Project 2

Ivan Lima do Espirito Santo
Rosa Yuliana Gabriela Paccotacya Yanque
Thiago Gomes Marçal Pereira

I. Introduction

A Gaussian process is a generalization of the Gaussian probability distribution. A Gaussian distribution in its general form is a function governed by stochastic properties (mean and covariance). We can understand this function as a infinity long vector, each entry in the vector specifying the function value $f(x)$ at a particular input x. If we ask only for the properties of the function at a finite number of points, then inference in the Gaussian process will give you the same answer if you ignore the infinitely many other points, as if you would have taken them all into account! [3]

In this project we implement a Gaussian Process and explore its accurance regarding the number of training points and the choice of different Kernels and their parameters.

II. Gaussian Processes

A. Overview

In supervised learning, we observe some inputs x_i and some outputs y_i. First, we assume that $y_i = f(x_i)$, for some unknown function f. Second, we infer a distribution over functions given the data, $p(f|X, y)$. Finally, we use this distribution to make predictions given new inputs, i.e., we compute:

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \int p(y_*|f, \mathbf{x}_*)p(f|\mathbf{X}, \mathbf{y})df \tag{1}$$

We can work with the function f in Equation 1 in its parametric representation, thus instead of inferring $p(f|D)$, we infer $p(\theta|D)$. Gaussian Process is a way to perform Bayesian inference over functions themselves,i.e. $p(f|D)$.

Gaussian process,normally abbreviated as GP, defines a **prior over functions**, which can be converted into a **posterior over functions** once we have seen some data. We only need to be able to define a distribution over the function's values at a finite, but **arbitrary**, set of points, say $x_1, ..., x_N$. A GP assumes that $p(f(x_1), ..., f(x_N))$ is **jointly Gaussian**, with some mean $\mu(\mathbf{x})$ and covariance $\mathbf{\Sigma}(\mathbf{x})$ given by $\Sigma_{ij} = \kappa(x_i, x_j)$, where κ is a positive definite kernel function (see Appendix A for more information about kernels). The key idea is that if $\mathbf{x_i}$ and $\mathbf{x_j}$ are deemed by the kernel to be similar, then we expect the output of the function at those points to be similar. See [3] for detailed explanations.

RA: 956694 - i956694@g.unicamp.br
RA: 263068 - r263068@dac.unicamp.br
RA: 189691 - t189691@g.unicamp.br

B. GPs for regression

Let the prior on the regression function be a GP, denoted by:

$$f(x) \sim GP(m(x), \kappa(x, \acute{x})) \tag{2}$$

where $m(x)$ is the mean function and $\kappa(x, \acute{x})$ is the kernel or covariance function, defined by:

$$m(x) \quad = \quad \mathbb{E}(f(x)) \tag{3}$$

$$\kappa(x, \acute{x}) = \mathbb{E}[(f(x) - m(x))(f(\acute{x}) - m(\acute{x}))^T] \tag{4}$$

We require $\kappa(.)$ to be a positive definite kernel. So, for any finite set of points, this process defines a joint Gaussian:

$$p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\mu, \mathrm{K}) \tag{5}$$

where $\mathrm{K}_{ij} = \kappa(x_i, x_j)$ and $\mu = (m(x_1), ..., m(x_N))$

C. Predictions using noise-free observations

Suppose we observe a noise-free training data where $f_i = f(x_i)$ is the noiseless observation of the function evaluated at x_i. Given a test data X_* of size $N_* \times D$, we want to predict the function outputs f_*. GP works as an **interpolator** of the training data. The joint distribution has the following form: (see [2] chapter 4 for detailed explanation on Gaussian properties)

$$\begin{pmatrix} f \\ f_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu \\ \mu_* \end{pmatrix}, \begin{pmatrix} \mathrm{K} & \mathrm{K}_* \\ \mathrm{K}_*^T & \mathrm{K}_{**} \end{pmatrix}\right) \tag{6}$$

From the Gaussian conditioning rules ([1] definition 78), the **posterior** has the form:

$$p(f_*|X_*, X, f) = \mathcal{N}(f_*|\mu_*, \Sigma_*) \tag{7}$$

$$\mu_* = \mu(X_*) + \mathrm{K}_*^T\mathrm{K}^{-1}(f - \mu(X)) \tag{8}$$

$$\Sigma_* = \mathrm{K}_{**} - \mathrm{K}_*^T\mathrm{K}^{-1}\mathrm{K}_*) \tag{9}$$

If we use squared exponential kernel, it assumes the form:

$$\kappa(x, \acute{x}) = \sigma_f^2 exp(-\frac{1}{2\ell^2}(x - \acute{x})^2) \tag{10}$$

D. Predictions using noisy observations

In this case, the function assumes the form $y = f(x) + \epsilon$, where $\epsilon \sim \mathcal{N}(0, \sigma_y^2)$. The covariance of the observed noise response is:

$$cov\left[y|X\right] = \mathrm{K} + \sigma_y^2 \mathrm{I}_N = \mathrm{K}_y \tag{11}$$

where the matrix is diagonal due to the assumption that the noise terms are independently added to each observation. The joint density of the observed data is then:

$$\begin{pmatrix} y \\ f_* \end{pmatrix} \sim N\left(0, \begin{pmatrix} \mathrm{K}_y & \mathrm{K}_* \\ \mathrm{K}_*^T & \mathrm{K}_{**} \end{pmatrix}\right) \tag{12}$$

and the **posterior predictive density** assumes the form:

$$p(f_*|X_*, X, f) = \mathcal{N}(f_*|\mu_*, \Sigma_*) \tag{13}$$

$$\mu_* = \mu(X_*) + \mathrm{K}_*^T \mathrm{K}_y^{-1} y) \tag{14}$$

$$\Sigma_* = \mathrm{K}_{**} - \mathrm{K}_*^T \mathrm{K}_y^{-1} \mathrm{K}_*) \tag{15}$$

We can write the posterior mean in another way, as follows:

$$\bar{f}_* = \mathrm{K}_*^T \mathrm{K}_y^{-1} y = \sum_{i=1}^{N} \alpha_i \kappa(x_i, x_*) \tag{16}$$

where $\alpha = K_y^{-1} y$. More on the posterior predictive density on section IV.

E. Effect of the kernel parameters

To illustrate the influence of the kernel parameter on the prediction, suppose we choose the squared-exponential (SE) kernel for the noisy observations, as following:

$$\kappa_y(x_p, x_q) = \sigma_f^2 exp(-\frac{1}{2\ell^2}(x_p - x_q)^2) + \sigma_y^2 \delta_{pq} \tag{17}$$

Note that ℓ is the horizontal scale over which the function changes, σ_f^2 controls the vertical scale of the function, and σ_y^2 is the noise variance.

F. Estimating the kernel parameters

Considering an empirical Bayes approach and maximizing the marginal likelihood, which in log has the following form:

$$\begin{aligned} log(p(y|X)) = \\ &= log(\mathcal{N}(y|0, \mathrm{K}_y)) \\ &= -\frac{1}{2} y \mathrm{K}_y^{-1} y - \frac{1}{2} log\left|\mathrm{K}_y\right| - \frac{N}{2} log(2\pi) \end{aligned} \tag{18}$$

The first term is a data fit term, the second term is a model complexity term, and the third term is just a constant. Now denoting the kernel hyper parameters by θ, we have:

$$\begin{aligned} \frac{\partial}{\partial \theta_j} log(p(y|X)) = \\ &= \frac{1}{2} y^T \mathrm{K}_y^{-1} \frac{\partial \mathrm{K}_y}{\partial \theta_j} \mathrm{K}_y^{-1} y - \frac{1}{2} tr(\mathrm{K}_y^{-1} \frac{\partial \mathrm{K}_y}{\partial \theta_j}) \end{aligned} \tag{19}$$

$$\frac{\partial}{\partial \theta_j} log(p(y|X)) = \frac{1}{2} tr((\alpha\alpha^T - \mathrm{K}_y^{-1}) \frac{\partial \mathrm{K}_y}{\partial \theta_j}) \tag{20}$$

where $\alpha = K_y^{-1} y$. One can note from equations 19 and 20 that the partial derivative depends on the form of the kernel. Given an expression for the log **marginal likelihood** and its **derivative**, we can estimate the kernel parameters using any standard **gradient-based optimizer**.

An alternative to computing a point estimate of the hyper-parameters is to compute their posterior via Bayesian inference for the hyper-parameters. More on that in [2], section 15.2.4.2. Also, one can use a quite different approach to optimizing kernel parameters known as **multiple kernel learning**. See [2], section 15.2.4.3 for more details.

G. Computational and numerical approach

The predictive mean depends on a matrix inversion. For reasons of numerical stability, it is unwise to directly invert it. A more robust alternative is to compute a Cholesky decomposition.

Summarizing, the six steps sequence for the GP approach is as follows:

1) $\mathbf{L} = \mathbf{cholesky}(\mathbf{K} + \sigma_\mathbf{y}^\mathbf{2}\mathbf{I})$
2) $\alpha = \mathbf{L}^\mathbf{T} \setminus (\mathbf{L} \setminus \mathbf{y})$
3) $\mathbf{E}[\mathbf{f}_*] = \mathbf{K}_*^\mathbf{T}\alpha$
4) $\mathbf{V} = \mathbf{L} \setminus \kappa_*$
5) $\mathbf{var}[\mathbf{f}_*] = \kappa(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\mathbf{T}\mathbf{v}$
6) $\mathbf{logp}(\mathbf{y}|\mathbf{X}) = -\frac{\mathbf{1}}{\mathbf{2}}\mathbf{y}^\mathbf{T}\alpha - \mathbf{\Sigma_i log L_{ii}} - \frac{\mathbf{N}}{\mathbf{2}}\mathbf{log}(\mathbf{2}\pi)$

Cholesky is a way to solve a linear system as follows:
$Ax = b$
$A = L^T L$ once A is a positive defined matrix, we have:
$L^T[Lx] = b$
$L^T\theta = b$, solve for θ.
$Lx = \theta$, solve for x.

III. Project Description

In this project we implement a Bayesian Optimization procedure using Gaussian Processes(GPs). We encapsulate the behavior of the GPs in a class that can be instantiated as follows:

```
gp = GP(kernel, sigma)
gp.addData(xtrain, ytrain)
mu, s2 = gp.posterior(xtest)
```

where GP is the class we created, kernel is any kernel function, and, sigma is the prior of the noise's standard deviation. Thus, our implementation encapsulates the Bayesian Optimization process in a single class.

IV. Implementation

Given an (n, d)-array of data xtrain and an n-vector ytrain, our python code predict the distribution of the (m, d) array of test data xtest. In order to do so, we compute the predictive posterior parameters μ and Σ of the function distribution. They are the mean and the covariance, respectively. Our default

kernel function is the widely known squared-exponential. For the one-dimensional case it is defined as:

$$\kappa_\alpha(x, \acute{x}) = \sigma^2 exp\left(-\frac{(x-\acute{x})^2}{2\ell^2}\right) \tag{21}$$

where $\alpha = (\sigma, \ell)$ is the vector of hyperparameters, and x and $\acute{x}$ are the data points used in the kernel, where the covariance matrix entries are $K_{ij} = k_\alpha(x_i, x_j)$. In general, the kernel receives two arrays of size (n, d) and (m, d) and returns an (n, m)-array of all the pairwise evaluations. The kernel signature is: kernel $(x1, x2, ell = 1.0, s = 1.0)$, where the parameters are optional.

Note that ell is ℓ and s is the σ in equation 21. We also call s as σ_f to differentiate from the noise variance, which is called σ_y

ℓ is the horizontal scale over which the function changes, σ_f^2 As already said, we have the following Gaussian joint distribution:

$$\begin{pmatrix} f \\ f_* \end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix} \mu \\ \mu_* \end{pmatrix}, \begin{pmatrix} \mathrm{K} & \mathrm{K}_* \\ \mathrm{K}_*^T & \mathrm{K}_{**} \end{pmatrix}\right) \tag{22}$$

We derive the posterior function by using the properties of a joint and condition distribution of a MVN. Equation 4.68 of [2]. The proof of the theorem stated in this equation requires inverse of a partitioned matrix using Schur complements and the matrix inversion lemma. One can find the entire algebraic steps in section 4.3.4 of [2]

Therefore, we implement the posterior function as stated in equations 7, 8, and 9.

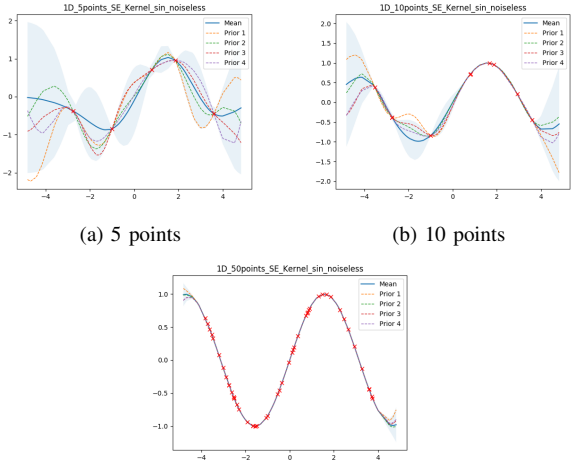
V. Experiments and Discussions

For our experiments we create a data set for training (seed 42) and testing (seed 33). we defined two fixed seeds for each data set so we can have the same results every time the code is executed. For our 1D tests we use sine, square and cubic functions with SE kernel (square exponential). We sample 5,10,50 points from each of them to define our train points. We repeat this very same process for Rational, Ornstein, and Gamma Kernel. For Rational Kernel we use $\alpha = 0.5$, and for Gamma Kernel we use $\gamma = 0.2$ for all the experiments. See Appendix A for Kernels definition.

A. Different Amount of Data

To observe the behavior of the learned function, we plot the mean plus minus two standard deviations to see the variation (uncertainty) of the predicted functions.

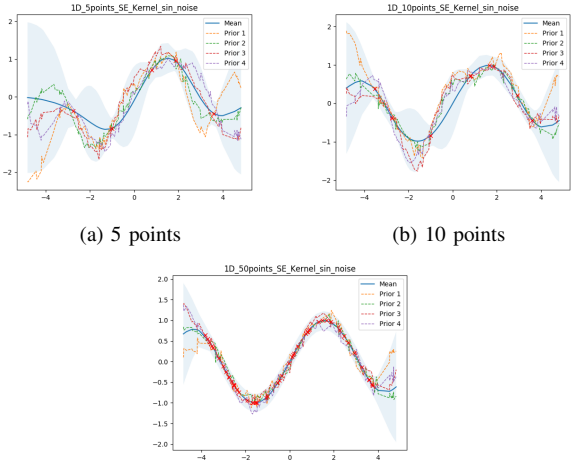
Figure 1 shows the results for 5, 10, and 50 number of points with and without noise for SE kernel and sin function. The shaded area represents the expectation of $f(x) \pm 2std(f(x))$ One can see from the behavior that for 5 points the shape of the curve resembles a sin function with zero uncertainty in the 5 points. For points other than training points, the uncertainty is high. As we increase the number of points, the curve tends to a sin in the region of the data training points. For other regions the uncertainty is still pretty high. In other words, we have a very good prediction on the training points region. From Figure 2, we see that the noise prevents perfect fit even in the training points due to the uncertainty caused by the prior noise. However, dispite the noise, it's still a good prediction.



(a) 5 points (b) 10 points

(c) 50 points

Fig. 1: Square Exponential Kernel for a noise-free sin function.



(a) 5 points (b) 10 points

(c) 50 points

Fig. 2: Square Exponential Kernel for a noise (0.1) sin function.

We observe the same behavior for other functions and kernels. See Appendix B for Rational, Ornstein, and Gamma Kernels and square and cubic functions.

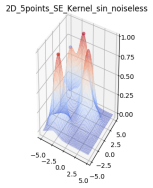
B. Higher Dimensional Data

Our implementation is general enough to evaluate higher dimensional data. The results for 2D data is shown in figures 3.

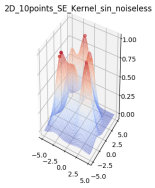
We observe a increase in accuracy as the number of training points gets higher.

C. Effect of Kernel Parameters

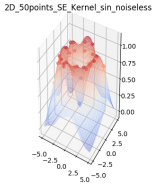
To understand the influence of the kernel parameters on the prediction, we need to have in mind the equation 23. If



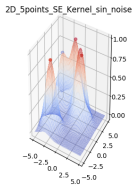
(a) 5 points



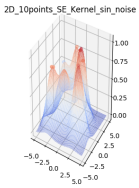
(b) 10 points



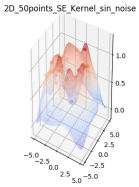
(c) 50 points



(d) 5 points



(e) 10 points



(f) 50 points

Fig. 3: Square Exponential Kernel for a sin function (a),(b), and (c) noise-free, and (d), (e), and (f) with 0.1 noise variance

we choose the squared-exponential (SE) kernel for a noisy observations:

$$\kappa_y(x_p, x_q) = \sigma_f^2 exp(-\frac{1}{2\ell^2}(x_p - x_q)^2) + \sigma_y^2 \delta_{pq} \qquad (23)$$

Note that ℓ is the horizontal scale over which the function changes, σ_f^2 controls the vertical scale of the function, and σ_y^2 is the noise variance.

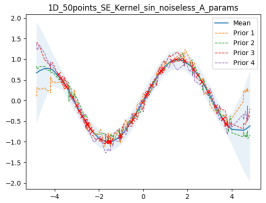
To illustrate how the kernel parameters influence the prediction we run with different combination of them. The parameters are chosen according table I.

parameters		(a)	(b)	(c)
horizontal scale	ℓ	1.0	0.3	3.0
vertical scale	σ_f^2	1.0	1.08	1.16
noise variance	σ_y^2	0.1	0.00005	0.89

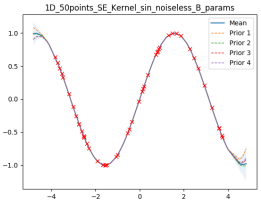
TABLE I: Parameters set.

Figure 4 shows the results with 50 training points.

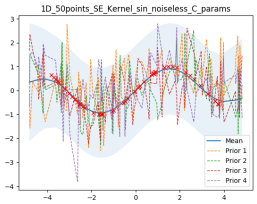
The results in (a) are a good fit. At (b) the function looks smoother. At experiment (c), the function looks more "wiggly"



(a)



(b)



(c)

Fig. 4: Square Exponential Kernel for different set of parameters.

and the uncertainty goes up faster, since the effective distance from the training points increases more rapidly.

We can find the optimal parameter by minimizing the negative log marginal likelihood with respect to the parameter (equation 18).The parameter optimization is shown the table II.

initial	1.0	0.5	2.0	2.0	3.0
initial	1.0	1.0	2.0	5.0	4.0
optimal	3.01	2.75	2.82	3.15	2.91
optimal	3.68	2.28	2.61	4.42	3.0
likelihood	-136.89	-137.06	-137.11	-136.63	-137.09

TABLE II: Parameters set.

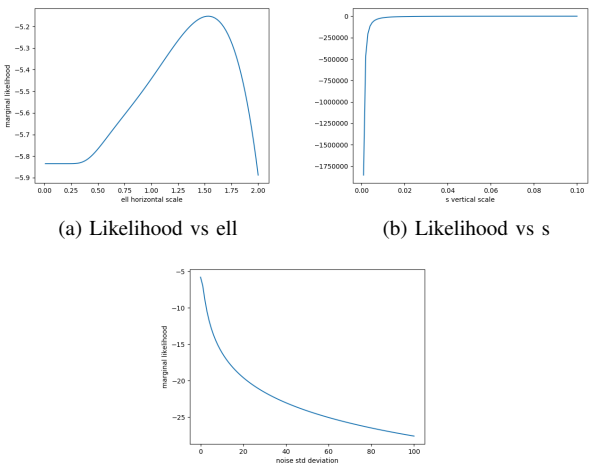
We see from the results that if we minimize directly from equation 18 does not work due the complexity of the marginal likelihood function. We should have use equation 20 instead with a **gradient-based optimizer**. Moreover, implement with Cholesky decomposition to make more stable. Note that the gradient depends on with respect to which parameter we derivative partially . Therefore, the gradient depends on the Kernel type as well.

If we fix one parameter and calculate the marginal likelihood as a function of only one of them, we can have a idea of their optimal value. See figure 5

Note from figure 5 (a) that the ℓ optimal occurs about 1.6 when fixing $\sigma_f^2 = s = 1.0$ and $\sigma_y^2 = noise = 0.00005$. From figure 5 (b), fixing $\ell = 1.0$ and $\sigma_y^2 = noise = 0.00005$, the $\sigma_f^2 = s$ too small affects negatively the likelihood, but only up to a certain value when its incriase has is minor impact on the likelihood. Finally, from figure 5 (c), fixing $\ell = 1.0$ and $\sigma_f^2 = s = 1.0$, as the variance of noise increase, the uncertain increase, as expected. See the Appendix E for the results using others Kernels.

Another way to visualize the parameter influence on the prediction is to plot the contour of the negative marginal likelihood at some parameter combination. See figure 6

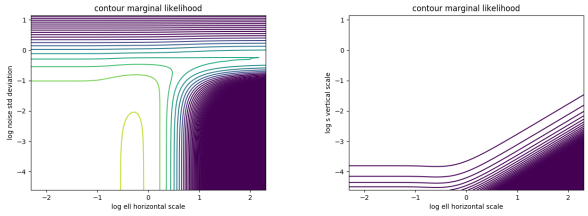
See Appendix F for contour plots with other Kernels.



(a) Likelihood vs ell (b) Likelihood vs s

(c) Likelihood vs noise

Fig. 5: Likelihood vs horizontal scale ell.



(a) ell vs noise (b) ell vs s

Fig. 6: Likelihood contour plots for Ornstein Kernel and sin function.

Also, the equation 23 can be generalized for higher dimension cases by introducing a term M in equation:

$$\kappa_y(x_p, x_q) = \sigma_f^2 exp(-\frac{1}{2\ell^2}(x_p - x_q)^T M(x_p - x_q)) + \sigma_y^2 \delta_{pq} \tag{24}$$

So, we can run experiments in the same way with the elements of the M matrix to understand the influence they have on the prediction.

D. Effect of Kernel

The Kernel defines the variance function between the points. It is by definition symmetric positively defined. See Appendix A for different Kernels definitions.

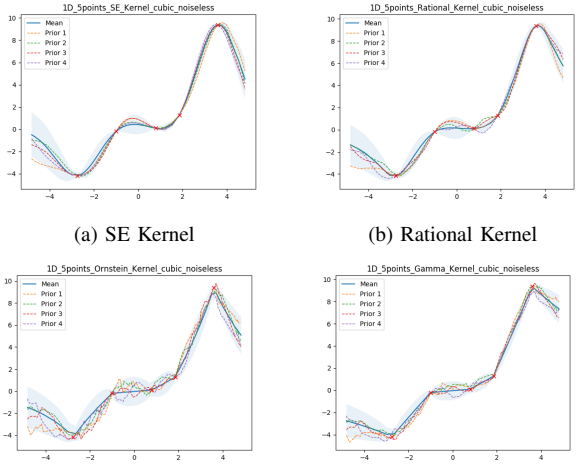
Figure 7 shows the same experiment with four different Kernels.

Note that some Kernels works better with low number of training points, however, all of them provide a good fit with more data points.

VI. Conclusions

In conclusion, the simple idea of finding a distribution over possible functions that are consistent with the observed data, combined with the Bayesian approach that updates the prior as more data is observed and generating posterior distribution over functions, proved to be extremely powerful. It can be understood as a generalization of other techniques with the advantages of the straight forward implementation for both regression and classification problems.

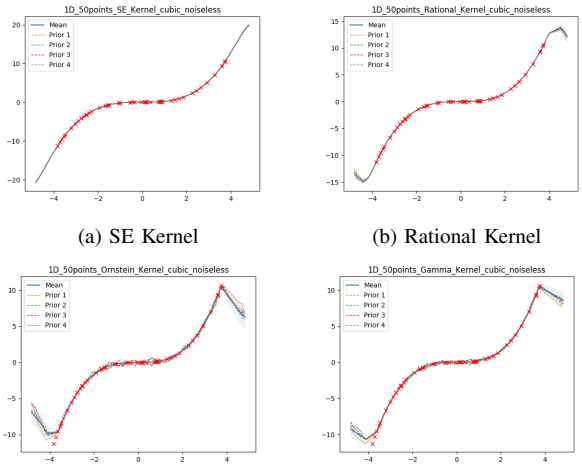
Although the accuracy of the predictions is closely related to the number of data training points, we achieve reasonable efficiency with a relatively low amount. Regarding the variance function defined by the Kernel, we can choose them from a variety of types and tuning its parameter to attend our predictions requirements of time and accuracy. As lesson learnt, the Cholesky decomposition to avoid matrix inversion proves to be the preferable implementation to gain computational speed and avoid numerical instability. Furthermore, we realized that taking the negative marginal likelihood in its log form and applying a python minimization built-in method to find the optimal kernel parameter should be avoided. Instead, we should implement the partial derivatives with respect to each hyperparameter and use a gradient-based optimizer.



(a) SE Kernel (b) Rational Kernel

(c) Ornstein Kernel (d) Gamma Kernel

Fig. 7: 4 different Kernels for a cubic function with 5 training points.



(a) SE Kernel (b) Rational Kernel

(c) Ornstein Kernel (d) Gamma Kernel

Fig. 8: 4 different Kernels for a cubic function with 50 training points.

REFERENCES

[1] D. Barber. *Bayesian Reasoning and Machine Learning*. Springer, 2010.
[2] K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press Cambridge, Massachusetts London,England, 2012.
[3] C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Massachusetts, 2006.

APPENDIX A
COVARIANCE FUNCTION

Defination of Covariance function: Given any collection of points $x^1, ..., x^M$, a covariance function $k(x_i, x_j)$ defines the elements of a $M \times M$ matrix:

$$[C_{ij}] = k(x_i, x_j) \tag{25}$$

such that C is positive semidefinite.

A. Stationary Kernel

A kernel $\kappa(x, \acute{x})$ is stationary if the kernel depends only on the separation $(x - \acute{x})$. That is

$$\kappa(x, \acute{x}) = \kappa(x - \acute{x}) \tag{26}$$

1) Squared Exponential:

$$\kappa(d) = e^{-|d|^2} \tag{27}$$

where $d = (x - \acute{x})$

2) γ-Exponential:

$$\kappa(d) = e^{-|d|^\gamma} \quad , 0 < \lambda \leq 2 \tag{28}$$

3) Matérn:

$$\kappa(d) = |d|^\nu K_\nu \left(|d|\right) \tag{29}$$

where K_ν is a modified Bessel function, $\nu > 0$.

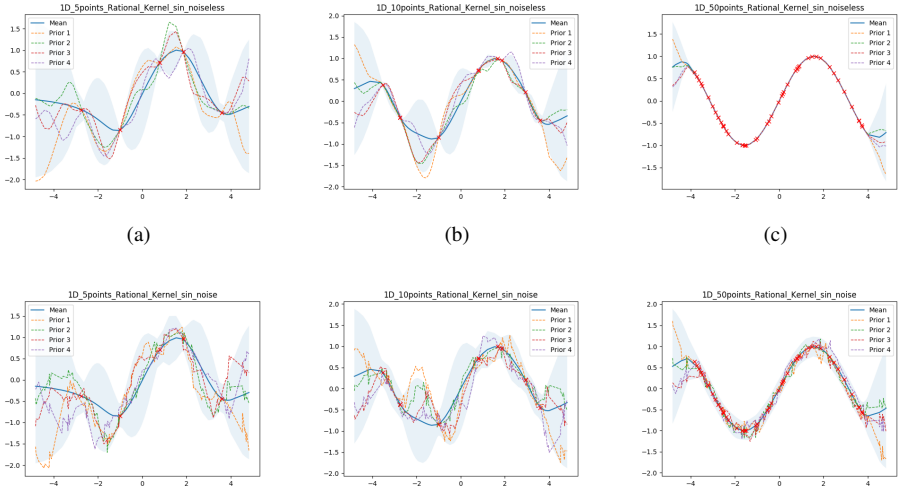
4) Rational Quadratic:

$$\kappa(d) = \left(1 + |d|^2\right)^{-\alpha} \quad , \alpha > 0 \tag{30}$$

5) Periodic: For 1-dimensional x and $\acute{x}$, a stationary (and isotropic) covariance function can be obtained by first mapping x to the two dimensional vector $u(x) = (cos(x), sin(x))$ and then using the SE covariance $e^{-\left(u(x) - u(\acute{x})\right)^2}$

$$k(x - \acute{x}) = e^{-\lambda sin^2(\omega(x - \acute{x}))} \quad , \lambda > 0 \tag{31}$$

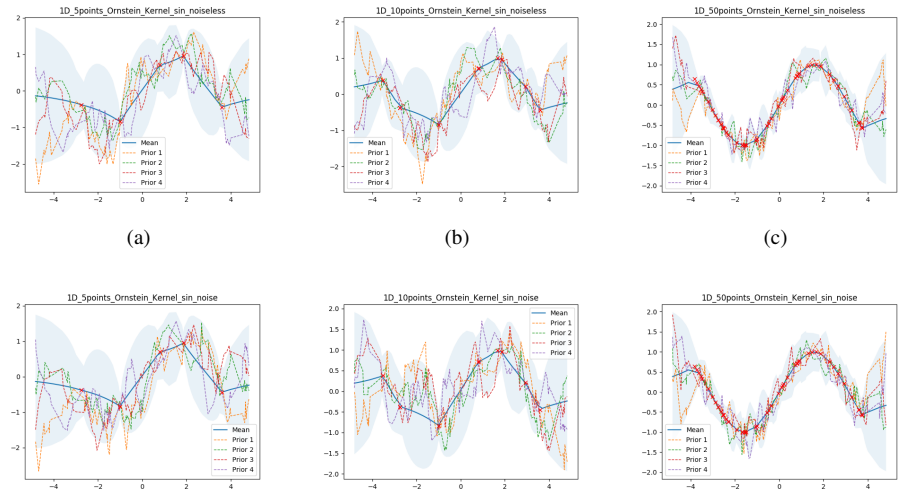
APPENDIX B
EXPERIMENTING IN 1 DIMENSION - CHANGING KERNEL, INPUT SIZE AND FUNCTION



(a) (b) (c)

(d) (e) (f)

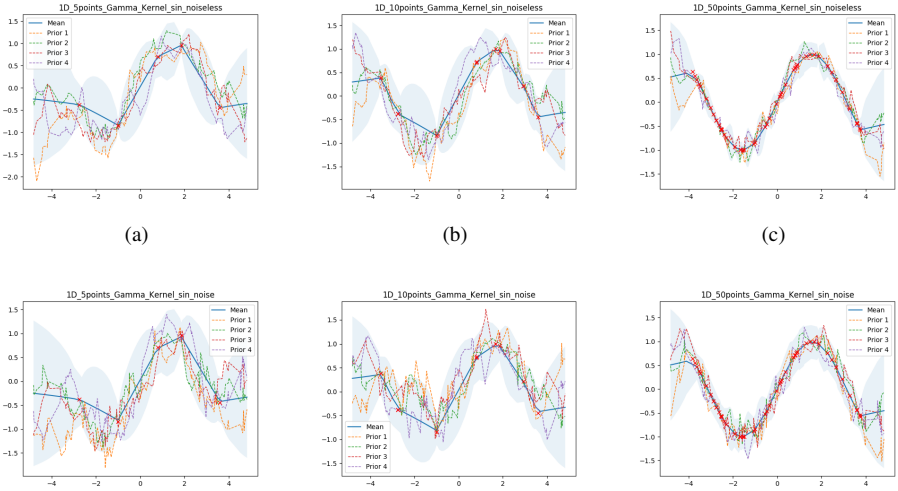
Fig. 9: Rational Kernel ($\alpha = 0.5$) for a sin function.



(a) (b) (c)

(d) (e) (f)

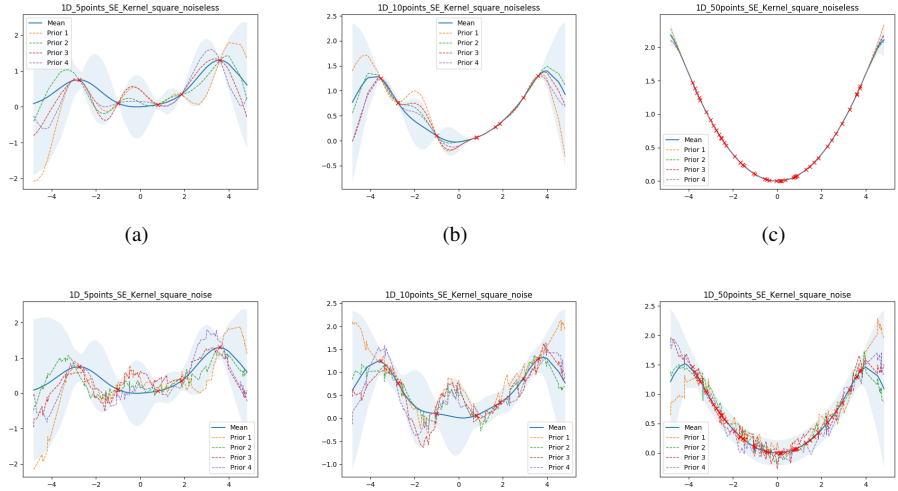
Fig. 10: Ornstein Kernel for a sin function.



(a) (b) (c)

(d) (e) (f)

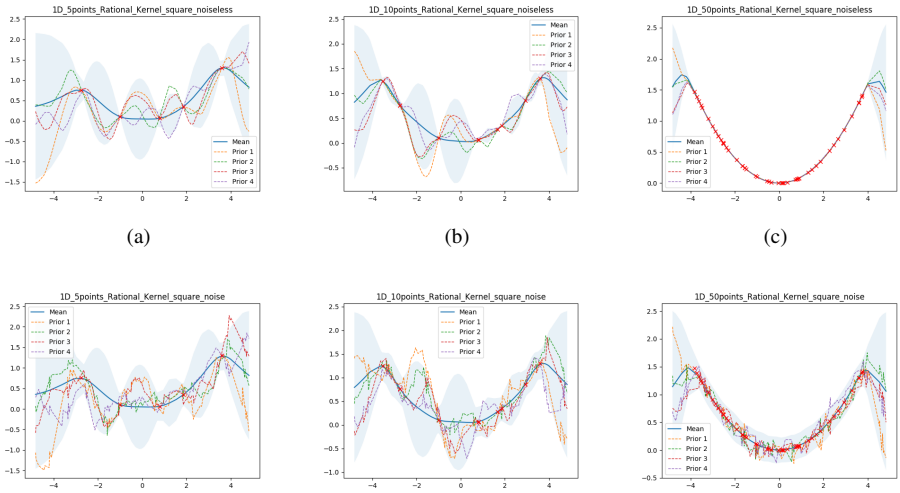
Fig. 11: Gamma Kernel ($\gamma = 0.2$) for a sin function.



(a) (b) (c)

(d) (e) (f)

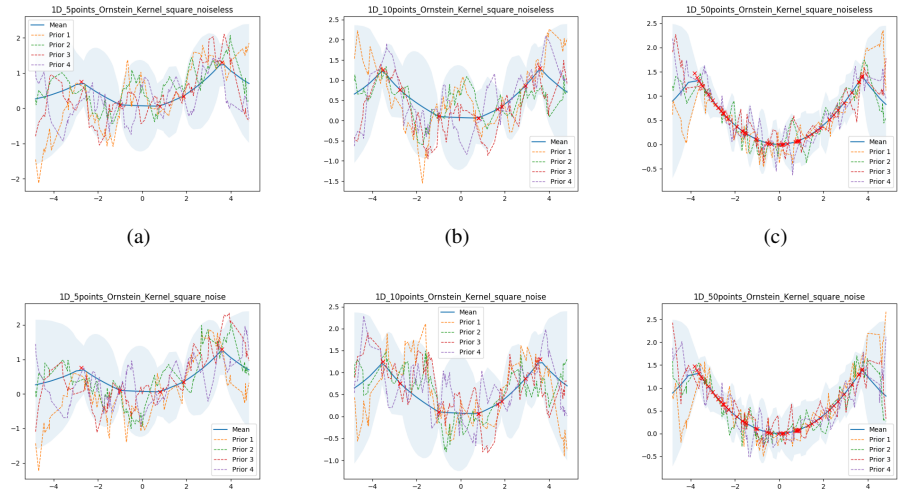
Fig. 12: Square Exponential Kernel for a square function.



(a) (b) (c)

(d) (e) (f)

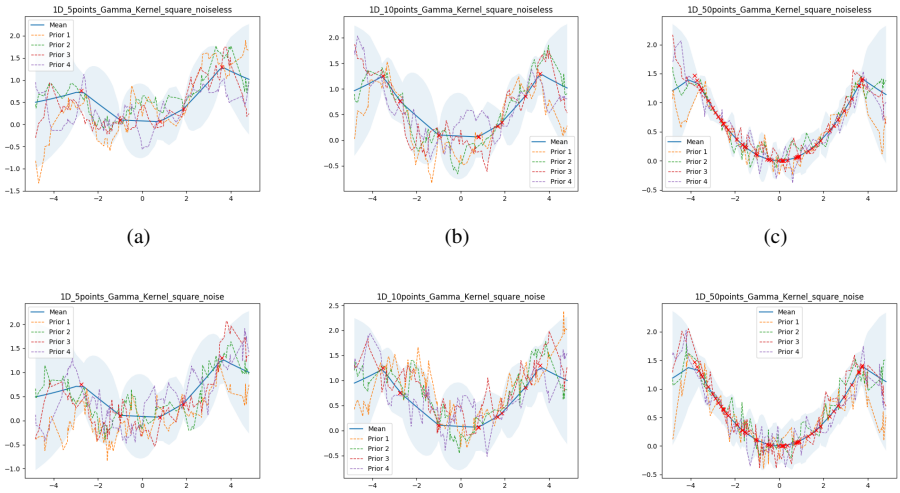
Fig. 13: Rational Kernel ($\alpha = 0.5$) for a square function.



(a) (b) (c)

(d) (e) (f)

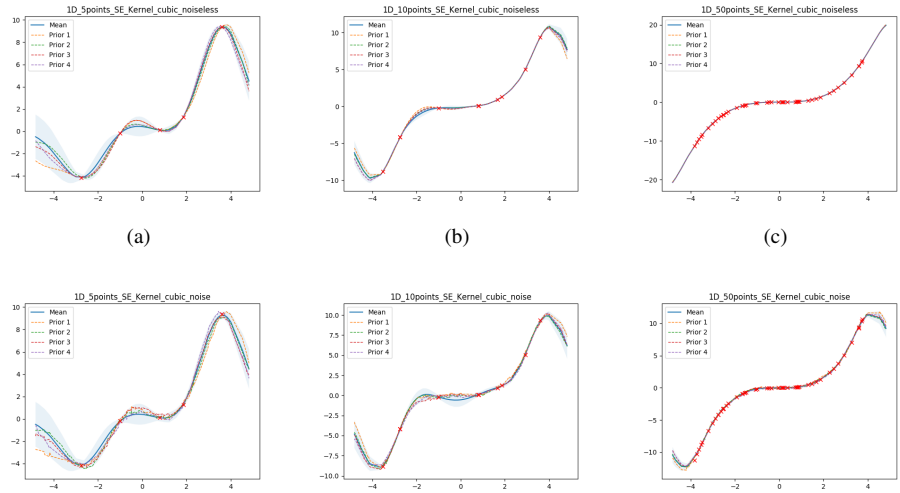
Fig. 14: Ornstein Kernel for a square function.



(a) (b) (c)

(d) (e) (f)

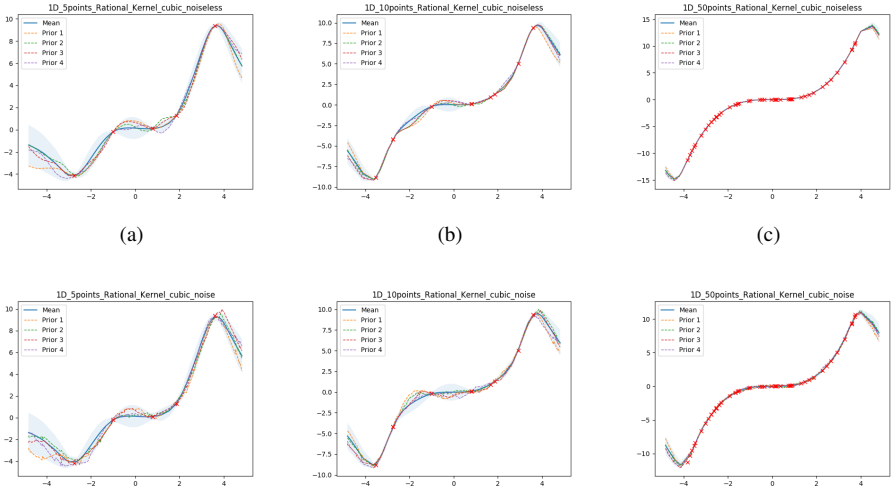
Fig. 15: Gamma Kernel ($\gamma = 0.2$) for a square function.



(a) (b) (c)

(d) (e) (f)

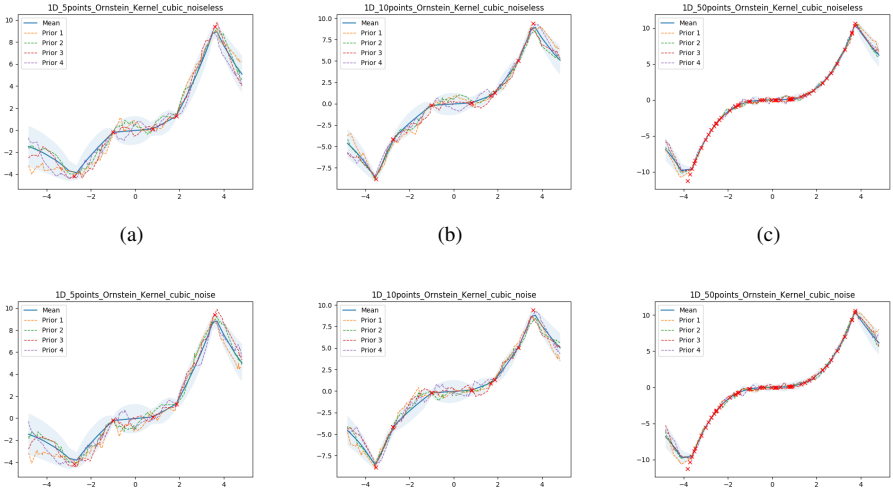
Fig. 16: Square Exponential Kernel for a cubic function.



(a) (b) (c)

(d) (e) (f)

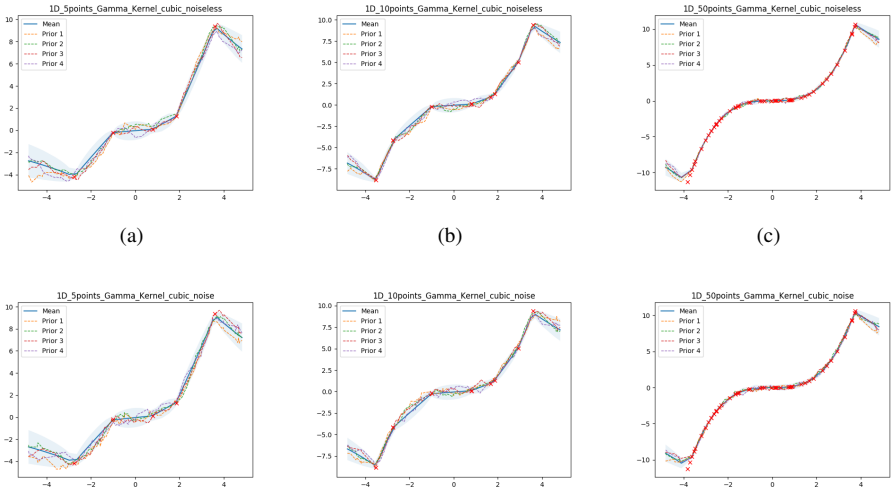
Fig. 17: Rational Kernel ($\alpha = 0.5$) for a cubic function.



(a) (b) (c)

(d) (e) (f)

Fig. 18: Ornstein Kernel for a cubic function.

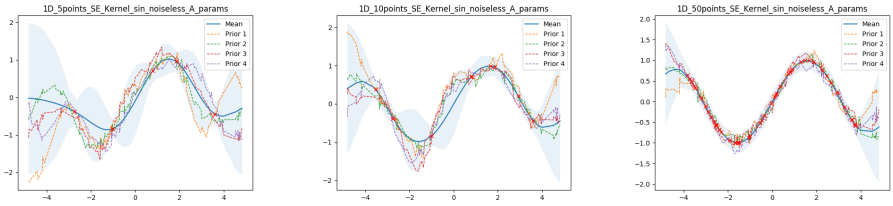


(a) (b) (c)

(d) (e) (f)

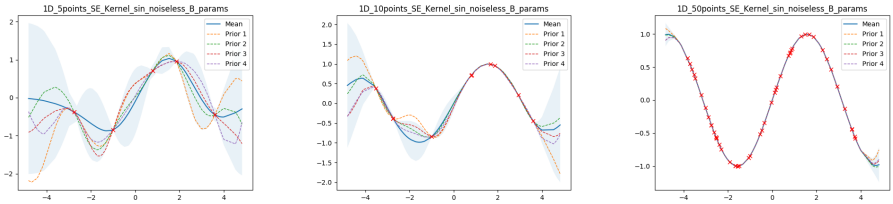
Fig. 19: Gamma Kernel ($\gamma = 0.2$) for a cubic function.

APPENDIX C
EXPERIMENTING IN 1 DIMENSION - CHANGING PARAMETERS ELL AND S



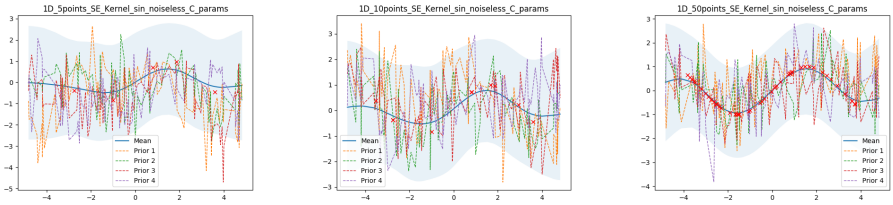
(a) (b) (c)

Fig. 20: Square Exponential Kernel accodding to (a) of table I



(a) (b) (c)

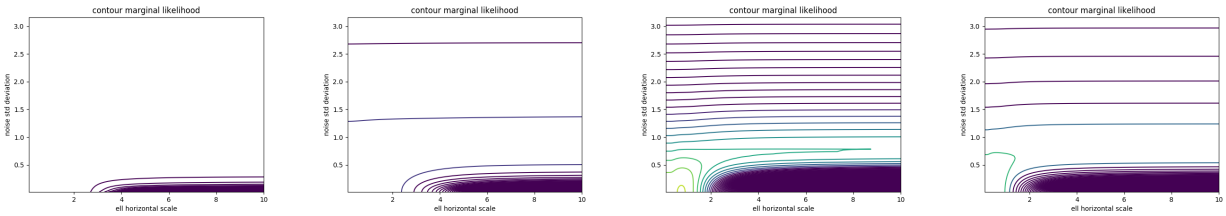
Fig. 21: Square Exponential Kernel accodding to (b) of table I



(a) (b) (c)

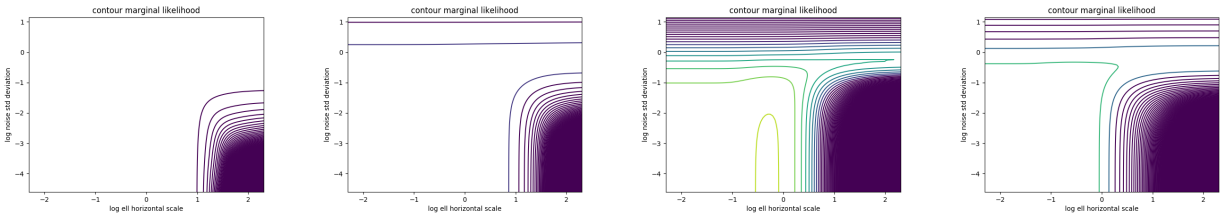
Fig. 22: Square Exponential Kernel accodding to (c) of table I

APPENDIX D
LIKELIHOOD CONTOUR PLOTS



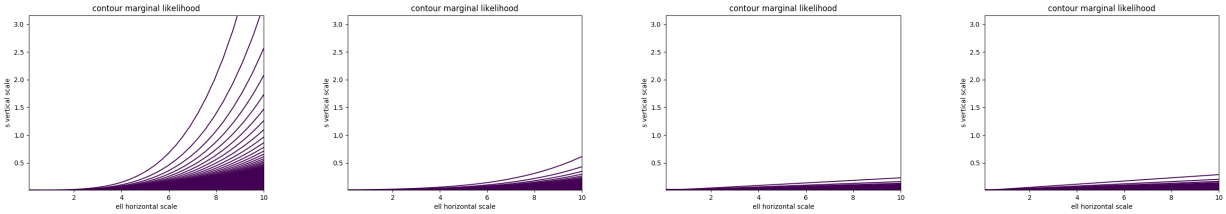
(a) SE Kernel (b) Rational Kernel (c) Ornstein Kernel (d) Gamma Kernel

Fig. 23: Likelihood contour of horizontal scale vs noise.



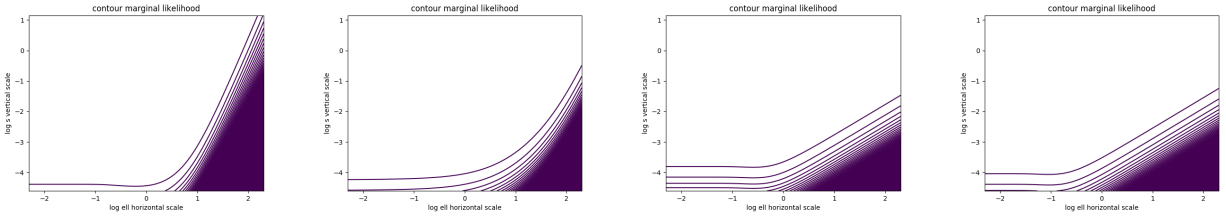
(a) SE Kernel (b) Rational Kernel (c) Ornstein Kernel (d) Gamma Kernel

Fig. 24: Likelihood contour of horizontal scale log ell vs log noise.



(a) SE Kernel (b) Rational Kernel (c) Ornstein Kernel (d) Gamma Kernel

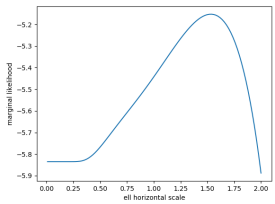
Fig. 25: Likelihood contour of horizontal scale ell vs vertical scale σ.



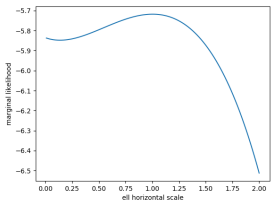
(a) SE Kernel (b) Rational Kernel (c) Ornstein Kernel (d) Gamma Kernel

Fig. 26: Likelihood contour of horizontal scale log ell vs vertical scale log σ.

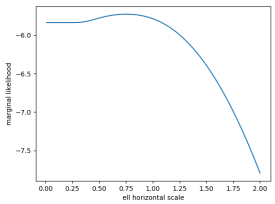
APPENDIX E
LIKELIHOOD



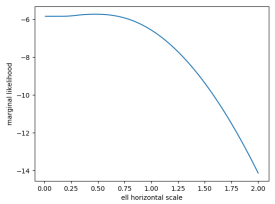
(a) SE Kernel



(b) Rational Kernel

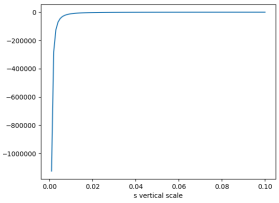


(c) Ornstein Kernel

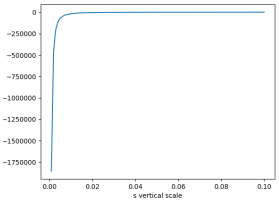


(d) Gamma Kernel

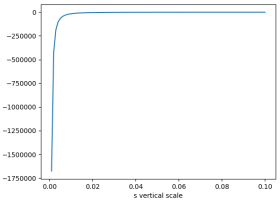
Fig. 27: Likelihood vs horizontal scale ell.



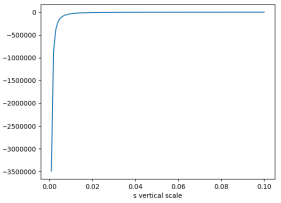
(a) SE Kernel



(b) Rational Kernel

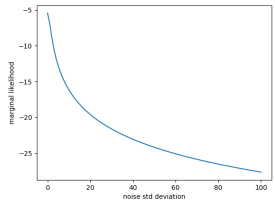


(c) Ornstein Kernel

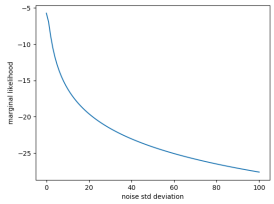


(d) Gamma Kernel

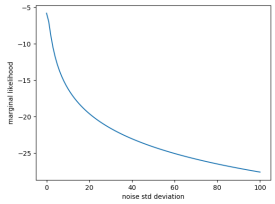
Fig. 28: Likelihood vs vertical scale σ.



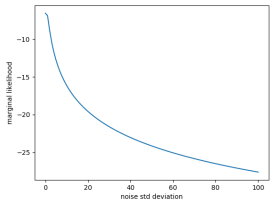
(a) SE Kernel



(b) Rational Kernel



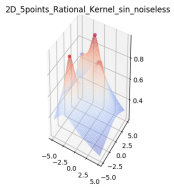
(c) Ornstein Kernel



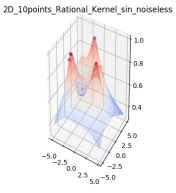
(d) Gamma Kernel

Fig. 29: Likelihood vs noise

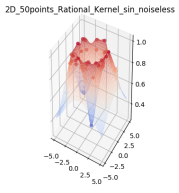
APPENDIX F
2D WITH SIN



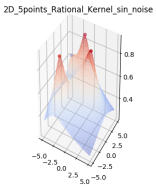
(a)



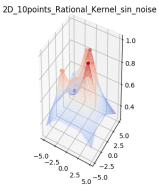
(b)



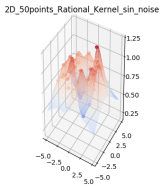
(c)



(d)

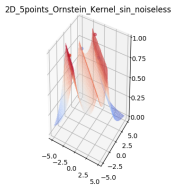


(e)

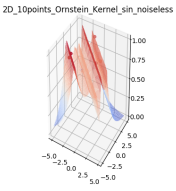


(f)

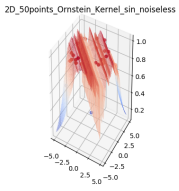
Fig. 30: Rational Kernel ($\alpha = 0.5$) for a sin function.



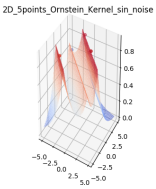
(a)



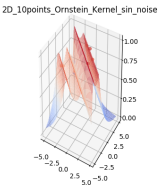
(b)



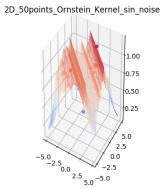
(c)



(d)



(e)



(f)

Fig. 31: Ornstein Kernel for a sin function.

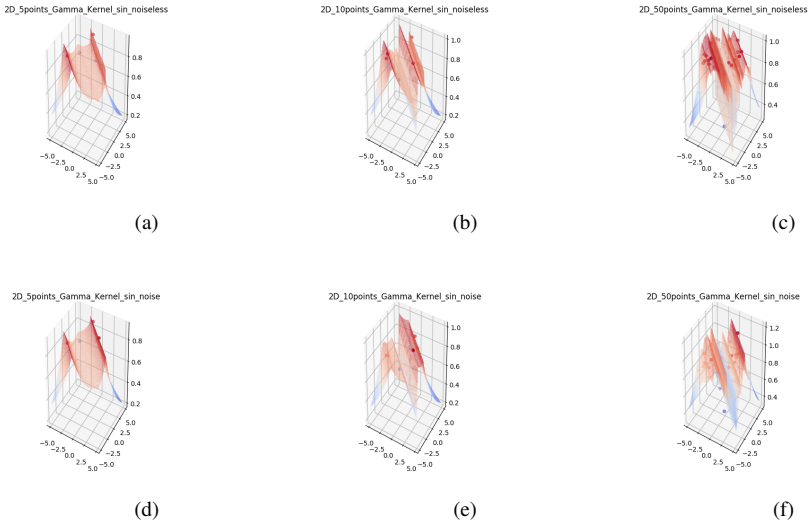
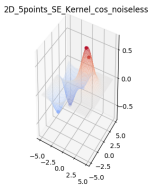
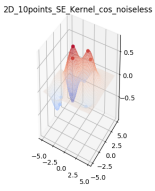


Fig. 32: Gamma Kernel ($\gamma = 0.2$) for a sin function.

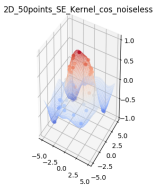
APPENDIX G
2D WITH COS



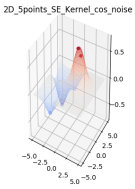
(a)



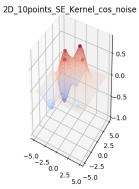
(b)



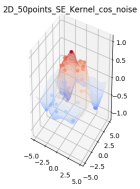
(c)



(d)

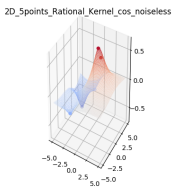


(e)

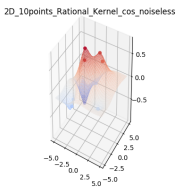


(f)

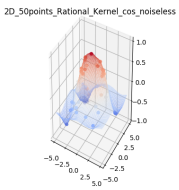
Fig. 33: Square Exponential Kernel for a cos function.



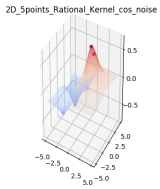
(a)



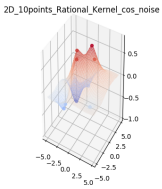
(b)



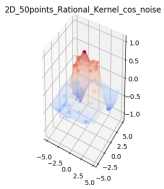
(c)



(d)

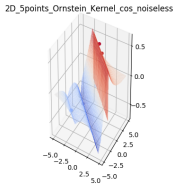


(e)

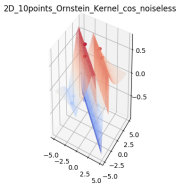


(f)

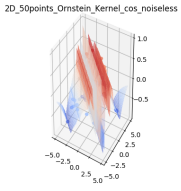
Fig. 34: Rational Kernel ($\alpha = 0.5$) for a cos function.



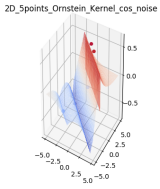
(a)



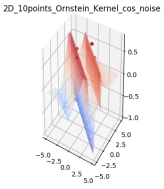
(b)



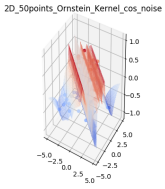
(c)



(d)

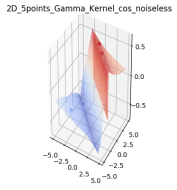


(e)

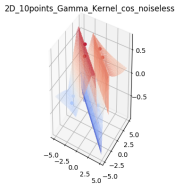


(f)

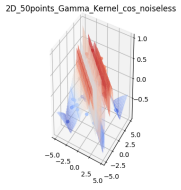
Fig. 35: Ornstein Kernel for a cos function.



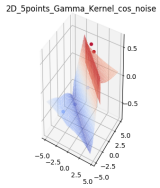
(a)



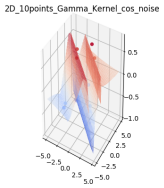
(b)



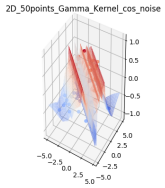
(c)



(d)



(e)



(f)

Fig. 36: Gamma Kernel ($\gamma = 0.2$) for a cos function.