Numerical Gaussian processes for solving linear ordinary differential equations

Xinyu Zhong
Queens' College

Word count: 4241

1 Introduction

The numerical solution of partial differential equations (PDEs) is a cornerstone of computational science and engineering, with applications ranging from fluid dynamics and heat transfer to quantum mechanics and financial modeling. Traditional approaches, such as finite difference, finite element, and spectral methods, have been the workhorses of scientific computing for decades. However, these methods often lack the ability to quantify uncertainty in their solutions, particularly when initial or boundary conditions are imprecisely known.

In recent years, there has been growing interest in leveraging machine learning techniques to address challenges in scientific computing. Among these, the numerical Gaussian Processes (GPs) have emerged as a promising approach for solving differential equations. GPs offer several attractive features:

- They provide probabilistic solutions, naturally quantifying uncertainty in their predictions.
- They can incorporate prior knowledge about the system through the selection of appropriate kernel functions.
- They offer a non-parametric approach, allowing for flexible modeling of complex phenomena.

This project aims to explore the application of Gaussian Processes for solving linear ordinary differential equations, with a specific focus on the Burger's equation and the wave equation. Our primary objectives are:

1. To implement GP-based solvers for these equations, integrating them with finite difference time-stepping methods.
2. To compare the performance of GP solvers with traditional finite difference methods in terms of accuracy and uncertainty quantification.
3. To investigate the impact of kernel selection on the quality of solutions.
4. To analyze the propagation of uncertainty through time in GP solutions.

We build upon the work of Raissi et al.(2017)[9] and extending their approaches and providing a detailed analysis of the strengths and limitations of GP-based PDE solvers. Our implementation utilizes the GPJax library, a Gaussian process framework built on JAX [7].

This report is structured as follows: Section 2 provides the theoretical background on Gaussian Processes and their application to differential equations. Section 3 details our implementation, including the choice of kernel functions and the integration with finite difference methods. Section 4 presents our results and analysis, comparing GP solutions and examining the impact of various parameters. Finally, we conclude with a discussion of the implications of our findings and potential directions for future research in Section 5.

Through this work, we aim to understand the growing knowledge on machine learning approaches to scientific computing, highlighting both the potential and the challenges of using numerical Gaussian Processes for solving differential equations.

2 Background theory on Gaussian processes

2.1 Gaussian process

Gaussian Process is a non-parametric method that can be used for regression, classification, and optimization problems. It is a collection of random variables, any finite number of which have a joint Gaussian distribution. It is completely specified by its mean function and covariance function[10].

The advantage of Gaussian Process includes a few key points [10]:

- It is a non-parametric method, which means that it does not assume a specific functional form for the data.
- It provides a measure of uncertainty for the predictions.
- It has a flexible kernel function that can be chosen based on the physical properties of the data.

Although it comes with a few drawbacks[10]:

- It is computationally expensive for large datasets.
- It is not suitable for high-dimensional data.

Gaussian process can be used solve regression problems, classification problems, and optimization problems[10]. In this project, we will on solving linear differential equation with Gaussian process regression and finite time step method.

2.2 Gaussian process regression

Gaussian process can be used to solve regression problems. A Gaussian process is completely specified by its mean function and covariance function. i.e.

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')) \tag{1}$$

where $m(x)$ is the mean function and $k(x, x')$ is the covariance function. The mean function is usually set to zero, and the covariance function is usually chosen to be a squared exponential kernel, or other kernels that are suitable for the data.

Suppose we have measured value at n points $\mathbf{X} = \{x_1, x_2, \ldots, x_n\}$, and the corresponding values $\mathbf{f} = \{f(x_1), f(x_2), \ldots, f(x_n)\}$, and we are interested in predicting the value at a new point x_*.

The joint distribution of the observed values and the predicted values is given by:

$$\begin{bmatrix} \mathbf{f}_* \\ \mathbf{f} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \mu_* \\ \mu \end{bmatrix}, \begin{bmatrix} \mathbf{k}_{**} & \mathbf{k}_*^{\mathbf{T}} \\ \mathbf{k}_* & \mathbf{K} \end{bmatrix} \right) \tag{2}$$

where $\mathbf{f}_*$ is the predicted value at x_*, μ_* is the mean of the predicted value. And the block matrices are defined as:

$$\begin{aligned} \mathbf{k}_{**} &= k(x_*, x_*) \\ \mathbf{k}_* &= k(x_*, \mathbf{X}) \\ \mathbf{K} &= k(\mathbf{X}, \mathbf{X}) \end{aligned} \tag{3}$$

The mean and variance of the predicted value is given by:

$$\mu_* = \mathbf{k}_*^{\mathbf{T}} \mathbf{K}^{-1} \mathbf{f} \tag{4}$$

$$\sigma_*^{\mathbf{2}} = \mathbf{k}_{**} - \mathbf{k}_*^{\mathbf{T}} \mathbf{K}^{-1} \mathbf{k}_* \tag{5}$$

The equation (4) is the mean of the predicted value. The equation (5) is the variance of the predicted value. It provides a measure of uncertainty for the prediction, which is a key advantage of Gaussian processes compared to the finite difference method and any other curve fiting methods.

2.3 Kernel function

The choice of the kernel function is important in Gaussian processes. A well-chosen kernel function can capture the physical properties of the data, and provide a good prediction. It should also be easy to implement, i.e. the derivatives of the kernel function can be computed analytically.

The squared exponential kernel is a popular choice for the kernel function, and it is defined as:

$$k(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2l^2} \right) \tag{6}$$

The squared exponential kernel is smooth and infinitely differentiable, which makes it a good choice for the kernel function for describing continuous data.

The squared exponential kernel is also known as the radial basis function (RBF) kernel. It is a stationary kernel, which means that the covariance between two points is only dependent on the distance between the two points, i.e.

$$k(x, x') = k(x - x') \tag{7}$$

This property may not be suitable for some data, such as the Burger's equation, where the kernel function should be non-stationary. The kernel function chosen for the Burger's equation will be discussed in the next section.

2.4 Numerical Gaussian processes

An Numerical Gaussian Process (NGP) is an extension of the GP framework specifically designed to solve differential equations. It combines the probabilistic nature of GPs with numerical methods (like finite difference) to approximate solutions to differential equations.

2.5 Linear transformation of Gaussian process

One important property of Gaussian process is that it can be transformed by a linear transformation. Consider $\mathcal{L}$ to be a linear transformation applied to the Gaussian process $f(x)$, i.e.

$$g(x) = \mathcal{L} f(x) \tag{8}$$

The Transformed Gaussian Process is given by: The mean value of the transformed Gaussian process is given by:

$$\mu_g(x) = \mathcal{L} \mu_f(x) \tag{9}$$

The covariance between $g(x)$ and $g(x')$ is given by:

$$\begin{aligned}\operatorname{cov}(g(x), g(x')) &= \mathcal{L}\operatorname{cov}(f(x), f(x'))\mathcal{L}^T \\ &= \mathcal{L}k(x, x')\mathcal{L}^T\end{aligned} \tag{10}$$

and the covariance between $g(x)$ and $f(x')$ is given by:

$$\begin{aligned}\operatorname{cov}(g(x), f(x')) &= \mathcal{L}\operatorname{cov}(f(x), f(x')) \\ &= \mathcal{L}k(x, x')\end{aligned} \tag{11}$$

Therefore, the linearly transformed Gaussian process can be easily computed by applying a linear transformation to the mean function and covariance function of the original Gaussian process.

In this project we will use this property of Gaussian process extensively to construct the solutions of time dependent partial differential equations.

2.6 Application of Linear transformation of Gaussian process

The linear transformation of Gaussian process can be used to solve partial differential equations. Consider the time dependent partial differential equation:

$$\frac{\partial u}{\partial t} = \mathcal{L}u \tag{12}$$

By applying a finite difference method, the time derivative can be approximated by:

$$\frac{\partial u}{\partial t} \approx \frac{u(t + \Delta t) - u(t)}{\Delta t} \tag{13}$$

where Δt is the time step.
Using backward Euler for example, the solution at the next time step is given by:

$$u^n = u^{n-1} + \Delta t \mathcal{L}u^n \tag{14}$$

where the superscript n denotes the time step. With rearranging the terms, we have:

$$u^{n-1} = (I - \Delta t\mathcal{L})u^n \tag{15}$$

where I is the identity matrix.
Equation (15) shows that the solution at the next step u^n and current step u^{n-1} are related by a linear transformation. Hence, we can easily construct a Gaussian Process to predict u^n with given u^{n-1}.

2.7 Posterior prediction

The posterior prediction of the Gaussian process can be used to predict the value at a new point, given the training data. The posterior prediction is given by:

$$u^n(x_*^n) \left| \begin{bmatrix} u_b^n \\ u^{n-1} \end{bmatrix} \sim \mathcal{N}\left(q^T K^{-1} \begin{bmatrix} u_b^n \\ u^{n-1} \end{bmatrix}, k_{u,u}^{n,n}(x_*^n, x_*^n) - q^T K^{-1} q \right), \right. \tag{16}$$

where

$$q^T := \begin{bmatrix} k_{u,b}^{n,n}(x_*^n, x_b^n) & k_{u,u}^{n,n-1}(x_*^n, x^{n-1}) \end{bmatrix}. \tag{17}$$

where u_b^n is the boundary condition at time step n, u^{n-1} is the solution at the previous time step, $k_{u,u}^{n,n}$ is the covariance between the boundary condition and the solution at time step n, $k_{u,u}^{n,n-1}$ is the covariance between the solution at time step n and the solution at time step $n-1$, and $k_{u,u}^{n,n}$ is the covariance between the solution at time step n and the solution at time step n.

2.8 Propagating the Error

One key advantage of using Gaussian process to solve partial differential equations is that the uncertainty of the prediction can be propagated through time. Since the in each step, the training data, $\{x^{n-1}, u^{n-1}\}$ is artificially generated, which comes with its own uncertainty. i.e.[9]

$$u^{n-1} \sim \mathcal{N}(\mu^{n-1}, \Sigma^{n-1,n-1}) \tag{18}$$

where μ^{n-1} is the mean of the prediction and $\Sigma^{n-1,n-1}$ is the variance of the prediction.

The prediction at the next time step is given by:

$$u^n \sim \mathcal{N}(\mu^n, \Sigma^{n,n}) \tag{19}$$

where the mean and variance of the prediction is given by:

$$\mu^n = q^T K^{-1} \begin{bmatrix} u_b^n \\ \mu^{n-1} \end{bmatrix} \tag{20}$$

and

$$\Sigma^{n,n}(x_*^n, x_*^n) = k_u^{n,n}(x_*^n, x_*^n) - q^T K^{-1} q + q^T K^{-1} \begin{bmatrix} 0 & 0 \\ 0 & \Sigma^{n-1,n-1} \end{bmatrix} K^{-1} q \tag{21}$$

In the actual implementation in `GPJax`, the `predict()` function is replaced by a customised function that takes one additional argument, the covariance matrix of the training data: `predict_with_prev_covariance()`. This improvisation is one of the key implementation in this project that modifies the original `GPJax` library in order to propagate the uncertainty through each training step.

3 Implementation

The implementation of the numerical Gaussian Process is done using the GPJax library, which is a Gaussian process library built on JAX. The library provides a simple interface to train and predict using Gaussian processes. Apart from the standard kernel functions, the library also provides a way to implement custom kernel functions[7]. Two equations are implemented using numerical Gaussian processes: Burger's equation and the wave equation.

3.1 Work flow of Numerical Gaussian Process

The implementation of numerical Gaussian Process on a time dependent differential equation is as follows:

1. The initial data $\{\mathbf{x}^0, \mathbf{u}^0\}$ and boundary data$\{\mathbf{x}_b^1, \mathbf{u}_b^1\}$ are collected or generated.[1]
2. A model is trained using the initial data and boundary data, to find the hyperparameters of the kernel functions.
3. The model is used to predict the value of the function at the next time step $\mathbf{u}^1$.
4. The predicted value $\{\mathbf{x}^1, \mathbf{u}^1\}$ and the boundary data $\{\mathbf{x}_b^2, \mathbf{u}_b^2\}$ are used as the initial data for the next time step.
5. The process is repeated until the desired time step is reached.

The parameters of the kernel function are updated in every time step to improve the prediction accuracy. To speed up the training process, each training uses the hyperparameters of the previous time step as the initial guess for the hyperparameters of the current time step. The hyperparameters are updated using Adam optimizer with a learning rate of 0.01.

3.2 Kernel Selection

The kernel function is an important component of the Gaussian process, as it captures the physical properties of the data. The choice of the kernel function in the projects involves the following criteria:

1. **Ease of implementation:** The kernel function should be easy to implement and understand. The implementation of the kernel function should be straightforward and should not require complex mathematical operations. The analytic form of the derivative of the kernel function should also be easy to compute.
2. **Capture of physical properties:** The kernel function should capture the physical properties of the data. For example, the kernel function for the Burger's equation should be non-stationary, as the covariance between two points is dependent on the absolute location of the points, instead of the relative difference. The effectiveness of the kernels are discussed in the next section.

3.2.1 Kernel for Burger's Equation

The choice of the kernel function for the Burger's equation is:

$$k(x, x') = \frac{2}{\pi} \sin^{-1}\left(\frac{2(\sigma_0^2 + \sigma^2 x^T x')}{\sqrt{(1 + (2\sigma_0^2 + \sigma^2 x^T x))(1 + 2((2\sigma_0^2 + \sigma^2 x'^T x')))}} \right) \tag{22}$$

Due to the non-stationary nature of the Burger's equation, the stationary RBF kernel should be replaced with a non-stationary kernel.

The rational is as follows:

A stationary kernel function typically implies a translation-invariant function, meaning the kernel's response should sole depend on the distance between the points, not on the absolute positions. However, the Burger's equation has a non-stationary nature, which favours the use of a non-stationary kernel function.

[1] The exponent term is used to denote the time step, and the subscript b is used to denote the boundary data.

A good choice of a non-stationary equation is an arcsine kernel[9] and the derivative of the kernel function is obtained using Mathematica and shown in the appendixB.

3.2.2 Kernel for Wave Equation

The choice of the kernel function for the wave equation is the RBF kernel, which is defined as:

$$k(x, x') = \sigma^2 \exp\left(-\frac{(x - x')^2}{2l^2}\right) \tag{23}$$

The most common RBF kernel is a good choice for the wave equation, as it is smooth and infinitely differentiable. The derivative of the kernel function is straightforward to compute and is shown in the appendix C for reference.

3.3 Burger's Equation

The Burger's equation is a non-linear partial differential equation, which is given by:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2} \tag{24}$$

The target is to simulate the propagation of the Burger's equation in the domain of $x \in [-1, 1]$ and $t \in [0, 0.5]$.

We solve this equation together wit the Dirichlet boundary condition that $u(t, -1) = u(t, 1) = 0$ and the initial condition that $u(0, x) = -\sin(\pi x)$.

The behaviour of Burger's equation, simulated using Rutta-Kunge method, is shown in the Figure 1. The simulation is done with $\nu = 0.01$ and $\Delta x = 0.01$. The simulation shows the formation of sharp and continuous gradient at the centre of the domain, which is the characteristic of the Burger's equation.

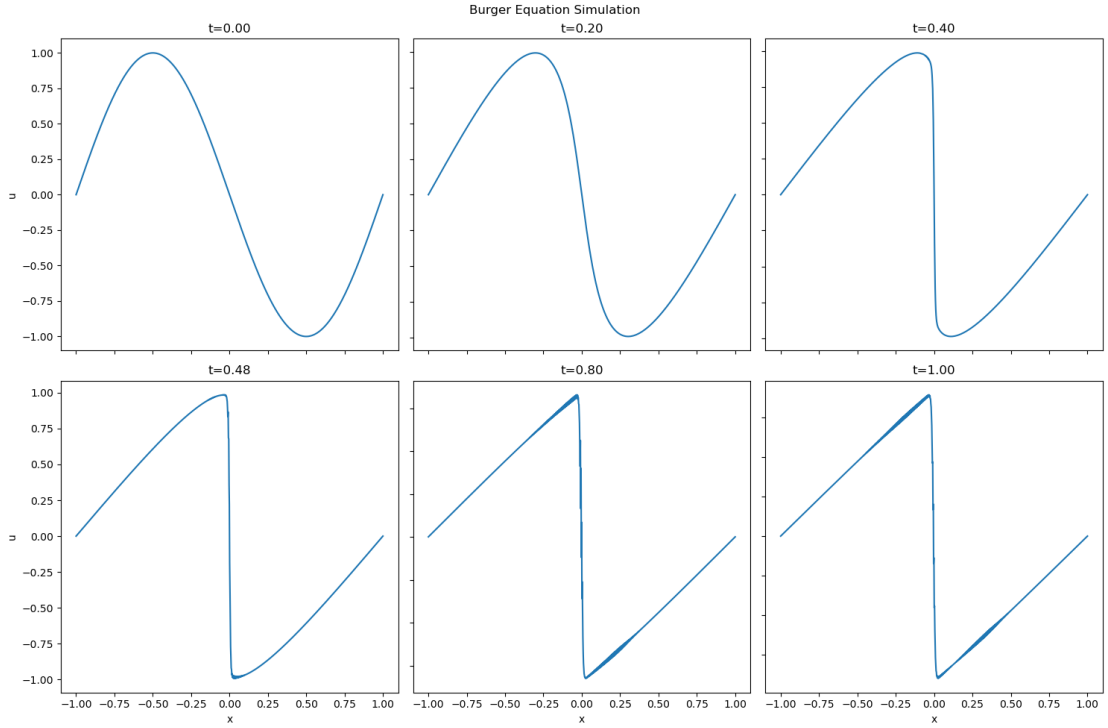

Fig. 1: Simulation of Burger's equation using Rutta-Kunge method, with $\nu = 0.01$ and $\Delta x = 0.01$. The simulation shows the formation of sharp and continuous gradient at the centre of the domain.

Apply Backward Euler's Scheme to solve the Burger's equation, which is given by:

$$u^n = u^n - \Delta t u^{n-1}\frac{d}{dx}u^{n-1} + \nu\Delta t\frac{d^2}{dx^2}u^{n-1} \tag{25}$$

where is superscript n denotes the time step.

The term $u^n\frac{d}{dx}u^n$ is not a linear term, which makes the Burger's equation a non-linear equation. It will cause a problem as the product of two Gaussian processes is not a Gaussian process. To solve this problem, we use approximate the non-linear term with

$$u^n\frac{d}{dx}u^n \approx \mu^{n-1}(x)\frac{d}{dx}u^n(x) \tag{26}$$

where $\mu^{n-1}(x)$ is the posterior mean of the previous time step.

The approximate Burger's equation, after rearranging the terms, is given by:

$$u^n + \Delta t \mu^{n-1}(x)\frac{d}{dx}u^n - \nu\Delta t\frac{d^2}{dx^2}u^n = u^{n-1} \tag{27}$$

3.3.1 Implementation of Burger Equation

We construct the following Gaussian Process:

$$\begin{bmatrix}\mathbf{u^n}\\ \mathbf{u^{n-1}}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}0\\0\end{bmatrix}, \begin{bmatrix}k_{u,u}^{n,n} & k_{u,u}^{n,n-1}\\ k_{u,u}^{n-1,n} & k_{u,u}^{n-1,n-1}\end{bmatrix}\right) \tag{28}$$

If we use kernel function $k_{u,u}^{n,n}$ to define the covariance for $u^n(x)$ and $u^n(x)'$, then the covariance of $u^n(x)$ and $u^{n-1}(x')$ is given by:

$$k_{u,u}^{n,n-1} = [I + \Delta t\mu^{n-1}(x')\frac{d}{dx'} - \nu\Delta t\frac{d^2}{dx'^2}]k_{u,u}^{n,n} \tag{29}$$

given that u^{n-1} is a linear transformation of u^n, and its covariance is also linearly transformed respectively.

Similarly, the covariance of $u^{n-1}(x)$ and $u^{n-1}(x')$ is given by:

$$k_{u,u}^{n-1,n-1} = [I + \Delta t\mu^{n-1}(x)\frac{d}{dx} - \nu\Delta t\frac{d^2}{dx^2}][I + \Delta t\mu^{n-1}(x')\frac{d}{dx'} - \nu\Delta t\frac{d^2}{dx'^2}]k_{u,u}^{n,n} \tag{30}$$

The full expansion of the elements in the covariance matrix is defined as:

$$\begin{aligned}
k_{u,u}^{n,n} &= k := k(x,x')\\
k_{u,u}^{n,n-1} &= k + \Delta t\mu^{n-1}(x')\frac{d}{dx'}k - \nu\Delta t\frac{d^2}{dx'^2}k\\
k_{u,u}^{n-1,n-1} &= k + \Delta t\mu^{n-1}(x')\frac{d}{dx'}k - \nu\Delta t\frac{d^2}{dx'^2}k,\\
&\quad + \Delta t\mu^{n-1}(x)\frac{d}{dx}k + \Delta t^2\mu^{n-1}(x)\mu^{n-1}(x')\frac{d}{dx}\frac{d}{dx'}k\\
&\quad - \nu\Delta t^2\mu^{n-1}(x)\frac{d}{dx}\frac{d^2}{dx'^2}k - \nu\Delta t\frac{d^2}{dx^2}k\\
&\quad - \nu\Delta t^2\mu^{n-1}(x')\frac{d^2}{dx^2}\frac{d}{dx'}k + \nu^2\Delta t^2\frac{d^2}{dx^2}\frac{d^2}{dx'^2}k
\end{aligned} \tag{31}$$

where the kernel function is defined as:

$$k(x,x') = \frac{2}{\pi}\sin^{-1}\left(\frac{2(\sigma_0^2 + \sigma^2 x^T x')}{\sqrt{(1+(2\sigma_0^2+\sigma^2x^Tx))(1+2((2\sigma_0^2+\sigma^2x'^Tx')))}}\right) \tag{32}$$

where σ_0 and σ are the hyperparameters of the kernel function. The derivative of the kernel function are shown in the appendix.

3.4 Wave Equation

We can extend our implementation of numerical Gaussian processes to second-order partial differential equations.

One good example is the wave equation, which is given by:

$$\frac{\partial^2 u}{\partial t^2} = c^2\frac{\partial^2 u}{\partial x^2} \tag{33}$$

The task is to simulate the propagation of the wave equation in the domain of $x \in [0,1]$ and $t \in [0,1]$. We rewrite the wave equation as a system of first-order partial differential equations, which is given by:

$$\begin{aligned}
\frac{\partial u}{\partial t} &= v\\
\frac{\partial v}{\partial t} &= c^2\frac{\partial^2 u}{\partial x^2}
\end{aligned} \tag{34}$$

The true value of the wave equation is shown in the Figure 2. The simulation is done with $c = 1$. The simulation shows the formation of the wave propagating from the left to the right of the domain.

One solution to the function, and the one that is implemented in this project, is:

$$u(t,x) = \frac{1}{2}\sin(\pi x)\cos(\pi t) + \frac{1}{3}\sin(3\pi x)\sin(3\pi t) \tag{35}$$

with Dirichlet boundary condition:

$$\begin{aligned} u(0,x) = u^0(x) &:= \frac{1}{2}\sin(\pi x), \\ u_t(0,x) = v^0(x) &:= \pi \sin(3\pi x), \\ u(t,0) = u(t,1) &= 0. \end{aligned} \tag{36}$$

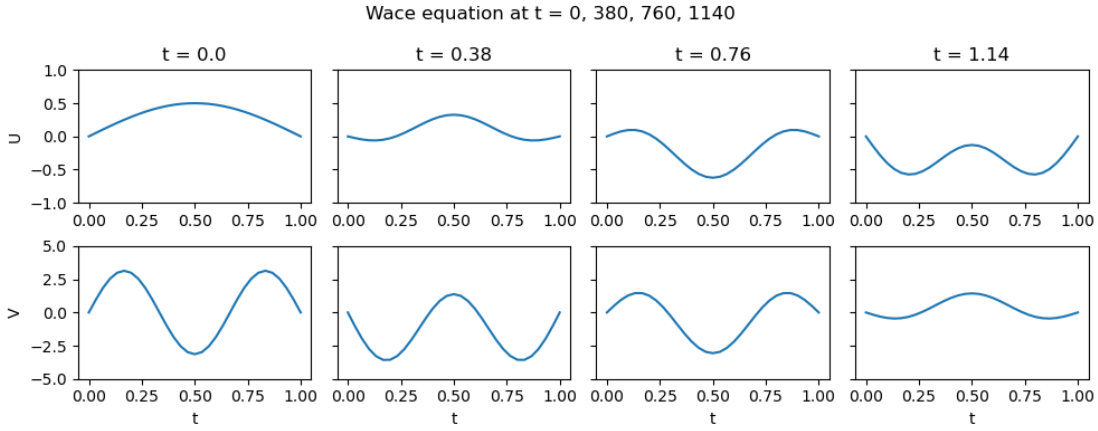

Fig. 2: Simulation of wave equation, using analytical solution for wavefunction

Similar to the backward Euler method implemented in 25 We apply the finite step method with trapezoidal rule to solve the wave equation, which is given by:

$$\begin{aligned} u^n &= u^{n-1} + \Delta t v^{n-1} + \frac{1}{2}\Delta t^2 v^{n-1} \\ v^n &= v^{n-1} + \frac{1}{2}\Delta t \left(\frac{\partial^2}{\partial x^2} u^n + \frac{\partial^2}{\partial x^2} u^{n-1} \right) \end{aligned} \tag{37}$$

where the prediction of next time step u^n and v^n are linearly related with their previous step values u^{n-1} and v^{n-1}.

3.4.1 Implementation of Wave Equation

In the implementation of numerical Gaussian Process on the wave equation, we have the following steps: Choose a base kernel function, for example, the RBF kernel function is a good choice due to its smoothness and infinitely differentiability that is required for the wave equation.

The Gaussian Process is defined as:

$$\begin{bmatrix} u^n \\ v^n \\ u^{n-1} \\ v^{n-1} \end{bmatrix} \sim GP\left(0, \begin{bmatrix} k_{u,u}^{n,n} & k_{u,v}^{n,n} & k_{u,u}^{n,n-1} & k_{u,v}^{n,n-1} \\ k_{v,u}^{n,n} & k_{v,v}^{n,n} & k_{v,u}^{n,n-1} & k_{v,v}^{n,n-1} \\ k_{u,u}^{n-1,n} & k_{u,v}^{n-1,n} & k_{u,u}^{n-1,n-1} & k_{u,v}^{n-1,n-1} \\ k_{v,u}^{n-1,n} & k_{v,v}^{n-1,n} & k_{v,u}^{n-1,n-1} & k_{v,v}^{n-1,n-1} \end{bmatrix}\right), \tag{38}$$

where the terms in the kernels are defined such that:

$$
\begin{aligned}
k_{u,u}^{n,n} &= k_u + \frac{1}{4}\Delta t^2 k_v, \\
k_{u,u}^{n,n-1} &= k_u - \frac{1}{4}\Delta t^2 k_v, \\
k_{v,v}^{n,n} &= k_v + \frac{1}{4}\Delta t^2 \frac{d^2}{dx^2}\frac{d^2}{dx'^2} k_u, \\
k_{v,v}^{n,n-1} &= k_v - \frac{1}{4}\Delta t^2 \frac{d^2}{dx^2}\frac{d^2}{dx'^2} k_u, \\
k_{u,v}^{n,n} &= \frac{1}{2}\Delta t \frac{d^2}{dx'^2} k_u + \frac{1}{2}\Delta t k_v, \\
k_{u,v}^{n,n-1} &= -\frac{1}{2}\Delta t \frac{d^2}{dx'^2} k_u + \frac{1}{2}\Delta t k_v, \\
k_{v,u}^{n,n-1} &= -\frac{1}{2}\Delta t k_v + \frac{1}{2}\Delta t \frac{d^2}{dx^2} k_u, \\
k_{u,v}^{n-1,n-1} &= k_u + \frac{1}{4}\Delta t^2 k_v, \\
k_{v,v}^{n-1,n-1} &= k_v + \frac{1}{4}\Delta t^2 \frac{d^2}{dx^2}\frac{d^2}{dx'^2} k_u, \\
k_{v,v}^{n-1,n} &= -\frac{1}{2}\Delta t k_v, \\
k_{u,v}^{n-1,n} &= -\frac{1}{2}\Delta t k_v + \frac{1}{2}\Delta t \frac{d^2}{dx^2} k_u, \\
k_{v,u}^{n-1,n-1} &= k_v - \frac{1}{4}\Delta t^2 \frac{d^2}{dx^2}\frac{d^2}{dx'^2} k_u.
\end{aligned}
$$

which is derived similarly to the Burger's equation.

where kernel k_u and k_v are chosen to be the RBF kernel function:

$$
k_{u/v}(x, x') = \sigma_{u/v}^2 \exp\left(-\frac{(x - x')^2}{2l_{u/v}^2}\right) \tag{39}
$$

3.5 Implementation in GPJax

The project is implemented with GPJax library in Python.

GPJax is a Gaussian process library built on JAX, which provides a simple interface to train and predict using Gaussian processes. Apart from the standard kernel functions, the library also provides a way to implement custom kernel functions[7].

3.5.1 Custom Kernel Functions

`Burger_Kernel()` and `Wave_Kernel()` are implemented as custom kernel functions in GPJax. They are defined in `solver/kernel.py` and are used in the the Gaussian process models the notebooks.

3.5.2 Data Structure

`X_train`, will contain the x values of the training data, as well as the time step, and additional information such as the d_n values which track which physical quantity is being tracked. Using the example of the wave equation:

$$\texttt{X\_train} = \begin{bmatrix} x_1 & t_1 & d_1 \\ x_2 & t_1 & d_1 \\ \vdots & \vdots & \vdots \\ x_n & t_1 & d_1 \\ x_1 & t_2 & d_1 \\ x_2 & t_2 & d_1 \\ \vdots & \vdots & \vdots \\ x_n & t_2 & d_1 \\ x_1 & t_1 & d_2 \\ x_2 & t_1 & d_2 \\ \vdots & \vdots & \vdots \\ x_n & t_1 & d_2 \\ x_1 & t_2 & d_2 \\ x_2 & t_2 & d_2 \\ \vdots & \vdots & \vdots \\ x_n & t_2 & d_2 \end{bmatrix}$$

where t_1 and t_2 represent the time steps $n-1$ and n respectively, d_1 and d_2 represent the u and v values respectively, and n is the number of training data points.

Y_train will contain the u and v values of the training data, in the same order as `X_train`. i.e.

$$\texttt{Y\_train} = \begin{bmatrix} u_{x=1,t=0,d=1} \\ u_{x=2,t=0,d=1} \\ \vdots \\ u_{x=n,t=0,d=1} \\ \\ u_{x=1,t=1,d=1} \\ u_{x=2,t=1,d=1} \\ \vdots \\ u_{x=n,t=1,d=1} \\ \\ u_{x=1,t=0,d=2} \\ u_{x=2,t=0,d=2} \\ \vdots \\ u_{x=n,t=0,d=2} \\ \\ u_{x=1,t=1,d=2} \\ u_{x=2,t=1,d=2} \\ \vdots \\ u_{x=n,t=1,d=2} \end{bmatrix}$$

The test data will be in the same format as the training data.

3.5.3 Replacement of `predict()` with `predict_with_noise()`

One limitation that in the GPJax `predict()` function is that, the observation standard deviation is fixed to the initial value and immutable. A new function `predict_with_noise()` is implemented to allow the observation standard deviation to be updated in every time step, taking the consideration of the noise in the observation data. The instructions of replacement can be found in the appendix D.

4 Result Analysis

The Numerical Gaussian Process is implemented to solve two Burgers' equation and Wave Equation, with a various choice of kernels, number of test points, time steps and initial error estimation. The results and analysis are presented in this section. The following aspects are discussed in this section:

- Results: Results of the numerical Gaussian Process on Burger's Equation and Wave Equation
- Uncertainty Estimation: The unique feature of numerical Gaussian Process
- Importance of Kernel Selection: The choice of kernel function
- Errors: The error of the numerical Gaussian Process at large time steps
- Computational Cost: The computational cost of numerical Gaussian Process

4.1 Results

In general, numerical Gaussian Process generates a good estimation within a reasonable time frame. The results of the GP on Burger's Equation and Wave Equation solved by GP in a noise-free environment are shown in Figure 3 and Figure 4 respectively. The results show that the numerical Gaussian process is able to capture the dynamics of the Burger's equation and the wave equation, and provide a good estimation of the true value.

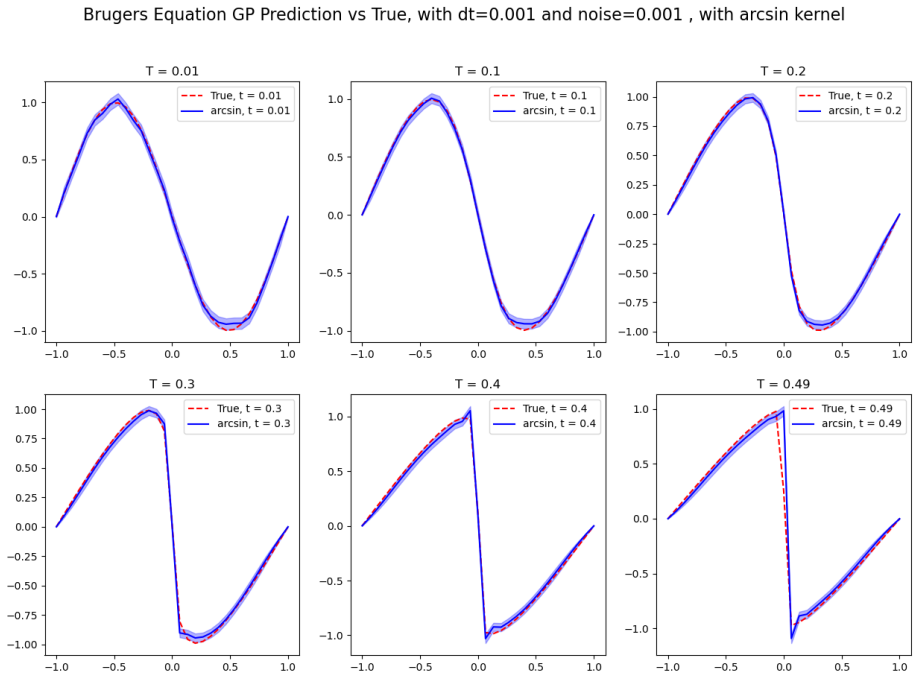

Fig. 3: Prediction of Burger's equation using numerical Gaussian Process. with noise of variance 0.001 and timestep of 0.01

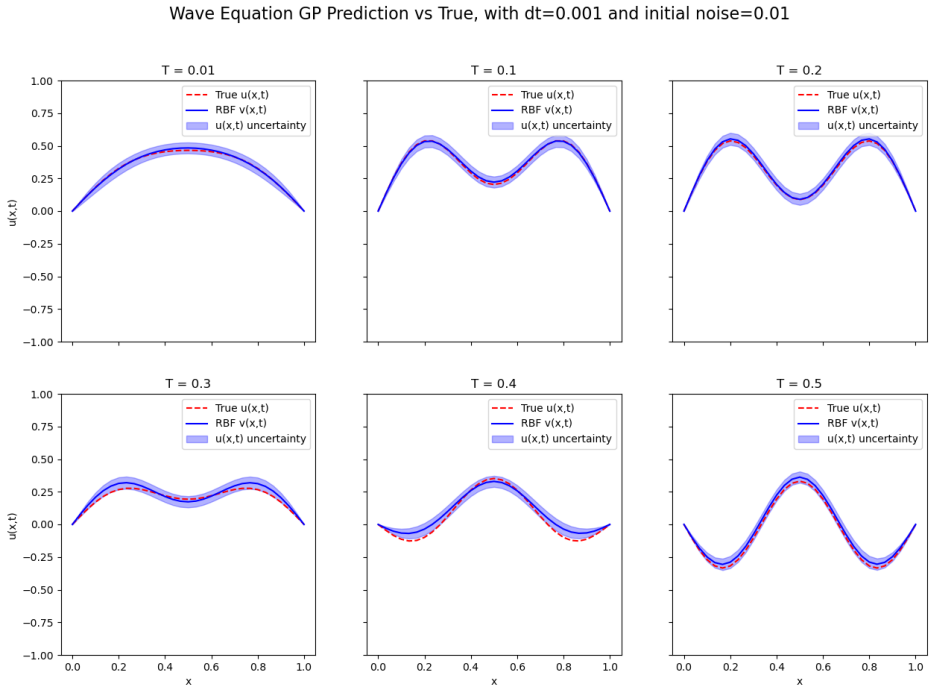

Fig. 4: Prediction of Burger's equation using numerical Gaussian Process, with noise of variance 0.01 and timestep of 0.001

4.2 Uncertainty Estimation

Uncertainty estimation is one of the unique features of numerical Gaussian Process. The uncertainty of the prediction is accumulated at every time step, and is dependent on the error in the initial data. The magnitude of the uncertainty, with respect to the error in the initial data, is shown in Figure 3. The result shows that the uncertainty of the prediction increases with the error in the initial data. This is because the uncertainty of the prediction is accumulated at every time step, and is dependent on the error in the initial data.

While finite difference methods provides a solution with error estimation, it does not provide a measure of uncertainty at every point in the domain. The numerical Gaussian Process, on the other hand, provides a measure of uncertainty at every point in the domain, which is a unique feature of the numerical Gaussian Process.

4.3 Importance of Kernel Selection

The kernel function is an important component of the numerical Gaussian process, as it captures the physical properties of the data. The choice of the kernel function will affect the prediction accuracy of the Gaussian process.

The result of the error from the numerical Gaussian process with different kernel functions is shown in Figure 5. The error is calculated as the mean squared error between the predicted value and the true value. The result shows that the numerical Gaussian process with the arcsine kernel has a lower error compared to the numerical Gaussian process with the RBF kernel.

RBF kernel is a popular choice for the Gaussian process, due to its smoothness and flexibility. However, the RBF kernel may not be suitable for the Burger's equation. This is because that a stationary kernel imply that the behaviour of the process is translational invariant. It also implies that the process is smooth and continuous, while the Burger's equation develops discontinuities at $x = 0$, for small values of viscosity parameter ν, which motivates the usage of a non-stationary kernel such as Equation 32.

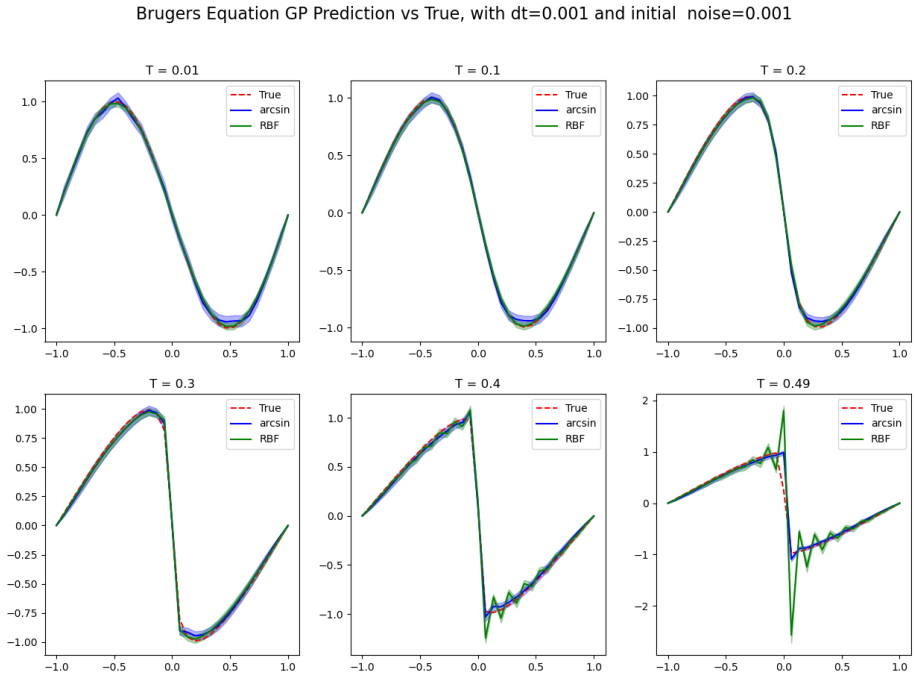

Fig. 5: Comparison of the error from the Gaussian process with different kernel functions. The RBF kernel, while has a solid performace at low

The error generated by both kernels are shown in Figure 6. Even though the RBF kernel seems to be a good choice for the Burger's equation, in the early stage of the prediction, the error of the RBF kernel is much more significant once the discontinuity is developed. The arcsine kernel, on the other hand, is able to capture the discontinuity and provide a better estimation of the true value even at the point of discontinuity..

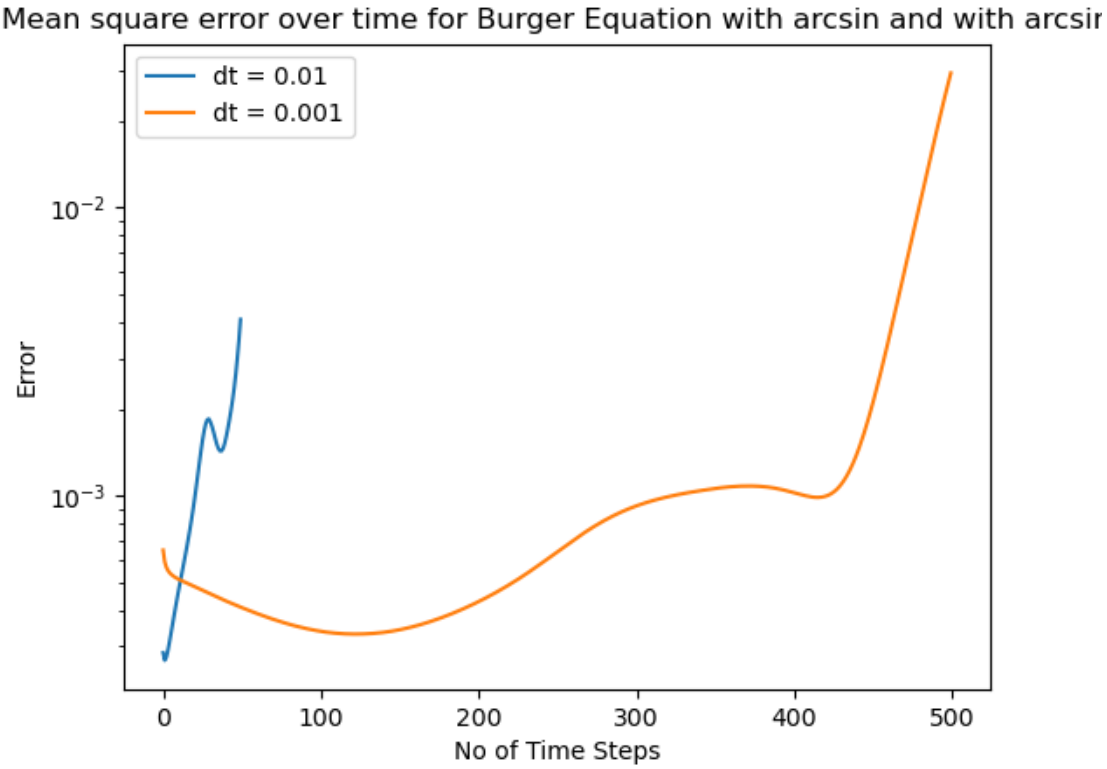

Fig. 6: Error of the numerical Gaussian process with different kernel functions. The error is calculated as the mean squared error between the predicted value and the true value. The result shows that the numerical Gaussian process with the arcsine kernel has a lower error compared to the numerical Gaussian process with the RBF kernel. The noise has a variance of 0.001 and the each time step is 0.01.

4.4 Errors

Similarly to the truncation error in Euler's method, the numerical Gaussian process has an error that is accumulated at every time step. The error of the numerical Gaussian process is calculated as the mean squared error between the predicted value and the true value.

Given a finite time step, the error of the numerical Gaussian process will eventually increase with time, just like the truncation error in Euler's method. The error of the numerical Gaussian process with respect to time is shown in Figure 7. The result shows that the error of the numerical Gaussian process increases with time, and is dependent on the time step.

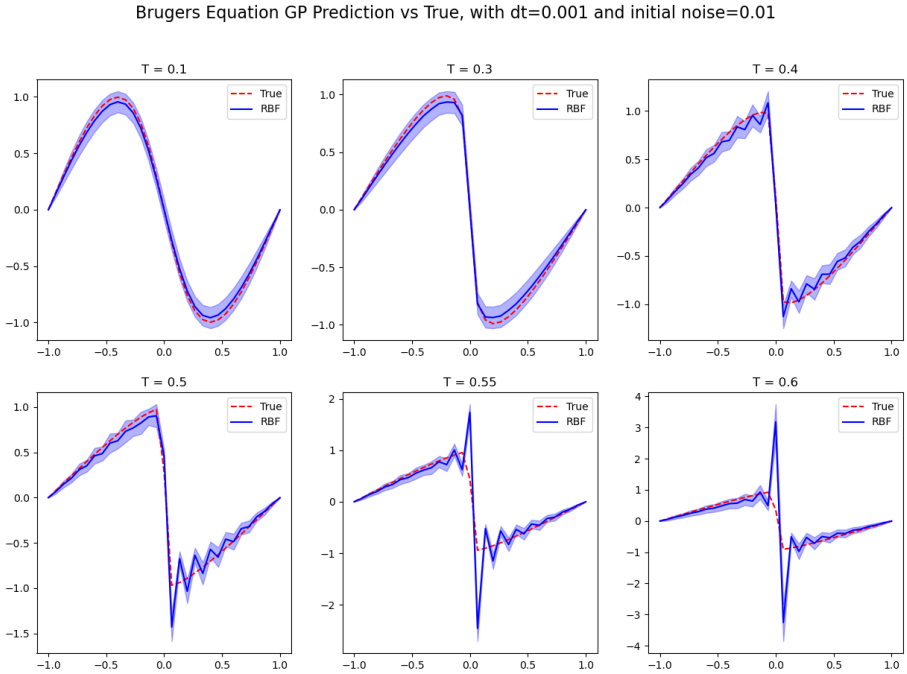

Fig. 7: Prediction of Burger's equation with different time steps. At small time steps, the numerical Gaussian process is able to provide a good estimation of the true value. However, as the time step increases, the error of the numerical Gaussian process increases, and the prediction deviates from the true value. The effect is more obvious with the RBF kernel.

The error of the Gaussian process with respect to the time step is shown in Figure 8. The error is calculated as the mean squared error between the predicted value and the true value. The result shows that the error of the Gaussian process remains at a low level for small time steps, but increases exponentially with the time step.

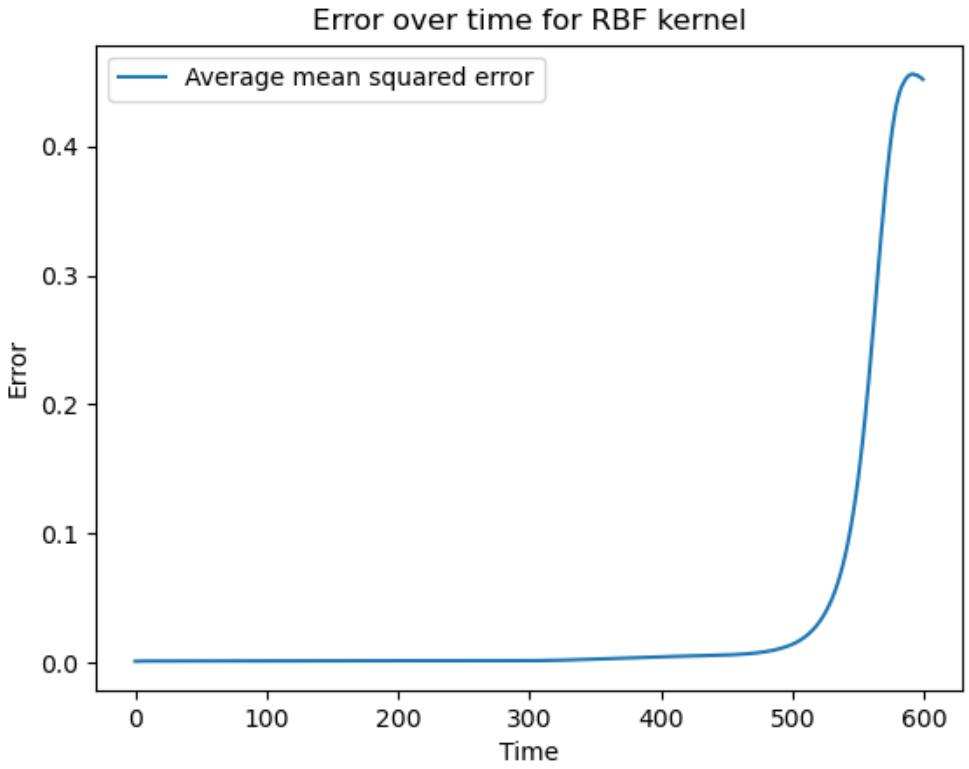

Fig. 8: Error of the Gaussian process with respect to the time step. The error is calculated as the mean squared error between the predicted value and the true value. The error of the Gaussian process remains at a low level for small time steps, but increases exponentially with the time step. The exponential can be explained by the failure to capture the discontinuity in the Burger's equation at large time steps.

It can be seen that numerical Gaussian Process is able to provide a good estimation of the true value within a reasonable time frame. However, the error of the numerical Gaussian process increases with the time step, and is dependent on the time step. Therefore, the time step should be chosen carefully to balance the accuracy of the prediction and the computational cost.

4.5 Computational Cost

The downside of numerical Gaussian Process is that it is computationally expensive for large datasets. The computational cost of Gaussian Process is $O(n^3)$, where n is the number of data points. This is because the inversion of the covariance matrix is required in the prediction step, and the inversion of a matrix is $O(n^3)$.

In practise, a few tricks are used to reduce the computational cost of Gaussian Process. For example, the Cholesky decomposition can be used to avoid the inversion of the covariance matrix, however, the computational cost is still $O(n^3)$[10].

The number of data points n, can be further decomposed into the number of points sampled n_{points} and the dimension of the data D. Note that two factors may contribute to the dimension of the data: the dimension of the domian and the number of independet variables.

In the case of the wave function, the second order linear ordinary differential equation is rewritten as two first order linear ordinary differential equations. This means that the dimension of the data is doubled, and the computational cost of the Gaussian Process will be 8 time larger.

If we are dealing with high-dimensional data, i.e. $f(x1, x2)$, the test vector x_* will be a described by a matrix of shape (n_{points}, D) that is reshaped into a vector of shape $(n_{points} \times D, 1)$ [1].

The computational cost of the Gaussian Process will be $O(n_{points}^3) \times O(D^3)$. Therefore, in practise, the Gaussian Process is not suitable for high-dimensional data. Also, the number of points sampled n_{points} should be chosen carefully to balance the computational cost and the coverage of the data.

In more complex cases, the dimension of the data can be further increased, such as higher order differential equations or differential equatons in higher dimensions, leading to a cubic increase with respect to the dimension D in the computational cost of the numerical Gaussian Process, which is not feasible for large datasets.

4.6 Further Work

Further work with the numerical Gaussian Process including observation at arbitraty time $t = t'$ to improve the accurancy of the prediction. Apart from the initial coniditions and boundary conditions, any observations can be included as part of the training data [3] [8]. This will improve the accuracy of the prediction, and provide a better estimation of the true value.

The inclusion of artificial points, that follows a certain physical constrain, e.g. (divergence free wave form) in the domain will also leads to good estimation of the true value, espcially in the case of solving the wave equation. [3].

Further model can be tested on more complex differential equations.For example: The Korteveg-de Vries equation which contains the third order term u_{xxx} [4], with general form of:

$$u_t + \lambda_1 u_x + \lambda_2 u_{xxx} = 0 \tag{40}$$

and the Kuramoto-Sivashinsky equation which contains the second order term $u_x x$ and the fourth order term $u_x xxx$ [5], with general form of:

$$u_t + \lambda_1 u u_x + \lambda_2 u_{xx} + \lambda_3 u_{xxxx} = 0 \tag{41}$$

and most famously, the 2D Navier-Stokes equation which contains the second order term u_{xx} and u_{yy} [6], with general form of:

$$u_t + \lambda_1 (u u_x + v u_y) = -p_x + \nu (u_{xx} + u_{yy}) \tag{42}$$

The difficulty of solving these equations lies in the non-linearity and the high dimensionality of the data. For example, we would need to compute the 8th order derivative of the data in the case of the Kuramoto-Sivashinsky equation, which is either computationally expensive or too complicated to derive the higher order derivative of kernel function.

5 Conclusion

This project has demonstrated the effective application of Numerical Gaussian Processes (GPs) for solving linear ordinary differential equations, specifically focusing on the Burger's equation and the wave equation. The implementation of Numerical GPs offers several advantages over traditional finite difference methods:

1. Uncertainty Quantification: Numerical GPs provide not only predictions but also uncertainty estimates, allowing for a more comprehensive understanding of the solution space.
2. Flexible Kernel Selection: The choice of kernel functions allows for the incorporation of domain-specific knowledge, as demonstrated by the superior performance of the arcsine kernel over the RBF kernel for the Burger's equation.
3. Error Propagation: Numerical GPs naturally propagate uncertainties from initial conditions through the solution, providing a more robust framework for dealing with noisy or uncertain data.

However, it should always be bear in mind that numerical GPs are not a panacea and have their own limitations and challenges, including:

1. Computational Cost: The $O(n^3)$ time complexity of GPs limits their applicability to large-scale problems or high-dimensional systems.
2. Accuracy at Large Time Scales: Similar to finite difference methods, numerical GPs suffer from accumulated errors over large time scales, necessitating careful selection of time step sizes.
3. Complexity in Higher Dimensions: The implementation becomes increasingly complex for higher-order differential equations or systems with multiple variables.

Despite these challenges, the numerical GP approach offers a promising avenue for solving differential equations, particularly in scenarios where uncertainty quantification is crucial. The method's ability to incorporate prior knowledge through kernel selection and to provide probabilistic solutions makes it a valuable tool in the computational scientist's toolkit.

Future work should focus on addressing the computational limitations, possibly through sparse GP approximations or other efficiency improvements. Additionally, extending the approach to more complex systems, such as the Navier-Stokes equations or coupled PDEs, could further demonstrate the method's versatility and applicability in real-world problems.

In conclusion, while not a replacement for traditional numerical methods, Gaussian Processes offer a complementary approach to solving differential equations, particularly valuable in situations where uncertainty quantification and incorporation of domain knowledge are paramount. As computational techniques and hardware continue to advance, the potential for GPs in scientific computing is likely to grow, opening new avenues for research and application in various fields of science and engineering.

A Use of generative AI

Chat-GPT 3.5, accessible for free at https://chat.openai.com/, was utilized for certain small sections of the source code. These instances, where using Chat-GPT provided a more efficient solution than other sources could offer, are clearly marked in the code with comments. Additionally, Chat-GPT helped in formatting some of the tables in this report using LATEX. The concepts and execution of this project are entirely my own, built upon the knowledge gained from this course's materials, my own experiences, and the cited references.

B Derivatives of the kernel function for Burger's equation

The derivatives of the kernel function for Burger's equation is a straightforward but tedious process. They can be computed using Mathematica. The kernel function for Burger's equation is defined as:

$$k(x,x') = \frac{2}{\pi}\sin^{-1}\left(\frac{2(\sigma_0^2+\sigma^2 xx')}{\sqrt{(1+(2\sigma_0^2+\sigma^2x^2))(1+2((2\sigma_0^2+\sigma^2x'^2)))}}\right) \tag{43}$$

The first order derivative of the kernel function with respect to x is:

$$\frac{\partial k(x,x')}{\partial x} = \frac{4\sigma^2(x'-2(x-x')\sigma_0^2)}{\pi(1+2x^2\sigma^2+2\sigma_0^2)^{3/2}\sqrt{1+2x'^2\sigma^2+2\sigma_0^2}\sqrt{1-\frac{4(xx'\sigma^2+\sigma_0^2)^2}{(1+2x^2\sigma^2+2\sigma_0^2)(1+2x'^2\sigma^2+2\sigma_0^2)}}} \tag{44}$$

The other derivatives of the kernel function with respect to x and x' are:

$$\frac{\partial^2 k(x,x')}{\partial x\partial x'} = \frac{4\sigma^2(1+4\sigma_0^2)}{\pi\sqrt{1+2x^2\sigma^2+2\sigma_0^2}\sqrt{1+2x'^2\sigma^2+2\sigma_0^2}(1+2x^2\sigma^2+2x'^2\sigma^2+4(1+x^2\sigma^2-2xx'\sigma^2+x'^2\sigma^2)\sigma_0^2)\sqrt{1-\frac{4(xx'\sigma^2+\sigma_0^2)^2}{(1+2x^2\sigma^2+2\sigma_0^2)(1+2x'^2\sigma^2+2\sigma_0^2)}}} \tag{45}$$

$$\frac{\partial^2 k(x,x')}{\partial x^2} = \frac{N_1+N_2+N_3}{D} \tag{46}$$

$$N_1 = 8\sigma^2(-xx'\sigma^2(3+6x^2\sigma^2+4x'^2\sigma^2)),$$

$$N_2 = 8\sigma^2\left[(-1+8x^4\sigma^4-24x^3x'\sigma^4-2xx'\sigma^2(9+8x'^2\sigma^2)+2x^2(\sigma^2+12x'^2\sigma^4))\sigma_0^2\right],$$

$$N_3 = 16\sigma^2\left[(-3+8x^4\sigma^4-24x^3x'\sigma^4-4xx'\sigma^2(3+2x'^2\sigma^2)+4x^2(\sigma^2+6x'^2\sigma^4))\sigma_0^4-8\sigma_0^6\right],$$

$$D = \pi(1+2x^2\sigma^2+2\sigma_0^2)^{5/2}\sqrt{1+2x'^2\sigma^2+2\sigma_0^2}(1+2x^2\sigma^2+2x'^2\sigma^2+4(1+x^2\sigma^2-2xx'\sigma^2+x'^2\sigma^2)\sigma_0^2)\sqrt{1-\frac{4(xx'\sigma^2+\sigma_0^2)^2}{(1+2x^2\sigma^2+2\sigma_0^2)(1+2x'^2\sigma^2+2\sigma_0^2)}}.$$

$$\frac{\partial^3 k(x,x')}{\partial x\partial x'^2} = -\frac{24\sigma^4(1+4\sigma_0^2)(x'-2(x-x')\sigma_0^2)}{\pi\sqrt{1+2x^2\sigma^2+2\sigma_0^2}\sqrt{1+2x'^2\sigma^2+2\sigma_0^2}(1+2x^2\sigma^2+2x'^2\sigma^2+4(1+x^2\sigma^2-2xx'\sigma^2+x'^2\sigma^2)\sigma_0^2)^2\sqrt{1-\frac{4(xx'\sigma^2+\sigma_0^2)^2}{(1+2x^2\sigma^2+2\sigma_0^2)(1+2x'^2\sigma^2+2\sigma_0^2)}}} \tag{47}$$

$$\frac{\partial^4 k(x,x')}{\partial x^2\partial x'^2} = -\frac{48\sigma^4(1+4\sigma_0^2)(-5xx'\sigma^2+(-1+8x^2\sigma^2-20xx'\sigma^2+8x'^2\sigma^2)\sigma_0^2+4(-1+4x^2\sigma^2-8xx'\sigma^2+4x'^2\sigma^2)\sigma_0^4)}{\pi\sqrt{1+2x^2\sigma^2+2\sigma_0^2}\sqrt{1+2x'^2\sigma^2+2\sigma_0^2}(1+2x^2\sigma^2+2x'^2\sigma^2+4(1+x^2\sigma^2-2xx'\sigma^2+x'^2\sigma^2)\sigma_0^2)^3\sqrt{1-\frac{4(xx'\sigma^2+\sigma_0^2)^2}{(1+2x^2\sigma^2+2\sigma_0^2)(1+2x'^2\sigma^2+2\sigma_0^2)}}} \tag{48}$$

Terms not mentioned above, such as $\frac{\partial k(x,x')}{\partial x'}$ can be derived by considering the symmetry of the kernel function.

C Derivatives of the squared exonential kernel

The squared exponential kernel is defined as:

$$k(x,x') = \sigma^2\exp\left(-\frac{(x-x')^2}{2l^2}\right) \tag{49}$$

The first derivative of the squared exponential kernel with respect to x is:

$$\frac{\partial k(x,x')}{\partial x} = \frac{\sigma^2}{l^2}(x-x')\exp\left(-\frac{(x-x')^2}{2l^2}\right) \tag{50}$$

The second derivative of the squared exponential kernel with respect to x is:

$$\frac{\partial^2 k(x,x')}{\partial x^2} = \frac{\sigma^2}{l^2}\left(\frac{(x-x')^2}{l^2}-1\right)\exp\left(-\frac{(x-x')^2}{2l^2}\right) \tag{51}$$

The third derivative of the squared exponential kernel with respect to x is:

$$\frac{\partial^3 k(x,x')}{\partial x^3} = \frac{\sigma^2}{l^2}\left(1-\frac{3(x-x')^2}{l^2}\right)(x-x')\exp\left(-\frac{(x-x')^2}{2l^2}\right) \tag{52}$$

The fourth derivative of the squared exponential kernel with respect to x is:

$$\frac{\partial^4 k(x,x')}{\partial x^4} = \frac{\sigma^2}{l^2}\left(\frac{3(x-x')^4}{l^4}-\frac{6(x-x')^2}{l^2}+1\right)\exp\left(-\frac{(x-x')^2}{2l^2}\right) \tag{53}$$

It should be noted that the derivatives of the squared exponential kernel with respect to x' are the negative of the derivatives with respect to x, as the kernel is a function for the difference between x and x'.

D Predict with noise

The prediction with noise is a simple process. The mean and variance of the prediction are given by: where the mean and variance of the prediction is given by:

$$\mu^n = q^T K^{-1} \begin{bmatrix} u_b^n \\ \mu^{n-1} \end{bmatrix} \tag{54}$$

and

$$\Sigma^{n,n}(x_*^n, x_*^n) = k_u^{n,n}(x_*^n, x_*^n) - q^T K^{-1} q + q^T K^{-1} \begin{bmatrix} 0 & 0 \\ 0 & \Sigma^{n-1,n-1} \end{bmatrix} K^{-1} q \tag{55}$$

Unfortunately, when retraining the data, variance of the previous artificial data points cannot be updated due to the immutable nature of `prior` class. Therefore, a new function are written under the GPJax framework to update the variance of the artificial data points. The new function is called `predict_with_noise`, and can be found within `src/utilities.py`.

To be able to use the new function, the function in the should be put under the `ConjugatePosterior` class in file `gpjax/gps.py`. Note that the new function is a crucial part of the project, as it allows the variance of the artificial data points to be updated when retraining the data. While it is not a part of the original GPJax framework, it is a necessary addition to the project, and should be mannually added to the framework to be able to use it.

References

[1] Mauricio A. Alvarez, Lorenzo Rosasco, and Neil D. Lawrence. *Kernels for Vector-Valued Functions: a Review*. 2012. arXiv: 1106.6251 [stat.ML]. URL: https://arxiv.org/abs/1106.6251.

[2] Titus A. Beu. *Introduction to numerical programming : a practical guide for scientists and engineers using Python and C/C++*. CRC Press, 2014. DOI: 10.1201/B17384.

[3] Carl Jidling et al. *Linearly constrained Gaussian processes*. 2017. arXiv: 1703.00787 [stat.ML]. URL: https://arxiv.org/abs/1703.00787.

[4] D. J. Korteweg and G. de Vries. *On the change of form of long waves advancing in a rectangular canal, and on a new type of long stationary waves*. 1895. URL: https://doi.org/10.1007/BF01608402.

[5] Y. Kuramoto. *Diffusion-induced chaos in a system of coupled oscillators*. 1978. URL: https://doi.org/10.1143/PTP.79.223.

[6] C. L. M. H. Navier. *Mémoire sur les lois du mouvement des fluides*. 1822. URL: https://doi.org/10.1016/S1631-0721(02)01413-6.

[7] Thomas Pinder and Daniel Dodd. "GPJax: A Gaussian Process Framework in JAX". In: *Journal of Open Source Software* 7.75 (2022), p. 4455. DOI: 10.21105/joss.04455. URL: https://doi.org/10.21105/joss.04455.

[8] Maziar Raissi and George Em Karniadakis. "Hidden physics models: Machine learning of nonlinear partial differential equations". In: *Journal of Computational Physics* 357 (2018). ISSN: 0021-9991. DOI: 10.1016/j.jcp.2017.11.039. URL: http://dx.doi.org/10.1016/j.jcp.2017.11.039.

[9] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. *Numerical Gaussian Processes for Time-dependent and Non-linear Partial Differential Equations*. 2017. arXiv: 1703.10230 [stat.ML].

[10] Carl Edward Rasmussen and Christopher Williams. *Gaussian Processes for Machine Learning*. 2005.