maximin Ordering process identifies each unselected point by maximizing its distance from both the boundary and previously selected points.

After they use the Conditioned Maximin Ordering to order the non-derivative type of measurement, they use an arbitrary ordering to order the derivative type of measurement. This kind of permutation will generate a new order of those measurements, thus they can have the permutation matrix P_{perm} accordingly. We denote the permuted kernel matrix as $K(\tilde{\phi}, \tilde{\phi})$.

2.2.2 Select non-zero entries of U^ρ

Next, they define the lengthscale parameter l_j of each ordered point as the shortest distance from the corresponding measurement to other measurements, which has the formula [6]

$$l_i = \text{dist}(x_{P(i)}, x_{P(1)}, ..., x_{P(i-1)} \cup A). \tag{2.10}$$

To establish the selection radius, they define the parameter ρ. For every column of U^ρ, only rows within a distance of ρl_j from this measurement can have non-zero values. This approach ensures that the non-zero entries in the matrix are sparse, as dictated by the algorithm.

2.2.3 Optimize U^ρ

Finally, they use Kullback-Leibler (KL) minimization to have the approximation $K(\tilde{\phi}, \tilde{\phi}) \approx U^\rho {U^\rho}^\top$, which minimize

$$U = \arg \min_{\hat{U} \in S_{P,l,\rho}} KL\Big(\mathcal{N}(0, K(\tilde{\phi}, \tilde{\phi})) \parallel \mathcal{N}(0, (\hat{U}\hat{U}^T)^{-1})\Big) \tag{2.11}$$

In other words, instead of approximating the matrix based on the matrix norm, they put both matrices into two multivariable normal distributions and measure the difference between both distributions.

Based on Theorem 2.1 in [20], j-th column of the matrix U can be represented as

$$U_{s_j,j} = \frac{(K(\tilde{\phi},\tilde{\phi})_{s_j,s_j})^{-1} e_{\#s_j}}{\sqrt{e_{\#s_j}^T (K(\tilde{\phi},\tilde{\phi})_{s_j,s_j})^{-1} e_{\#s_j}}} \tag{2.12}$$

where $\#s_j$ is the number of non-zero entry in j-th column. $e_{\#s_j} \in \mathbb{R}^{\#s_j}$ is the standard basis vector with the last entry being 1 and other entries equal 0.

In this algorithm, there is no construction of a kernel matrix. Also, based on Theorem 4.2 in [6], this algorithm has a provably accurate sparse factors when $\rho = O(\log(N/\epsilon))$.

2.3 Second order optimization methods

Based on the previous algorithm, we obtain the approximation of the inverse of the kernel matrix K using formula (2.7). Subsequently, the Gauss-Newton algorithm is employed to compute z, as outlined in Algorithm 3.

By incorporating the nonlinear constraint, equations (2.4) or (2.6) can be reformulated as an unconstrained problem. Initially, Gauss-Newton iterations are utilized.

Below are the Gauss-Newton iterations for equation (2.6):

$$z_{k+1} = K(\phi,\phi)(DF(z^k))^T \gamma \tag{2.13}$$

where $\gamma \in \mathbb{R}^M$ satisfies

$$K(\phi^k,\phi^k)\gamma = y - F(z^k) + DF(z^k)z^k \tag{2.14}$$

Here, z^k means the vector z in k-th iteration, ϕ^k is the reduced set of measurements, which has the equation $\phi^k = DF(z^k)\phi$. $DF(z^k) \in \mathbb{R}^{M\times N}$ is the Jacobian of F at z^k.

Now the dimension is reduced. The computational bottleneck lies in the linear system with the reduced kernel matrix $K(\phi^k,\phi^k)$.

Now they use similar method to obtain the approximation of $K(\phi^k,\phi^k)$, they apply

Algorithm 2 Sparse Cholesky Factorization for $K(\phi^k, \phi^k)^{-1}$

Input: Measurements ϕ^k, kernel function K, sparsity parameters ρ_r, ρ_r, supernodes parameter λ
Output: Solution $U_r^{\rho_r}$, Q_{perm}
Step 1: Reorder and create sparsity pattern
  Order boundary measurements using maximin ordering.
  Order interior measurements using maximin ordering conditioned on $\partial\Omega$.
  Obtain permutation matrix Q_{perm} based on ordering such that $Q_{perm}\phi^k = \tilde{\phi}^k$.
  Calculate lengthscales l for each measurement in $\tilde{\phi}^k$.
  Construct aggregate sparsity pattern $S_{Q,l,\rho_r,\lambda}$.
Step 2: Solve KL minimization
  Solve (2.11) with $\Theta = K(\tilde{\phi}^k, \tilde{\phi}^k)$.
  Use (2.12) to obtain $U_r^{\rho_r}$.
Return: $U_r^{\rho_r}$, Q_{perm}

Algorithm 2 here. However, since the dimension is reduced, this algorithm is different from Algorithm 1. One key thing is that they need to order boundary condition first. The reason is that the interior points are now containing derivative type of informations, whereas boundary points still don't.

Algorithm 3 Sparse Cholesky Accelerated Gauss-Newton for Solving (2.6)

Input: Measurements ϕ, data functional F, data vector $\mathbf{y}$, kernel function K, number of Gauss-Newton steps t, sparsity parameters ρ, ρ_r, supernodes parameter λ
Output: Solution $\mathbf{z}^t$
Factorize $K(\phi, \phi)^{-1} \approx P_{perm}^T U^\rho U^{\rho T} P_{perm}$ using Algorithm 1
Set $k = 0$, $\mathbf{z}^k = \mathbf{0}$ or other user-specified initial guess
while $k < t$ **do**
  Form the reduced measurements $\phi^k = DF(\mathbf{z}^k)\phi$
  Factorize $K(\phi^k, \phi^k)^{-1}$ to get $Q_{perm}^T U_r^{\rho_r} U_r^{\rho_r T} Q_{perm}$ using Algorithm 2
  Use PCG to solve (2.14) with the preconditioner $Q_{perm}^T U_r^{\rho_r} U_r^{\rho_r T} Q_{perm}$
  $\mathbf{z}^{k+1} = P_{perm}^T U^\rho U^{\rho T} P_{perm} \setminus ((DF(\mathbf{z}^k))^T)\gamma$
  $k = k + 2$
end while
return $\mathbf{z}^t$

After Algorithm 2, they apply pCG method, and with enough iteration, they get z.

Chapter 3

Our Purpose

The Sparse Cholesky factorization algorithm in [6] is innovative, providing a faster algorithm for solving the kernel matrix with derivative-type information. Nevertheless, it leaves room for discussion, most notably, in the selection of the kernel.

3.1 Kernel and Lengthscale

In [6], they used Matérn kernels with parameters $\nu = \frac{5}{2}$, $\nu = \frac{7}{2}$, $\nu = \frac{9}{2}$, $\nu = \frac{11}{2}$, and a Gaussian kernel. Let me first introduce those two kind of kernels.

The Matérn covariance between measurements taken at two points separated by d distance units is given by

$$C_\nu(d) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu} \frac{d}{\rho} \right)^\nu K_\nu \left(\sqrt{2\nu} \frac{d}{\rho} \right),$$

The Gaussian kernel is

$$\lim_{\nu \to \infty} C_\nu(d) = \sigma^2 \exp\left(-\frac{d^2}{2\rho^2} \right),$$

where Γ is the gamma function, K_ν is the modified Bessel function of the second kind, ρ is the lengthscale, and ν is a positive parameter that determines the smoothness of

the kernel. The kernel will be smoother as ν increases, and the Gaussian kernel is the case when $\nu = +\infty$.

For lengthscale ρ, they set parameters for each PDEs. However, they didn't explain the method and the reason they selected those lengthscales, which may influence the accuracy. Therefore, we aim to explore whether the lengthscale will influence the accuracy, and what could be a reasonable range for the functions provided in [6]. The corresponding experiment is conducted in Chapter 4.1.

3.2 Truth function of Nonlinear Elliptic PDEs

Upon observing the functions, we realized the unusual significance of the low-frequency terms in the "truth function" of the nonlinear elliptic PDEs equation (2.1).

They set the truth function $u(x)$ as

$$u(x) = \sum_{k=1}^{600} \frac{1}{k^6} \sin(k\pi x_1) \sin(k\pi x_2) \tag{3.1}$$

When $\sin(k\pi x_1) \sin(k\pi x_2)$ is a high frequency term, k is also large. Since this term is bounded by 1, when k is large, the term $\frac{1}{k^6} \sin(k\pi x_1) \sin(k\pi x_2)$ will be insignificant. Thus, we want to investigate the precision of the algorithm when we change the "truth solution", specifically, see its performance when high frequency term becomes significant.The corresponding experiment is conducted in Chapter 4.2.

3.3 GN Steps and other parameters

In Algorithm 3, they use the Gauss-Newton to get the preconditioner. They claim that pCG inside the Gauss-Newton iteration usually converges in 10-40 steps. However, there is no theoretical guarantee the factor is as accurate as before. Therefore, we want to explore whether the change in Gauss-Newton steps can have an influence on

the performance, especially when the algorithm not perform well.

We also want to going deep into their code to see if change in some of their parameters, like KNN and ρ, will help improve the algorithm. The corresponding experiment is conducted in Chapter 4.3.

Chapter 4

Experiments

Since our paper is largely based on [6], we use their code to process the experiment. Their code is at https://github.com/yifanc96/PDEs-GP-KoleskySolver. It is compiled in Julia, and some of its coefficients are changed during our experiment.

In the Numerical experiments part of [6], they used three equations to present the algorithm:

(1). Nonlinear elliptic PDEs, by using "main_MongeAmpere2d.jl" file.

(2). Burgers' equation, by using "main_Burgers1d.jl" file.

(3). Monge-Ampère equation in 2D space, by using "main_MongeAmpere2d.jl" file.

We are interested in the accuracy performance under those three equations as the hyper-parameter of the algorithm changes.

To better adapt our experiment to using his code, we made some changes. Different functions have different magnitudes, instead of the absolute L^2 error, we use the relative L^2 error to measure the preciseness. Also, we tested the lengthscale distributed exponentially between from 10^{-7} to 100, with each point representing an interval of $10^{0.1}$. These are the following results.

4.1 Kernel and Lengthscale influence on accuracy

We first see whether the selection of Kernel and Lengthscale will influence accuracy. In this part, every other hyper-parameter, if is not mentioned, takes the default value in [6]'s code.

4.1.1 Nonlinear elliptic PDEs

The Nonlinear elliptic PDEs are given by

$$\begin{cases} -\Delta u + \tau(u) = f & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega, \end{cases}$$

with $\tau(u) = u^3$. Here $\Omega = [0,1]^2$, and they set the truth function as $u(x) = \sum_{k=1}^{600} \frac{1}{k^6} \sin\left(k\pi x_1\right) \sin\left(k\pi x_2\right)$.

They set Gauss-Newton iterations steps to be 3, initial guess as zero function, and lengthscale of the kernels $\rho = 0.3$. After looping through the lengthscale for Nonlinear elliptic PDEs, the result is shown in Figure 4.1.

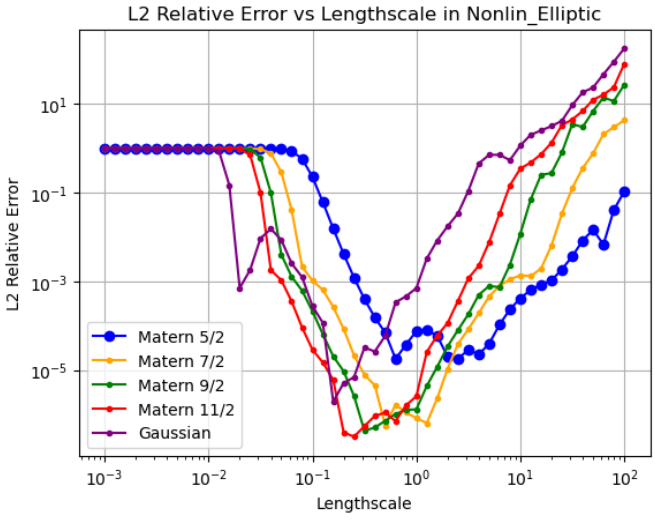


Figure 4.1: Demonstration of Lengthscale ρ's relationship with L^2 relative error for different kernels in Nonlinear Eliptic Equation. For lengthscale less than 10^{-3}, the L^2 relative error for all five kinds of kernel is 1. When lengthscale is greater than 100, the L^2 relative error keeps increasing, and the error will be greater than the result itself.

We find that the Matérn 7/2, 9/2 and 11/2 kernel performs roughly equally well. When the lengthscale is between 0.316 ($10^{-0.5}$) and 1, all three kernels have an error less than 10^{-5}. Thus the best lengthscale for the equation presented here is between 0.1 and 1.

We realized that the selected kernel can influence the error. Here, the error of the Matérn 5/2 kernel can't be less than 10^{-5}, while the Matérn 7/2, 9/2 and 11/2 kernel can achieve that at $\rho = 1$.

The lengthscale parameter ρ can also influence the error. When the lengthscale is small (less than 0.01 in this case), the relative error will be 1, which means zero vector; when it is large, the result will be unstable.

Also, for the different kernels, their best lengthscale interval is different. Matérn 7/2, 9/2 and 11/2 kernel performs best when the lengthscale is in [0.1,1], while it is [0.8, 5] for Matérn 5/2.

4.1.2 Viscous Burgers' equation

The Burgers' equation is

$$\begin{aligned} \partial_t u + u\partial_x u - 0.001 u \partial_x u = 0, \quad \forall (x,t) \in (-1,1) \times (0,1], \\ u(x,0) = -\sin(\pi x), \\ u(-1,t) = u(1,t) = 0. \end{aligned}$$

To solve such PDE, they discretize the equation in time using the Crank–Nicolson scheme with time stepsize Δt to obtain:

$$\begin{aligned} \frac{\hat{u}(x,t_{n+1}) - \hat{u}(x,t_n)}{\Delta t} + \frac{1}{2}(\hat{u}(x,t_{n+1}) + \hat{u}(x,t_n)\partial_x \hat{u}(x,t_n)) \\ = \frac{0.001}{2}(\partial_x^2 \hat{u}(x,t_{n+1}) + \partial_x^2 \hat{u}(x,t_n)) \end{aligned} \tag{4.1}$$

where $\hat{u}(t_n, x)$ is an approximation of the true solution $u(t_n, x)$ with $t_n = n\Delta t$.

Here, they set Gauss-Newton iterations to be 2, and the initial guess as the solution at the last time step. the lengthscale ρ is chosen to be 0.02. Our result is shown in Figure 4.2.

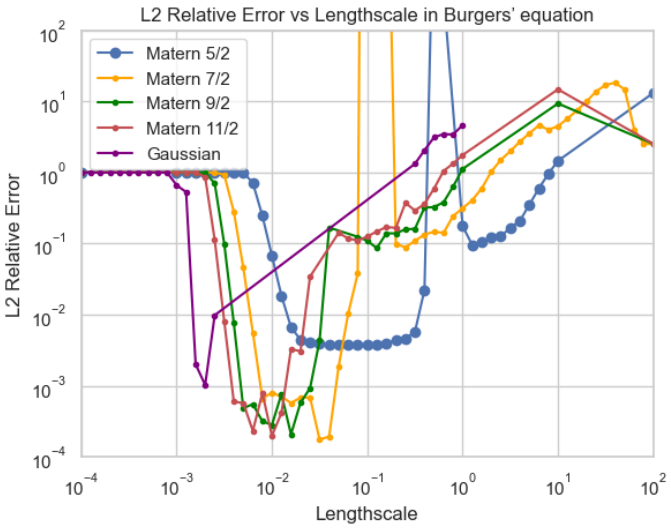


Figure 4.2: Demonstration of lengthscale ρ's relationship with L^2 relative error for different kernels in Burgers' equation. This graph only shows the lengthscale between 10^{-4} and 100, and L^2 relative error between 10^{-4} and 100. For the Matérn 7/2 and the Matérn 5/2, they both have an extreme error at a certain lengthscale. Unlike what we drew here, not all lengthscale values are valid. To check which lengthscale is valid for a certain kernel, check Table 4.1.

However, it is worth noticing that during the iteration of lengthscales, some lengthscale is invalid when they lie in a certain interval. Table 4.1 shows the starting and ending points of invalid lengthscale intervals for different kernels.

Kernel	Start	End
Matérn 5/2	$10^{-0.2} = 0.501187234$	$10^{0} = 1$
Matérn 7/2	0.1	$10^{-0.7} = 0.199526231$
Matérn 9/2	$10^{-1.4} = 0.039810717$	$10^{-1.1} = 0.079432823$
Matérn 11/2	$10^{-1.6} = 0.025118864$	$10^{-1.3} = 0.050118723$
Gaussian	$10^{-2.6} = 0.002511886$	$10^{-0.5} = 0.316227766$

Table 4.1: The starting and ending point of invalid lengthscale interval for different kernels. Note that the start and end point is the last lengthscale that is still valid. Recall that the lengthscale is distributed exponentially with an interval of $10^{0.1}$.

When we take an invalid lengthscale as our input, the Cholesky factorization algorithm complains about the non-Hermitian matrix. Theoretically, however, the matrix will be Hermitian. This error may caused by numerical error.

When choosing Matérn kernels, Matérn 7/2, 9/2 and 11/2 kernel performs roughly equally well. To make sure that the lengthscale could make this algorithm work, it is a good idea to make the length scale in the interval [0.01,0.025].

We find that the Gaussian kernel has the longest invalid lengthscale interval. Thus before running this algorithm, one should also make sure that the lengthscale to be valid.

4.1.3 Monge-Ampère equation in two-dimensional space

Monge-Ampère equation is given by

$$\det(D^2u) = f, \quad x \in (0,1)^2$$

and truth function is defined as $u(x) = \exp(0.5((x_1 - 0.5)^2 + (x_2 - 0.5)^2))$.

Here, they set Gauss-Newton iterations to be 3, and initial guess $u(x) = \frac{1}{2}\|x\|^2$. Matérn kernel with $\nu = 5/2$. The lengthscale ρ of the kernel is set to be 0.3. The result is presented in Figure 4.3.

In the Monge-Ampère equation, they didn't provide code for the Matérn 9/2 and 11/2 kernel. When looping through the lengthscale ρ, we fail to proceed for those kernels because of the same "non-Hermitian matrix" error mentioned in Burgers' equation.

We realized that the kernel and ρ selected in [6] is not optimal. Matérn 7/2 kernel with ρ in the interval [0.3, 2] outperforms Matérn 5/2 kernel at $\rho = 0.3$.

We realize that some kernels may not have a small error regardless of the lengthscale we choose. The Gaussian kernel has an error greater than 0.1 for all valid lengthscales.

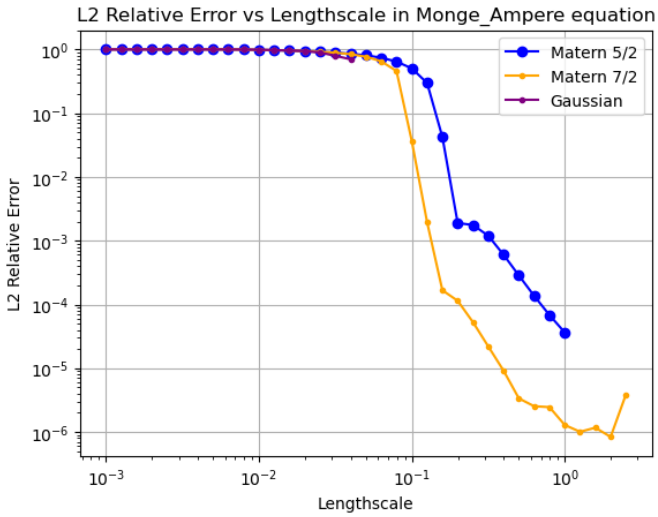


Figure 4.3: Demonstration of lengthscale ρ's relationship with L^2 relative error for different kernels in Monge-Ampère equation in two-dimensional space. Like previous experiments, the lengthscale proceeds $10^{0.1}$ during each iteration, until the stop. In this experiment, however, there is no such interval as Table 4.1 since no values greater than the stopping point are still valid.

4.2 Change truth function in Nonlinear elliptic PDEs

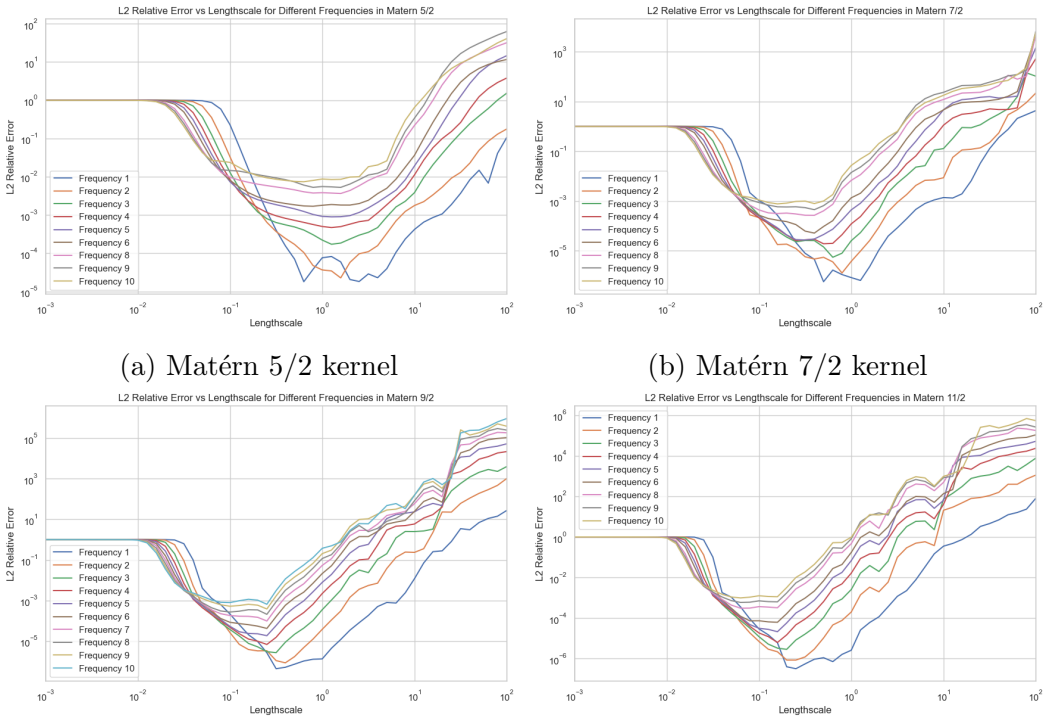
Examining the function $u(x)$ in Nonlinear elliptic PDEs, we find something interesting: the factor $\frac{1}{k^6}$ will make $k = 10$ roughly 10^{-6} of $k = 1$, which makes the high-frequency term, and the term afterward insignificant, since $\sin\left(k\pi x_1\right)\sin\left(k\pi x_2\right) \in [-1, 1]$. In this case, we want to explore how the precision of the algorithm will change as we change the "truth solution", especially when reducing the significance of the low-frequency terms.

4.2.1 Truncate the low-frequency terms

The first way is to remove the low-frequency terms, in other words, change the a in the following equation.

$$u(x) = \sum_{k=a}^{600} \frac{1}{k^6} \sin\left(k\pi x_1\right) \sin\left(k\pi x_2\right)$$

Here we let a ranging from 1 to 10, and we look at all four Matérn kernels.



(a) Matérn 5/2 kernel

(b) Matérn 7/2 kernel

(c) Matérn 9/2 kernel

(d) Matérn 11/2 kernel

Figure 4.4: Demonstration of Lengthscale ρ's relationship with L^2 relative error for different a in Nonlinear elliptic PDEs. In this figure, different subfigures represent different kernels. Each graph reveals the interval of lengthscale between 10^{-3} to 100. The label "Frequency x" means $a = x$.

As we see in Figure 4.4, the L^2 relative error increases for all four Matérn kernels as a increases from 1 to 10. The smallest L^2 relative error for $a = 10$ is 10^{-3}, whereas $a = 1$ is 10^{-6}. Thus truncating the low-frequency terms will cause a larger error for this algorithm.

Another interesting fact is the best lengthscale interval also changed as the frequency changed. For Matérn 11/2 kernel, the best lengthscale interval is $[0.2, 1]$ when $a = 1$ and is $[0.03, 0.15]$ when $a = 10$. For the pattern we observed, the best lengthscale gets smaller when more low-frequency terms are removed.

4.2.2 Change the degree of k

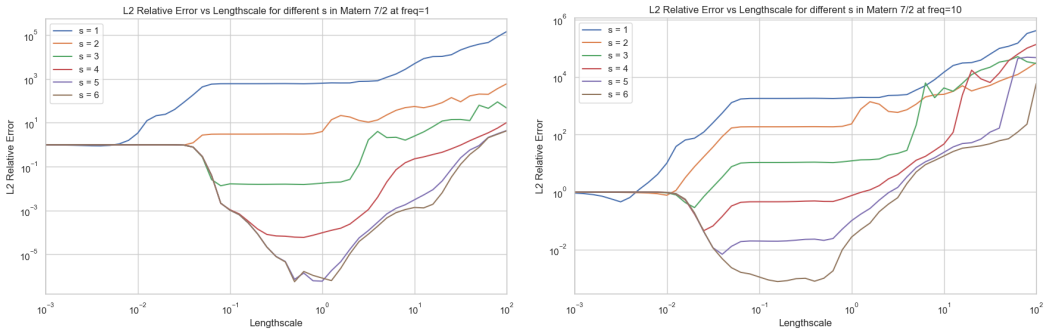
The other way to reduce the significance of low-frequency terms is to reduce the degree of k. In other words, we reduce the parameter s in the following equation.

$$u(x) = \sum_{k=a}^{600} \frac{1}{k^s} \sin\left(k\pi x_1\right) \sin\left(k\pi x_2\right)$$

Here we try $s = 1, 2, 3, 4, 5, 6$, and we use Matérn 7/2 kernel, the result is shown in Figure 4.5.

The result is significant. When we fix the low frequency as 1 ($a = 1$), the algorithm doesn't converge for $s = 1$ and $s = 2$, and $s = 3$ has an error greater than 0.01. Next, we truncate the low-frequency terms. When fixing the lowest frequency as 10 ($a = 10$), all values of s except for $s = 6$, which is the original $u(x)$, fail to have an error less than 0.01. Thus reducing the significance of low frequency will cause larger errors.

From those two experiments, we realized that we need further research on the behavior of the algorithm when operating on higher frequencies.



(a) $a = 1$ (b) $a = 10$

Figure 4.5: Demonstration of Lengthscale ρ's relationship with L^2 relative error for different s in Nonlinear elliptic PDEs. Subfigure 4.5a shows the case when $a = 1$, or no truncation case of nonlinear elliptic PDEs. Subfigure 4.5b shows the case when $a = 10$, or we truncate 9 lowest frequency of the truth function of nonlinear elliptic PDEs.

4.2.3 Representative truth functions

From previous part, we realized that the algorithm fails in some truth functions. Now we want to explore further how to improve the algorithm's performance as we use higher frequency in truth functions. Since we want to make sure the magnitude is roughly the same, we use the truth function:

$$u(x) = \sum_{k=1}^{600} \frac{1}{k^s} \sin\big((k+a)\pi x_1\big) \sin\big((k+a)\pi x_2\big)$$

Since the "Matern 7/2" kernel is fast and is similar to the behavior of the other four kernels, we now only use that to continue our experiment.

We pick five representative sets of s and a:

1. $s = 6$, $a = 20$
2. $s = 6$, $a = 100$
3. $s = 3$, $a = 50$
4. $s = 3$, $a = 0$
5. $s = 1$, $a = 0$

Under the default setting, their L_2 relative error are presented as Figure 4.6.

These five truth equations will be used in later parts to show the improvement of algorithms.

We have two main objectives for improving those five truth equations. The first one is to improve its accuracy, for cases like improving the error for Case 1($s = 6$, $a = 20$), and Case 4 ($s = 3$, $a = 50$). We aim to narrow the error down or increase the "good" interval of the Lengthscale. The other is convergence. For Case 2($s = 6$, $a = 100$), Case 3($s = 3$, $a = 50$), and especially Case 5 ($s = 1$, $a = 0$), their error can be easily greater than itself. Therefore, we want to see whether they can converge by changing hyperparameters.

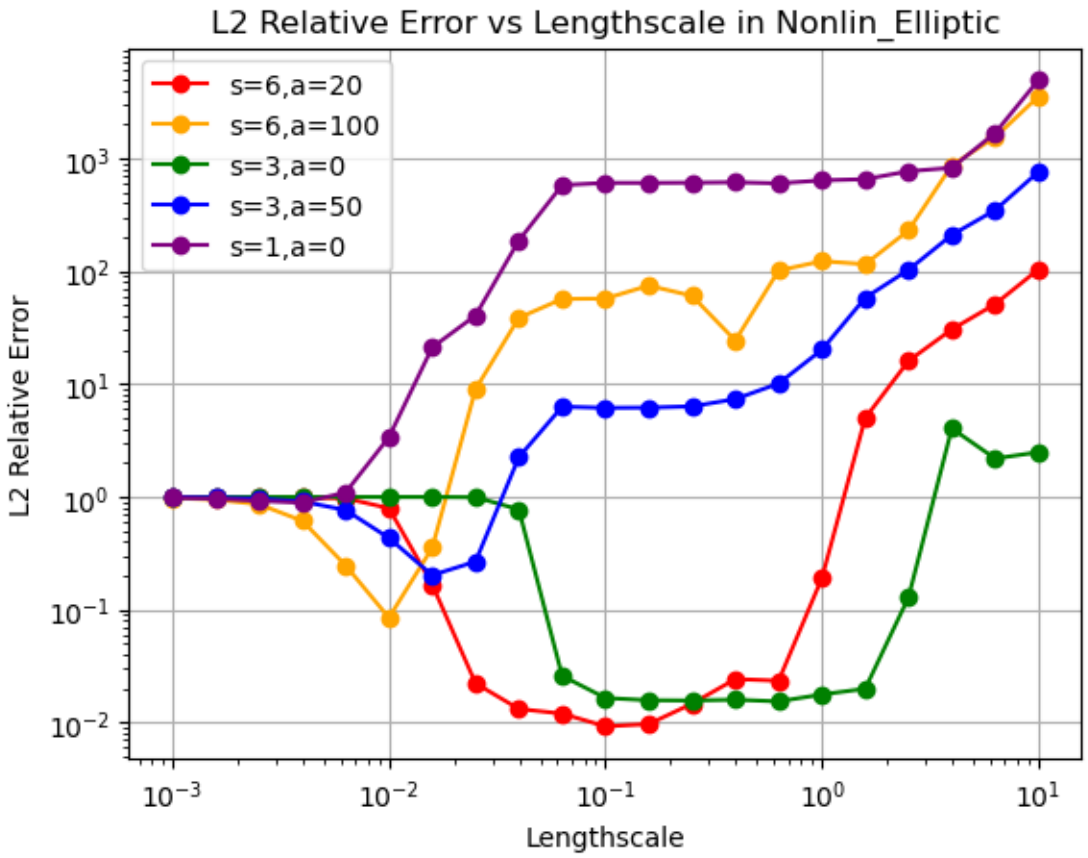


Figure 4.6: The performance of five truth equations in non-linear elliptic PDE.

4.3 Improvements

4.3.1 Increase Gauss-Newton steps

In [6], Gauss-Newton (GN) steps is always less than 10. The number of steps may be insufficient when the algorithm didn't perform well. Thus we want to see whether increase GN steps can improve the algorithm performance.

4.3.1.1 Nonlinear elliptic PDEs

The default number of GN steps is 3 for Nonlinear elliptic PDEs. We tested steps to be 10 and 30, to see if this could change. Our result is presented in Figure 4.7.

The default number of GN steps is efficient for all five truth equations. In Figure

4.7a and 4.7c, there are no clear accuracy improvements in the optimal lengthscale when we increase the GN step from 3 to 30. In Figure 4.7b, 4.7d, and 4.7e, the GN step also didn't solve the convergence problem caused by high frequency.

Overall, an increase in GN steps only have minor help in improving performance.

4.3.1.2 Monge-Ampère equation

The default number of GN steps is 3 for Monge-Ampère equation. We tested steps to be 10 and 30, to see if this could change. In Figure 4.8, we realized that this change of GN steps have minor influence on performance.

Since an increase in GN steps did minor help in improving performance, for later part, we fixed the number of GN step as default, which is 3.

4.3.2 Increase small (Algorithm 2) KNN value

K-nearest-neighbor (KNN) is the algorithm makes predictions based on the majority class or average value of the K nearest data points in the feature space. In this algorithm, KNN value is another parameter that influence the precision, increasing it will increase the precision of the algorithm. It is not presented Algorithm 1, but in their code. We aim to increase the value of KNN to see if this will help increase the accuracy.

The original code uses the same KNN value for the Sparse Cholesky factorization outside and Algorithm 2 inside. We decided to separate them, to see whether the KNN value in Algorithm 1 and Algorithm 2 will change the result.

To make a good notation, we define all parameters that belongs to Sparse Cholesky factorization, or Algorithm 1, as "big", and those belongs to Algorithm 2 as "small". Here we represent the k-nearest neighbors value in Algorithm 1 as the "big" KNN value, while the KNN value of Algorithm 2 as the "small" KNN value.

We first explore the effectiveness of increasing the small KNN value.

In Nonlinear elliptic PDEs, the default small KNN value is 3 for all five truth equations. We changed the small KNN to be 10 and 30, to see if this could change. Our result is presented in Figure 4.9. In this case, we will keep the default value 3 for small KNN.

For all five experiments, an increase in small (Algorithm 2) KNN value did no help either improve its accuracy or on convergence. So I fixed the small KNN value to be 3.

4.3.3 Increase big KNN value

In Nonlinear elliptic PDEs, the default big KNN value is 3 for all five truth equations. We changed the big KNN to be 10 and 30, to see if this could change. Our result is presented in Figure 4.10.

For all five experiments, an increase in big KNN value did improve the accuracy in some cases, especially $s = 6, a = 20$, where an increase of big KNN from 3 to 10 reduce the L2 relative error from 10^{-2} to 10^{-3}.

However, it didn't fix the convergence problem in case $s = 3, a = 50$ and $s = 1, a = 0$.

Since the improvement of big KNN from 10 to 30 is minor, I set big KNN value to 10 in later part of my experiment.

4.3.4 Increase small ρ value

Next we want to change the ρ value. In [6], they claimed that they observe a consistent decay of the KL errors when the sparsity parameter ρ increases. Thus I want to see how much this parameter can improve the performance.

Similar to KNN value, we need to discuss about ρ value separately. The ρ value in Algorithm 1 is called "big" ρ, and for Algorithm 2 is "small" ρ.

In Nonlinear elliptic PDEs, the default small ρ value is 3 for all five truth equations. We changed the small ρ value to 10 and 30, to see if this could solve accuracy and

convergence problems. Our result is presented in Figure 4.11.

The result is surprising, the change in small ρ didn't make any change to the performance of the algorithm. Their result overlap with each others. Thus we fix it as 3 for later part of experiments.

4.3.5 Increase big ρ value

Next we want to change the big ρ value for those Nonlinear elliptic PDEs.

The default big ρ value is 3 for all five truth equations. We changed the small ρ_{big} value to 10 and 30, to see if this could solve accuracy and convergence problems. Our result is presented in Figure 4.12.

When changing the big ρ value, the performance of the algorithm improved a little, but still fail to solve the convergence problem in case $s = 1, a = 0$.

4.4 Summary

4.4.1 Kernel and Lengthscale

Our experiment demonstrates that the selection of kernel and its corresponding lengthscale parameter, denoted as ρ, influences the performance of the Sparse Cholesky factorization algorithm in [6].

We found that the optimal lengthscale intervals for different kernels vary, and the Gaussian kernel exhibited instability, while the Matérn 7/2 and 9/2 kernels performed optimally under certain conditions. Specifically, for equations like Burgers' equation, certain lengthscales may lead to a non-Hermitian matrix during the Cholesky factorization algorithm, indicating the need for further research to enhance the stability of this algorithm.

Moreover, our experiments suggest that neither too small (e.g., 10^{-4}) nor too large (e.g., 100) values are suitable for the algorithm. Generally, small lengthscales result

in a 100% relative error, while large lengthscales fail to converge. Therefore, if the algorithm converges, the appropriate range for the lengthscale typically lies in the middle.

Furthermore, for different kernels or the same kernel applied to different testing functions, the optimal range of lengthscales varies.

4.4.2 Truth function of Nonlinear Elliptic PDEs

When running nonlinear elliptic PDEs using the Sparse Cholesky factorization algorithm, truncating or reducing the significance of the low-frequency terms in the truth function leads to a larger error for this algorithm. In some cases, convergence is not achieved. Moreover, the optimal lengthscale interval changes when the low-frequency terms are removed.

In particular, when the problem still converges, we observed a pattern where the best lengthscale range decreases as higher frequency terms become more significant, both by truncating lower frequency terms and by decreasing the degree of k.

4.4.3 GN Steps and other parameters

For this part of experiment, we pick 5 truth function for

$$u(x) = \sum_{k=1}^{600} \frac{1}{k^s} \sin\big((k+a)\pi x_1\big) \sin\big((k+a)\pi x_2\big)$$

which we sets the parameter s and a to be:

1. $s = 6$, $a = 20$
2. $s = 6$, $a = 100$
3. $s = 3$, $a = 50$
4. $s = 3$, $a = 0$
5. $s = 1$, $a = 0$

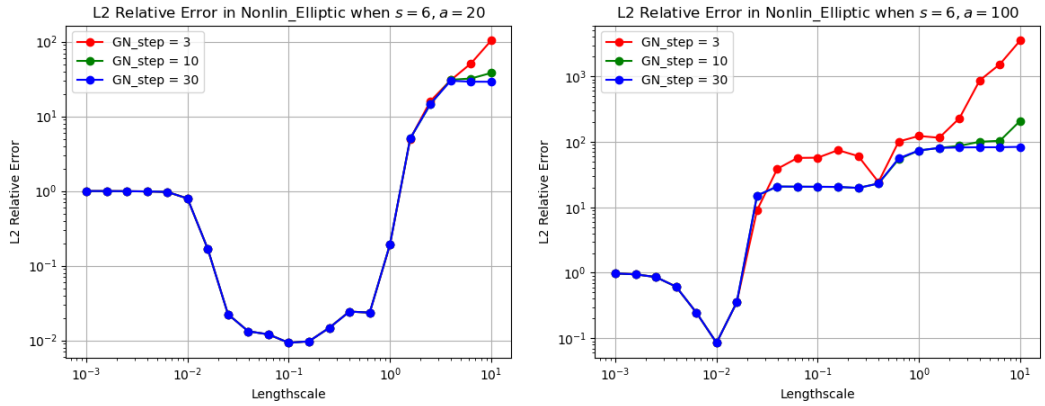
We adjusted five different parameters: the Gauss-Newton iteration steps (GN steps), large KNN (KNN of Algorithm 1), small KNN (KNN of Algorithm 2), large ρ (ρ of Algorithm 1), and small ρ (ρ of Algorithm 2).

From our experiments, we have not yet identified a hyper-parameter that resolves the convergence problem for cases like $s = 1$ and $a = 0$. However, we observed improvements in accuracy when increasing the large KNN and large ρ.

Despite the absence of a theoretical guarantee for the sparse Cholesky factorization in Algorithm 2, the GN steps remain adequate even when set to 3. However, we did not observe an increase in accuracy by increasing the GN steps.

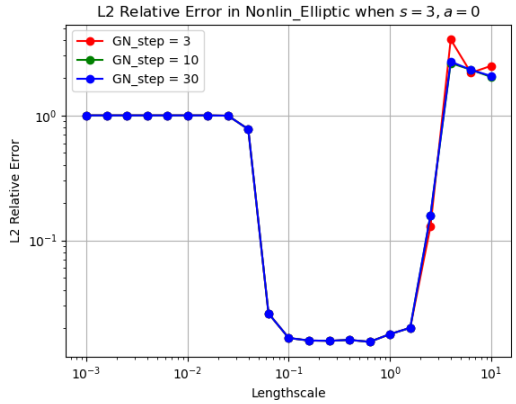
Similarly, increasing the small KNN and small ρ did not lead to improved accuracy. This suggests that Algorithm 3 may not be the cause of non-convergence in some cases.

The default settings of the program only show improvement when the hyper-parameters of the outer Sparse Cholesky factorization, which is Algorithm 1, are improved.

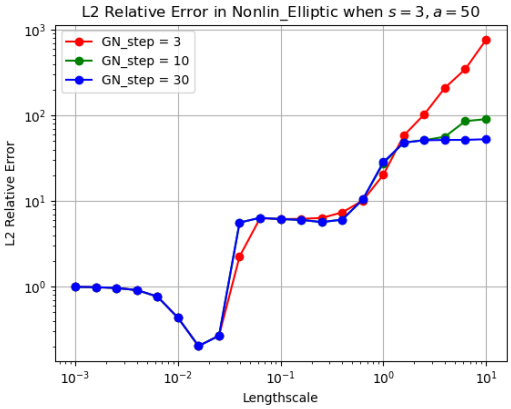


(a) $s = 6$, $a = 20$

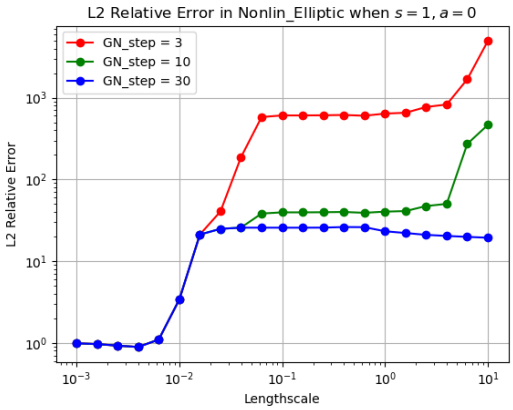
(b) $s = 6$, $a = 100$



(c) $s = 3$, $a = 0$



(d) $s = 3$, $a = 50$



(e) $s = 1$, $a = 0$

Figure 4.7: Demonstration of the relationship between the number of GN steps and L^2 relative error for different truth solutions in Nonlinear elliptic PDEs. The truth solutions is $u(x) = \sum_{k=1}^{600} \frac{1}{k^s} \sin\big((k+a)\pi x_1\big) \sin\big((k+a)\pi x_2\big)$, and is different in 5 subgraphs by choosing different s and a.

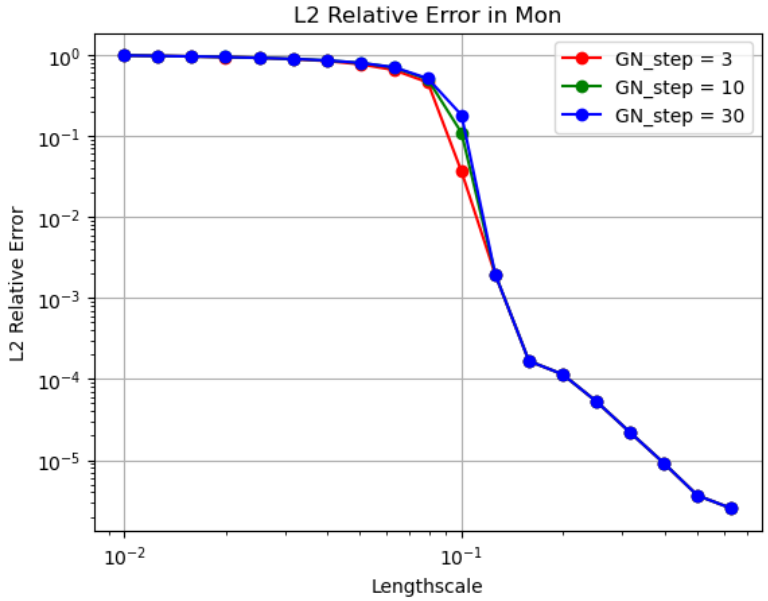
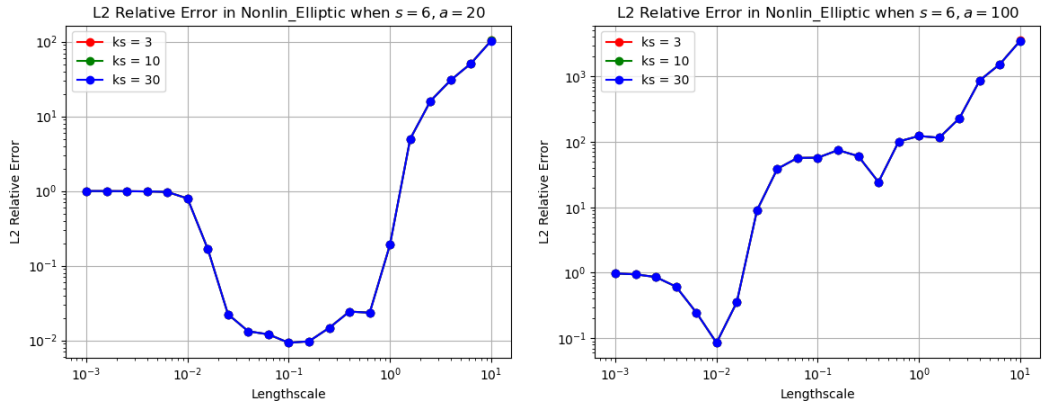
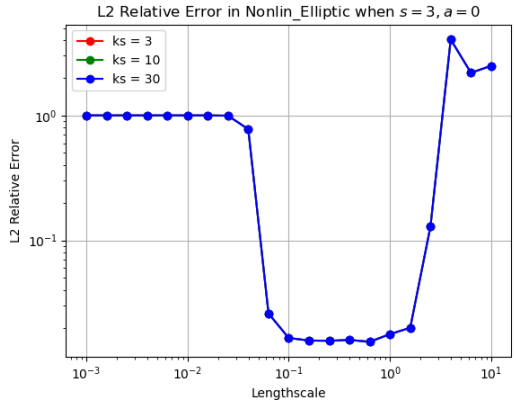


Figure 4.8: Demonstration of the relationship between the number of GN steps and L^2 relative error for Monge-Ampère equation, with truth function $u(x) = \exp(0.5((x_1 - 0.5)^2 + (x_2 - 0.5)^2))$.

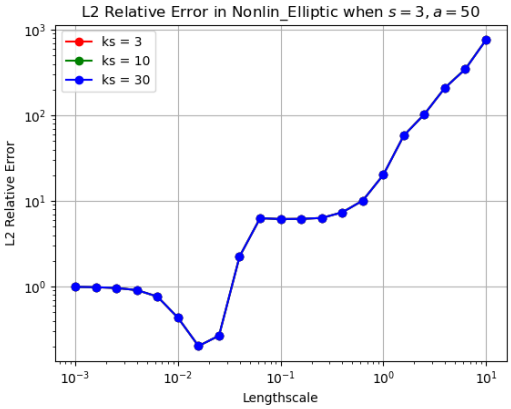


(a) $s = 6$, $a = 20$

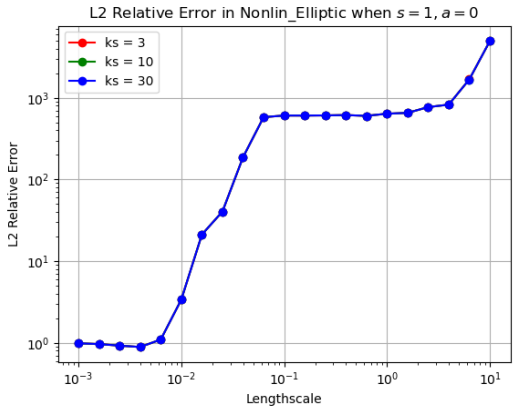
(b) $s = 6$, $a = 100$



(c) $s = 3$, $a = 0$

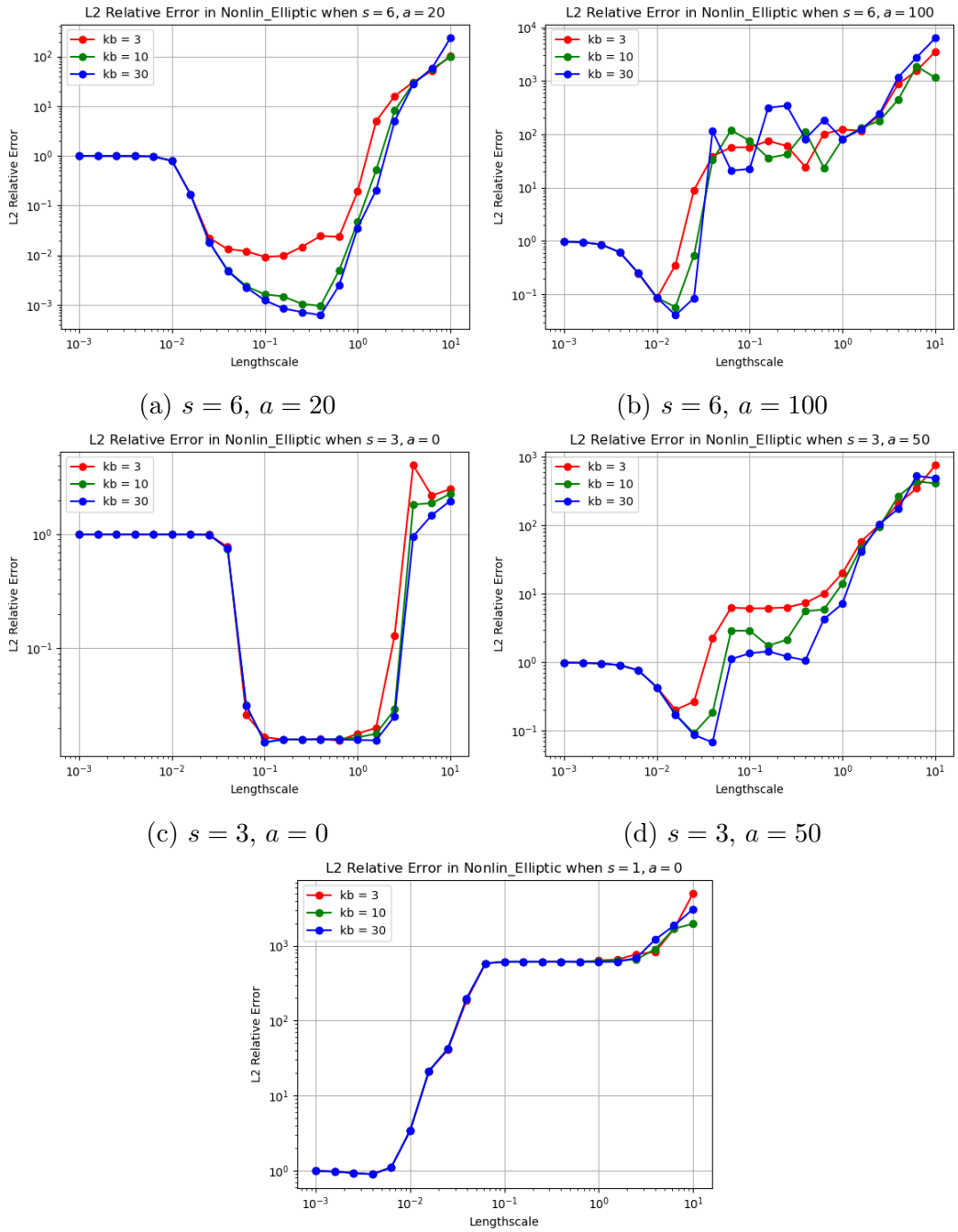


(d) $s = 3$, $a = 50$



(e) $s = 1$, $a = 0$

Figure 4.9: Demonstration of the relationship between the value of small KNN (labeled as ks) and L^2 relative error for different truth solutions in Nonlinear elliptic PDEs. The truth solutions is $u(x) = \sum_{k=1}^{600} \frac{1}{k^s} \sin\left((k+a)\pi x_1\right) \sin\left((k+a)\pi x_2\right)$, and is different in 5 subgraphs by choosing different s and a.



(a) $s = 6$, $a = 20$

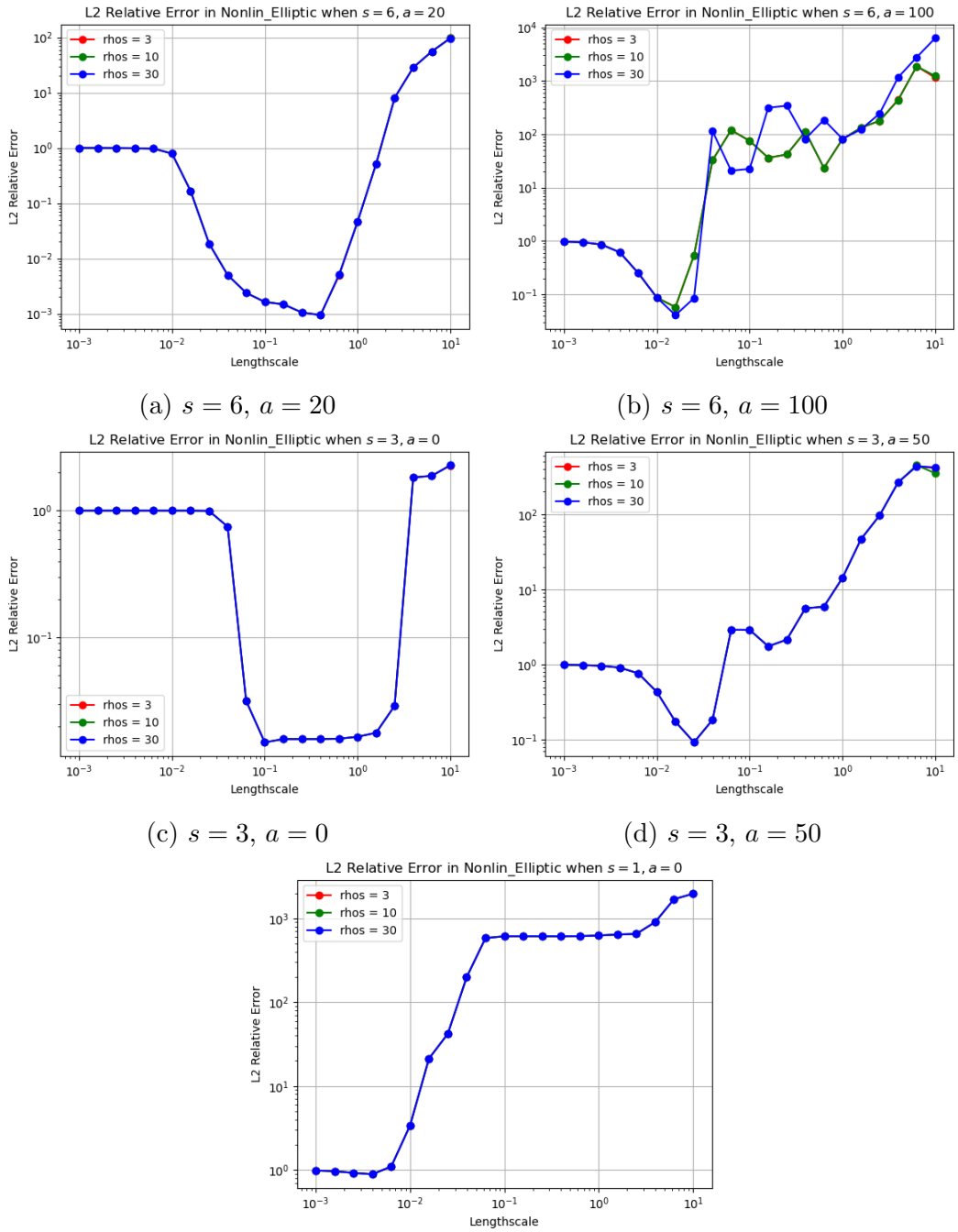
(b) $s = 6$, $a = 100$

(c) $s = 3$, $a = 0$

(d) $s = 3$, $a = 50$

(e) $s = 1$, $a = 0$

Figure 4.10: Demonstration of the relationship between the value of big KNN (labeled as kb) and L^2 relative error for different truth solutions in Nonlinear elliptic PDEs. The truth solutions is $u(x) = \sum_{k=1}^{600} \frac{1}{k^s} \sin\left((k+a)\pi x_1\right) \sin\left((k+a)\pi x_2\right)$, and is different in 5 subgraphs by choosing different s and a.



(a) $s = 6$, $a = 20$

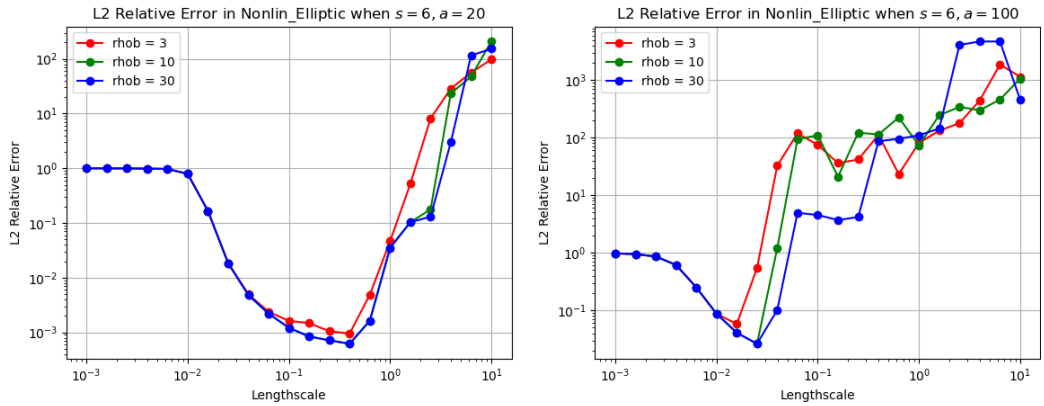
(b) $s = 6$, $a = 100$

(c) $s = 3$, $a = 0$

(d) $s = 3$, $a = 50$

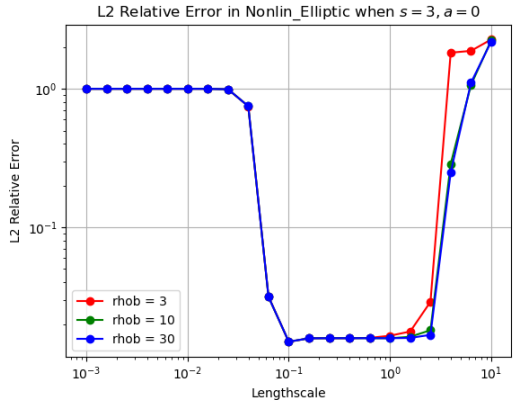
(e) $s = 1$, $a = 0$

Figure 4.11: Demonstration of the relationship between the value of small ρ (labeled as rhos) and L^2 relative error for different truth solutions in Nonlinear elliptic PDEs. The truth solutions is $u(x) = \sum_{k=1}^{600} \frac{1}{k^s} \sin\left((k+a)\pi x_1\right) \sin\left((k+a)\pi x_2\right)$, and is different in 5 subgraphs by choosing different s and a.

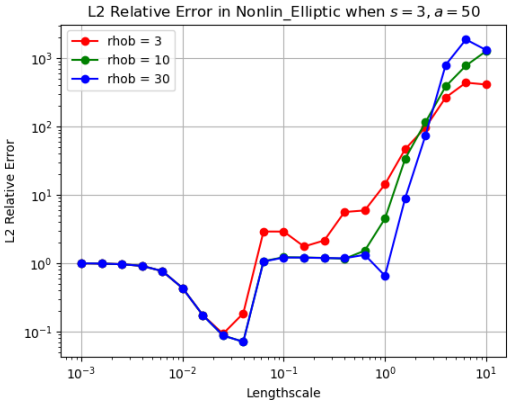


(a) $s = 6$, $a = 20$

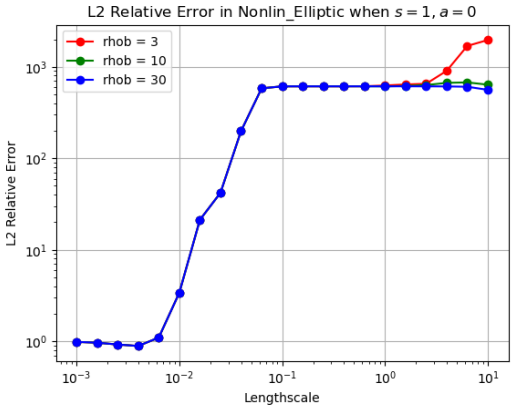
(b) $s = 6$, $a = 100$



(c) $s = 3$, $a = 0$



(d) $s = 3$, $a = 50$



(e) $s = 1$, $a = 0$

Figure 4.12: Demonstration of the relationship between the value of big ρ (labeled as rhob) and L^2 relative error for different truth solutions in Nonlinear elliptic PDEs. The truth solutions is $u(x) = \sum_{k=1}^{600} \frac{1}{k^s} \sin\left((k+a)\pi x_1\right) \sin\left((k+a)\pi x_2\right)$, and is different in 5 subgraphs by choosing different s and a.

Chapter 5

Conclusion

This study has delved into the framework for addressing nonlinear PDEs using Gaussian Process [6]. It has been observed that the choice of kernel, PDE type, and selection of truth function and parameters can significantly influence the algorithm's performance.

Regarding kernel selections, it has been noted that selecting a suitable lengthscale for different kernels is crucial. Either too large or too small will be inappropriate. Generally, smoother Matérn kernels exhibit better performance, but further research is warranted to validate this observation, especially since the Gaussian kernel, despite being the smoothest, does not yield the best performance.

In the context of nonlinear elliptic PDEs, it has been observed that the algorithm's performance deteriorates when high-frequency terms of the truth function u are weighted more heavily.

Efforts to enhance the algorithm involved adjustments to the number of Gauss-Newton iterations, KNN value, and lengthscale ρ for Algorithm 1 and 2. Interestingly, changes to the Gauss-Newton iteration and KNN value, as well as the lengthscale ρ in Algorithm 2, did not yield improvements in algorithm performance. Conversely, adjustments made to Algorithm 1 demonstrated some enhancements. Consequently, further exploration is warranted to devise strategies to enhance Algorithm 1 and

bolster accuracy. Potential avenues for future research include parameter adjustments, refining the algorithm's structure, or devising novel algorithms derived from it.

Bibliography

[1] Alain Berlinet and Christine Thomas-Agnan. *Reproducing kernel Hilbert spaces in probability and statistics*. Springer Science & Business Media, 2011.

[2] Kaushik Bhattacharya, Bamdad Hosseini, Nikola B Kovachki, and Andrew M Stuart. Model reduction and neural networks for parametric pdes. *The SMAI journal of computational mathematics*, 7:121–157, 2021.

[3] Susanne Brenner, Thirupathi Gudi, Michael Neilan, and Li-yeng Sung. c^0 penalty methods for the fully nonlinear monge-ampère equation. *Mathematics of Computation*, 80(276):1979–1995, 2011.

[4] Yifan Chen, Bamdad Hosseini, Houman Owhadi, and Andrew M Stuart. Solving and learning nonlinear pdes with gaussian processes. *Journal of Computational Physics*, 447:110668, 2021.

[5] Yifan Chen, Houman Owhadi, and Andrew Stuart. Consistency of empirical bayes and kernel flow for hierarchical parameter estimation. *Mathematics of Computation*, 90(332):2527–2578, 2021.

[6] Yifan Chen, Houman Owhadi, and Florian Schäfer. Sparse cholesky factorization for solving nonlinear pdes via gaussian processes. *arXiv preprint arXiv:2304.01294*, 2023.

[7] Matthieu Darcy, Boumediene Hamzi, Giulia Livieri, Houman Owhadi, and Pey-

man Tavallali. One-shot learning of stochastic differential equations with data adapted kernels. *Physica D: Nonlinear Phenomena*, 444:133583, 2023.

[8] Arka Daw, Jie Bu, Sifan Wang, Paris Perdikaris, and Anuj Karpatne. Rethinking the importance of sampling in physics-informed neural networks. *arXiv preprint arXiv:2207.02338*, 2022.

[9] Tamara G Grossmann, Urszula Julia Komorowska, Jonas Latz, and Carola-Bibiane Schönlieb. Can physics-informed neural networks beat the finite element method? *arXiv preprint arXiv:2302.04107*, 2023.

[10] Joseph Guinness. Permutation and grouping methods for sharpening gaussian process approximations. *Technometrics*, 60(4):415–429, 2018.

[11] Yipeng Huang, Ning Guo, Mingoo Seok, Yannis Tsividis, Kyle Mandli, and Simha Sethumadhavan. Hybrid analog-digital solution of nonlinear partial differential equations. In *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*, pages 665–678, 2017.

[12] Victor Ivrii and University of Toronto Department of Mathematics. Partial differential equations. Technical report, 2022. URL `https://www.math.utoronto.ca/ivrii/PDE-textbook/PDE-textbook.pdf`.

[13] Vishal jagota, Amanpreet Sethi, and Dr-Khushmeet Kumar. Finite element method: An overview. *Walailak Journal of Science & Technology*, 10:1–8, 01 2013. doi: 10.2004/wjst.v10i1.499.

[14] Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.

[15] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations. *arXiv preprint arXiv:2010.08895*, 2020.

[16] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2021.

[17] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3): 218–229, March 2021. ISSN 2522-5839. doi: 10.1038/s42256-021-00302-5. URL `http://dx.doi.org/10.1038/s42256-021-00302-5`.

[18] Houman Owhadi and Clint Scovel. *Operator-Adapted Wavelets, Fast Solvers, and Numerical Homogenization: From a Game Theoretic Approach to Numerical Approximation and Algorithm Design*, volume 35. Cambridge University Press, 2019.

[19] Houman Owhadi and Gene Ryan Yoo. Kernel flows: From learning kernels from data into the abyss. *Journal of Computational Physics*, 389:22–47, 2019.

[20] Florian Schäfer, Matthias Katzfuss, and Houman Owhadi. Sparse cholesky factorization by kullback–leibler minimization. *SIAM Journal on scientific computing*, 43(3):A2019–A2046, 2021.

[21] L Ridgway Scott. Introduction to automated modeling using fenics. 2017.

[22] Walter A Strauss. *Partial differential equations: An introduction*. John Wiley & Sons, 2007.

[23] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

[24] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why pinns fail to train: A neural tangent kernel perspective. *Journal of Computational Physics*, 449: 110768, 2022.

[25] Holger Wendland. *Scattered data approximation*, volume 17. Cambridge university press, 2004.

[26] Andrew Gordon Wilson, Zhiting Hu, Ruslan Salakhutdinov, and Eric P Xing. Deep kernel learning. In *Artificial intelligence and statistics*, pages 370–378. PMLR, 2016.

[27] Qi Zeng, Yash Kothari, Spencer H Bryngelson, and Florian Tobias Schaefer. Competitive physics informed networks. In *The Eleventh International Conference on Learning Representations*, 2023. URL `https://openreview.net/forum?id=z9SIj-IM7tn`.