$\mathcal{GP}$ Final Write Up

December 20, 2018

Convolutions on the Kernel matrix

1. Introduction

We propose a method for approximate iterative Gaussian process inference on grid structures by expressing matrix multiplication with kernel matrix as spatial convolutions. We perform experiments to compare the performance of our method with GPytorch which uses conjugate gradient descent.

Predictive mean for a GP is:

$$\mu_L = \mathbf{K}_*(\mathbf{K_X} + \sigma_\mathbf{n}^\mathbf{2}\mathbf{I})^{-1}\mathbf{y} \tag{1}$$

the parameters for the covariance function $k(.,.)$ are optimized by maximizing the log marginal likelihood given by:

$$\log p(\mathbf{y}|X) = -\frac{1}{2}\mathbf{y}^T(\mathbf{K} + \sigma_\mathbf{n}^\mathbf{2}I)^{-1}\mathbf{y} - \frac{1}{2}\log|\mathbf{K} + \sigma_n^2 I| - \frac{n}{2}\log 2\pi \tag{2}$$

Equation (1) involves computing inverses and this is computationally expensive and we need to circumvent this problem subject to some assumptions on the training data.

We will see one of the ways in avoiding the inverse when the training data is on a grid and the kernel is stationary.

Say:

$$\mathbf{v} = (\mathbf{K_X} + \sigma_\mathbf{n}^\mathbf{2}\mathbf{I})^{-1}\mathbf{y} \tag{3}$$

We can avoid calculating the inverse by solving the linear system:

$$(\mathbf{K_X} + \sigma_\mathbf{n}^\mathbf{2}\mathbf{I})\mathbf{v} = \mathbf{y} \tag{4}$$

for $\mathbf{v}$. The same system of linear equations is solved by approximate inference methods that use conjugate gradient descent.

2. Convolutions on a grid

When the training data $X \in \mathbb{R}^{n\times n}$, say $n = 2$, and with a stationary kernel

$$k(\mathbf{x_i}, \mathbf{x_j}) = f'(\mathbf{x_i} - \mathbf{x_j}) = f(x_i^{(0)} - x_j^{(0)}, x_i^{(1)} - x_j^{(1)}) \tag{5}$$

equation (4) can be written as:

$$\mathbf{y} = \begin{bmatrix} f(0,0) & f(0,-1) & f(-1,0) & f(-1,-1) \\ f(0,1) & f(0,0) & f(-1,1) & f(-1,0) \\ f(1,0) & f(1,-1) & f(0,0) & f(0,-1) \\ f(1,1) & f(1,0) & f(0,1) & f(0,0) \end{bmatrix} \begin{bmatrix} v_{0,0} \\ v_{0,1} \\ v_{1,0} \\ v_{1,1} \end{bmatrix} + \sigma_n^2\mathbf{I}.\,\mathbf{v} \tag{6}$$

This matrix multiplication can be represented as a convolution:

$$\mathbf{y} = \mathbf{K}' * \mathbf{v}' + \sigma_n^2 \mathbf{I}. \mathbf{v} \tag{7}$$

where:

$$\mathbf{K}' = \begin{bmatrix} f(-1,-1) & f(-1,0) & f(-1,1) \\ f(0,-1) & f(0,0) & f(0,1) \\ f(1,-1) & f(1,0) & f(1,1) \end{bmatrix} \tag{8}$$

and

$$\mathbf{v}' = \begin{bmatrix} v_{1,1} & v_{1,0} \\ v_{0,1} & v_{0,0} \end{bmatrix} \tag{9}$$

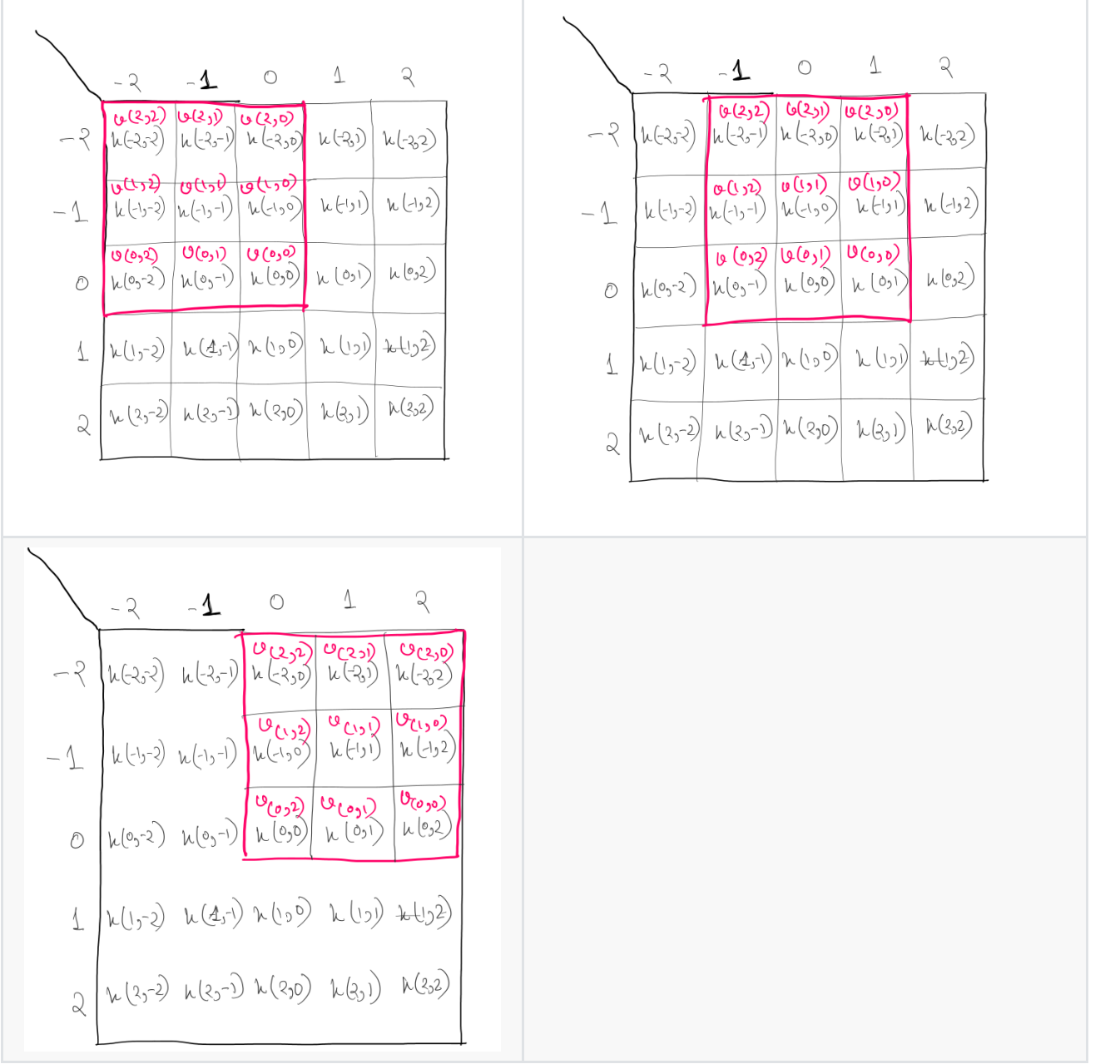
Lets see convolutions on a grid where $n = 3$:

like in equation (8)

$$\mathbf{K}' = \begin{bmatrix} f(-2,-2) & f(-2,-1) & f(-2,0) & f(-2,1) & f(-2,2) \\ f(-1,-2) & f(-1,-1) & f(-1,0) & f(-1,1) & f(-1,2) \\ f(0,-2) & f(0,-1) & f(0,0) & f(0,1) & f(0,2) \\ f(1,-2) & f(1,-1) & f(1,0) & f(1,1) & f(1,2) \\ f(2,-2) & f(2,-1) & f(2,0) & f(2,1) & f(2,2) \end{bmatrix} \tag{10}$$

and like in equation (9)

$$\mathbf{v}' = \begin{bmatrix} v_{2,2} & v_{2,1} & v_{2,0} \\ v_{1,2} & v_{1,1} & v_{1,0} \\ v_{0,2} & v_{0,1} & v_{0,0} \end{bmatrix} \tag{11}$$



Therefore

$$\mathbf{y} = \mathbf{K}' * \mathbf{v}' \tag{12}$$

where $\mathbf{y} \in \mathbb{R}^{n \times n}$ and $\mathbf{K}' \in \mathbb{R}^{(2n-1)\times(2n-1)}$. The computational complexity is still $O(n^4)$ however we have decreased the space complexity from $O(n^4)$ to $O(n^2)$ where $n \times n$ is the size of the grid.

3. Effect of Missing observations

Consider the scenario where some observations are missing on the grid and we fill the missing observations with samples from $\mathcal{N}(0, \sigma_w^2 I)$ where $\sigma_w \to \infty$. Lets say we have M observations and W missing observations for a total of N observations.

$$\mathbf{K}_N = \begin{bmatrix} \mathbf{K}_M & \mathbf{K}_{MW} \\ \mathbf{K}_{MW}^T & \mathbf{K}_W \end{bmatrix} + \begin{bmatrix} \sigma_m^2 I_M & \mathbf{0} \\ \mathbf{0} & \sigma_w^2 I_W \end{bmatrix} \tag{13}$$

from equation (3):

$$(\mathbf{K}_N + \mathbf{D}_N)^{-1}\mathbf{y} = \mathbf{v} \tag{14}$$

we can use the block matrix inversion algorithm to see what this inverse leads to:

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix}^{-1} = \begin{bmatrix} (A - BD^{-1}C)^{-1} & -A^{-1}B(I - D^{-1}CA^{-1}B)^{-1}D^{-1} \\ -D^{-1}C(A - BD^{-1}C)^{-1} & (I - D^{-1}CA^{-1}B)^{-1}D^{-1} \end{bmatrix} \tag{15}$$

where:

$$A = K_M + \sigma_M^2 I_M \tag{16}$$

$$B = K_{MW} \tag{17}$$

$$C = K_{MW}^T \tag{18}$$

$$D = K_W + \sigma_W^2 I_W = \sigma_W^2(\sigma_W^{-2}K_W + I_W) \tag{19}$$

$$\lim_{\sigma_W \to \infty} D^{-1} = 0 \tag{20}$$

Therefore equation (3) becomes

$$(\mathbf{K}_M + \sigma_M^2 I_M)^{-1}\mathbf{y}_M = \mathbf{v}_M \tag{21}$$

and

$$\mathbf{y}_M = (\mathbf{K}_M + \sigma_M^2 I_M)\mathbf{v}_M \tag{22}$$

from equation (1) the predictive mean for the entire grid will be:

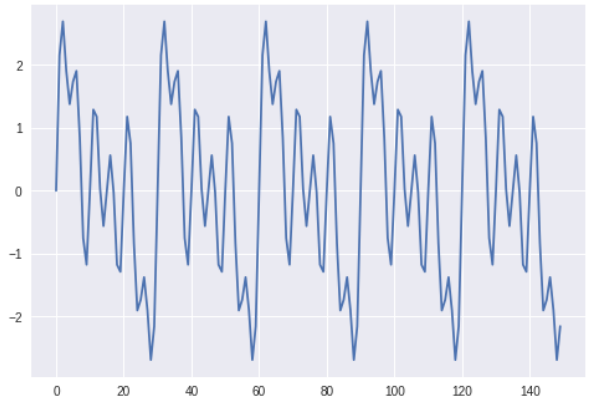
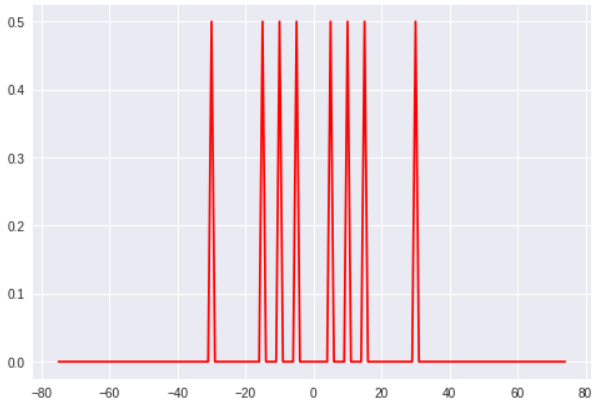
$$\mu_L = \mathbf{K}_{NM}\mathbf{v}_M \tag{23}$$

3. Analysis of GP inference in the Frequency domain

We analyze the learned GP kernel after optimizing for the log posterior for the covariance function parameters in the frequency domain. This experiment illustrates how smoothing can be done with GP inference.

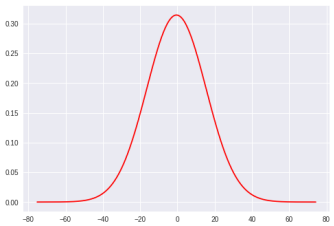
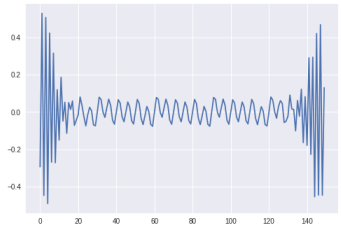
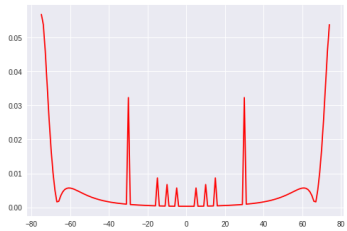
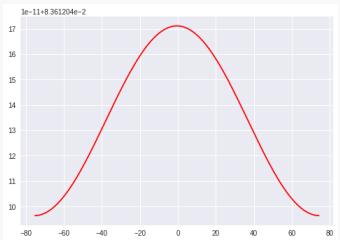
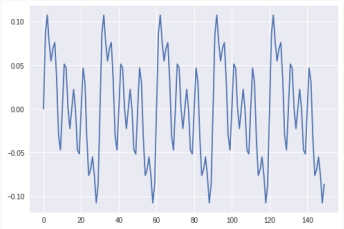
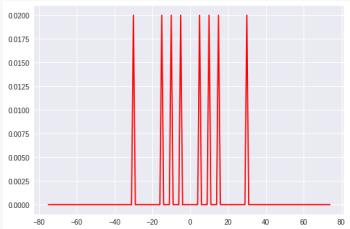
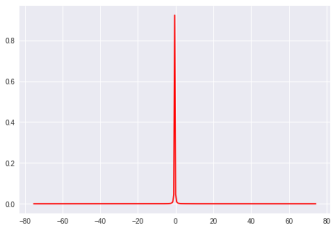
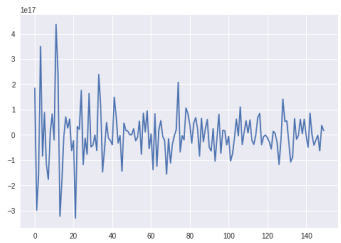
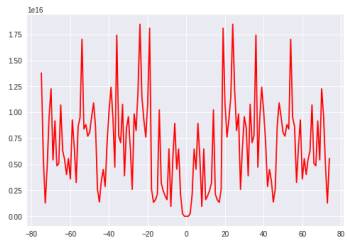
We take an input signal which is a sine wave of three different frequencies but of the same wavelength.

$$y = \sin(2\pi * 5) + \sin(2\pi * 10) + \sin(2\pi * 15) \tag{24}$$

Signal	Frequency Spectrum of the signal
	

The signal y is noise free and we are interested in v such that $y = Kv$ where K is the covariance matrix of the signal.

For the covariance function we choose the squared exponential $k(x, y) = \sigma_f^2 exp(\frac{-||x-y||_2^2}{2l^2})$.

	in freq domain	v	v frequency spectrum
5, 0.01			
5, 0.001			
1, $\sqrt{2}$			

From the above table only the first two kernels were able to recreate the signal y with machine precision(tolerance of 10^{-15}). These kernels have a wide spectrum and hence were able to approximate all the frequency components from our input signal. In fact for second case the spectrum is wide enough that v is similar to the input signal. What this means is that the kernel k should at least encompass the frequency spectrum of the input signal. We see that the RBF kernel acts as a band pass filter and to perform smoothing on the input signal we need to restrict the higher frequencies in the input signal and this is achieved by restricting the spectrum of the kernel.

4. Loss Function

4.1 Loss function based on Conjugate Gradient Descent

For a function:

$$f(x) = \frac{1}{2}x^T Ax - x^T b + c \tag{25}$$

if A is symmetric the stationary point is at:

$$Ax = b \tag{26}$$

and if A is positive definite then the solution to equation (26) is the minimum. So the solution to equation (26) can be obtained by minimizing equation (24).

So the optimization problem is:

$$min_v \frac{1}{2}\mathbf{v}^T\mathbf{K}\mathbf{v} - \mathbf{v}^T\mathbf{y} \tag{27}$$

which simplifies to:

$$\frac{1}{2}\mathbf{y}'_m(\mathbf{K}_M + \mathbf{D}_M)^{-1}\mathbf{y}'_m - \mathbf{y}'_m(\mathbf{K}_M + \mathbf{D}_M)^{-1}\mathbf{y}_m \tag{28}$$

Conjugate gradient descent with a symmetric positive definite kernel matrix will give the exact solution in n^2 iterations where $n \times n$ is the size of the grid. But convergence is only guaranteed for matrices with low condition number. We find that we could not achieve convergence with conjugate gradient descent loss function even after 10k iterations for a 50×50 grid. The most likely causes are the high condition number.

4.2 Loss function based on log posterior

We defined a new loss function based on the joint distribution:

For a set of data points X and covariance matrix K, with $y = \mathbf{f} + \epsilon$ where $\mathbf{f} \sim \mathcal{N}(0, K(X, X))$

and $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$ we can write

$$\log P(\mathbf{f}|X) \propto \frac{-1}{2}\mathbf{f}^T K^{-1}\mathbf{f} \tag{29}$$

$$\log P(y|\mathbf{f}, X) \propto \frac{-1}{2\sigma_n^2}(y - \mathbf{f})^T(y - \mathbf{f}) \tag{30}$$

$$\log P(\mathbf{f}, y|X) \propto \frac{-1}{2}\mathbf{f}^T K^{-1}\mathbf{f} - \frac{1}{2\sigma_n^2}(y - \mathbf{f})^T(y - \mathbf{f}) \tag{31}$$

we introduce a new variable $\mathbf{v}$ such that $K\mathbf{v} = \mathbf{f}$ and rewrite equation (4) as :

$$\log P(\mathbf{v}, y|X) \propto \frac{-1}{2}\mathbf{v}^T K\mathbf{v} - \frac{1}{2\sigma_n^2}(y - K\mathbf{v})^T(y - K\mathbf{v}) \tag{32}$$

the optimization problem w.r.t to equation (4) is:

$$\theta_* = argmin_\theta \frac{1}{2}\mathbf{f}^T K_\theta^{-1}\mathbf{f} + \frac{1}{2\sigma_n^2}(y - \mathbf{f})^T(y - \mathbf{f}) \tag{33}$$

where θ has the parameters for the kernel function and the optimization problem w.r.t to (5) is:

$$\mathbf{v}_* = argmin_{\mathbf{v}} \frac{1}{2}\mathbf{v}^T K_\theta \mathbf{v} + \frac{1}{2\sigma_n^2}(y - K_\theta \mathbf{v})^T (y - K_\theta \mathbf{v}) \tag{34}$$

5. Experiments

5.1 Machine Precision with naive GP

We are trying to determine if the predictive mean from naive GP and the convolutional method are equal.

That is:

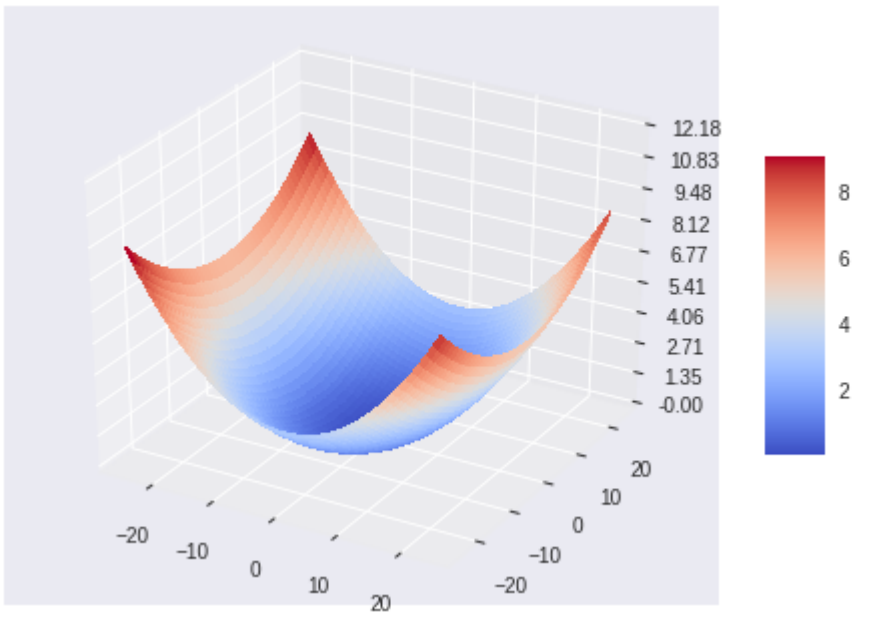
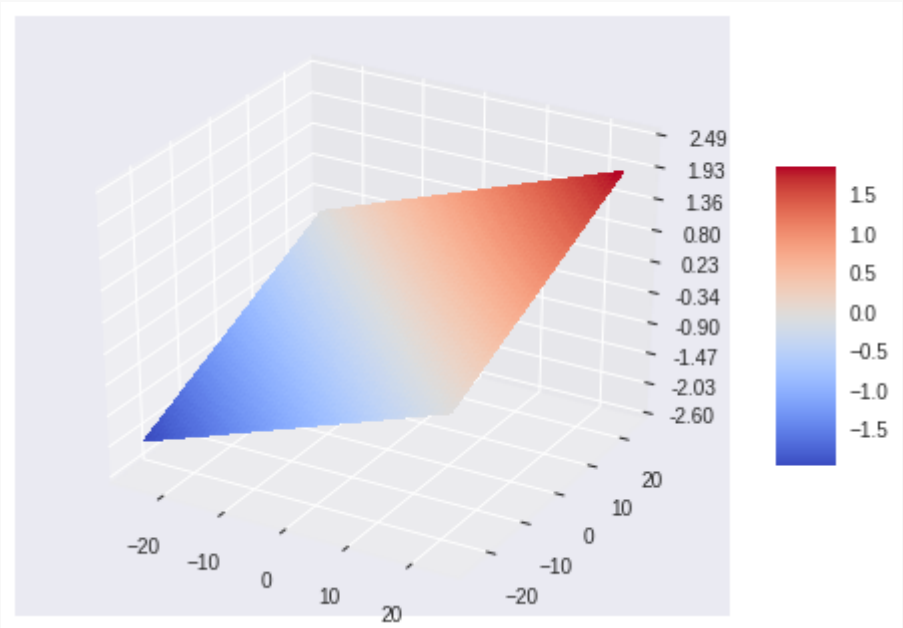
Naive:

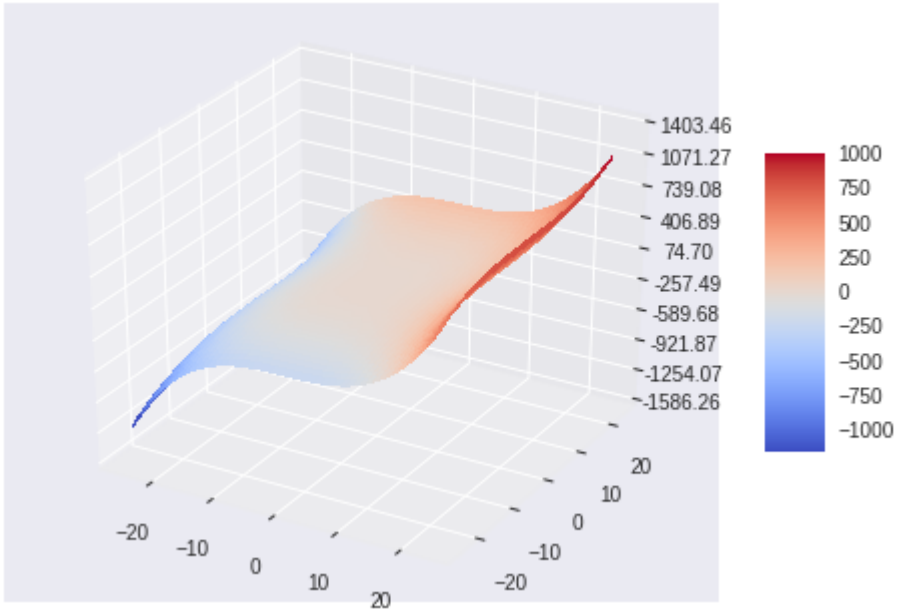
$$\mathbf{f}_* = \mathbf{K}(\mathbf{K} + \sigma_n^2 I)^{-1} y \tag{35}$$

Convolutional method:

$$v = \mathbf{K}^{-1}\mathbf{f}_* \tag{36}$$

We used three different smooth surfaces each defined between $X \in [-25, 25]$, $Y \in [-25, 25]$. We ran the network for 600 iterations, with a learning rate of 10^{-1} and we find that the predictive mean for the convolutional method and the naive GP are within machine precision of 10^{-15}. The observations have noise $\mathcal{N}(0, \sigma_n^2 I)$.

Equation	Predictive mean	SE
$\frac{X^2}{4} + \frac{Y^2}{8}$		1.7e-13
$X + Y$		8.4e-18

Equation	Predictive mean	SE
$\frac{X^3}{16} + \frac{Y^3}{64}$		9.5e-24

We achieve machine precision with naive GP with a tolerance of 10^{-15}.

5.2 Performance Comparison

We compare a time comparison of our method against naive GP and GPytorch. We have tested the performance of both of our methods for square grids of sizes between 10×10 and 60×60 with increments of 10 and we find that GPytorch always performs better and is on average three times faster on CPU and two times faster with the GPU.