Issues on positive semi-definiteness of covariance matrices in Gaussian Processes

Sota Yoshida[1, 2, *]
[1] *Department of Physics, the University of Tokyo, Hongo, Bunkyo-ku, Tokyo 113-0033, Japan*
[2] *Liberal and General Education Center, Institute for Promotion of Higher Academic Education, Utsunomiya University, Mine, Utsunomiya, 321-8505, Japan*

(Dated: December 13, 2019)

During discussions with the referees on our article [1], one question come up with our mind regarding the numerical treatment of positive semi-definiteness of covariance matrices in Gaussian Processes. We summarize them in this document. The famous packages GPy (Python) and GaussianProcesses.jl (Julia) are also mentioned. The codes used for this article are available on the author's GitHub page [2]. We would be happy if this could be helpful for someone using GPs.

We, practitioners of Gaussian processes, usually need to calculate conditional mean vectors and covariance matrices for predictions $\{\boldsymbol{y}^*\}$ under the given observations $\{\boldsymbol{y}\}$:

$$P(\boldsymbol{y}^*|\boldsymbol{y},\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{y}^*|\boldsymbol{y}}, \Sigma_{\boldsymbol{y}^*|\boldsymbol{y}}), \tag{1}$$

$$\boldsymbol{\mu}_{\boldsymbol{y}^*|\boldsymbol{y}}(\boldsymbol{\theta}) = \boldsymbol{\mu}^* + K^T_{XX^*}K^{-1}_{XX}(\boldsymbol{y}-\boldsymbol{\mu}), \tag{2}$$

$$\Sigma_{\boldsymbol{y}^*|\boldsymbol{y}}(\boldsymbol{\theta}) = K_{X^*X^*} - K^T_{XX^*}K^{-1}_{XX}K_{XX^*}, \tag{3}$$

where the $\boldsymbol{\theta}$ is hyperparameter for the kernel function, X and X^* are the set of observation points and prediction points. The positive semi-definiteness is required to achieve the Cholesky decomposition to evaluate K^{-1}_{XX} and to make samples from the multivariate normal distribution $\mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{y}^*|\boldsymbol{y}}, \Sigma_{\boldsymbol{y}^*|\boldsymbol{y}})$.

In some cases (e.g., points are located too close to each other), the covariance in Eq. (3) become non-PSD numerically, although it must be PSD mathematically. The typical prescription to this is to add infinitesimal diagonal matrix to the K_{XX}. Let us call this *the ϵ prescription* in the followings. For example, in the package GaussianProcesses.jl for Julia, the [make_posdef!] function do this.

The primary motivation of this survey is based on our experience: RBF kernel tends to have singular covariance (i.e. not positive semi-definite), when we write the GP code by ourselves. Then, we thought "Why do many people use RBF kernel in their papers without any problem?"

Here are toy two data sets to be considered:

- set A:
  $X = (6.0, 8.0)$, $Y = (-1.0, 1.0)$, $X^* = (10.0, 12.0)$
- set B:
  $X = (6.0, 8.0, 10.0, 12.0, 14.0)$, $Y = (-28.602, -30.213, -31.176, -31.713, -31.977)$, $X^* = (16.0, 18.0, 20.0)$

Who may give the *correct* answer?: set A

Before discussing PSD properties of covariance matrices, we would like to make it clear that the problem below is not due to the our implementation of GPs. The set A is one for that purpose. In this case, Eq. (2) and Eq. (3) can be analytically calculated under the given hyperparameters.

In the following, we use the four codes listed below:

A [python_gp_own.py]: own code in Python,

B [julia_gp_own.jl]: own code in Julia,

C [library_gp.py]: using the GPy package for Python,

D [julia_gp_library.jl]: using the GaussianProcesses.jl package for Julia.

All are available on GitHub [2].

For the simplicity of discussion, we restrict ourselves to use the RBF kernel:

$$k_{\mathrm{RBF}}(x_i, x_j) = \tau \exp\left(-\frac{|x_i - x_j|^2}{2\sigma^2}\right), \tag{4}$$

TABLE I. The summary of outputs of the four codes.

	A (own Python)	B (own Julia)	C (GPy)	D (Gaussian Processes.jl)
$\boldsymbol{\mu}_{\boldsymbol{y}^*\|\boldsymbol{y}}$	[0.15613, 0.00038795]	[0.15613, 0.00038795]	[0.07240, 0.00017990]	[0.15613, 0.00038795]
Diag($\Sigma_{\boldsymbol{y}^*\|\boldsymbol{y}}$)	[0.981355, 0.99999989]	[0.981355, 0.99999989]	[0.990803, 0.99999994]	[0.981355, 0.99999989]

and we fixed the hyperparameters $\tau = 1.0$ and $\sigma = 1.0$. Since the data values have zero mean, results is not unexpectedly influenced by a normalization function in the libraries.

The results are summarized in Table I. While the codes A, B and D gives the same results, C does not give the same. In the GPy package, the size of observation noise is fixed $\sigma = 1.0$ as default. This is unrealistically large noise especially when the data is normalized. On the other hand, in the GaussianProcesses.jl the default size of observation noise is -2.0 in logarithmic scale. We again set this observation noise as almost zero in the code D. We note that the default [println] function in Julia tends to round the float numbers in arrays. If one wants to compare values in high accuracy, one has to make sure to print each element of array or to use the [sprintf] function.

In this case of set A, we can calculate the mean vector Eq. (2) and Eq. (3) analytically:

$$\mu(x^* = 10) = 1 - \frac{1 - e^{-8} + e^{-12}}{e^4 - 1} \simeq 0.9813550426957166, \tag{5}$$

$$\mu(x^* = 12) = 1 - \frac{e^{-12} - 2e^{-24} + e^{-32}}{e^4 - 1} \simeq 0.9999998853666244. \tag{6}$$

Now the answer to the title of this section is at least our implementation and the GaussianProcesses.jl package for Julia.

We might have used GPy package incorrectly, but we have seen many sample codes draw credible intervals of the predictions as we did. We are not quite sure what the output of predict($\cdot$) and predict_noiseless($\cdot$) functions are. For this reason, we do not use GPy package in the followings.

At least, we are afraid that people sometimes use such an unrealistically large observation noise in their codes without noticing it.

How to deal with non-PSD covariance in the codes?: set B

Now we explore the PSD property of covariance matrices by using set B. Hereafter, we only use the codes written in Julia.

We use the following three Kernels. One is the RBF Kernel already used in the previous section.

$$k_{\mathrm{RBF}}(x_i, x_j) = \tau \exp\left(-\frac{|x_i - x_j|^2}{2\sigma^2}\right). \tag{7}$$

The other two are Matérn kernels with $\nu = 5/2$

$$k_{\mathrm{M}}(r \equiv |x_i - x_j|; \nu = 5/2) = \tau \left(1 + \frac{\sqrt{5}r}{\ell} + \frac{5r^2}{3\ell^2}\right) \exp\left(-\frac{\sqrt{5}r}{\ell}\right), \tag{8}$$

and with $\nu = 3/2$

$$k_{\mathrm{M}}(r \equiv |x_i - x_j|; \nu = 3/2) = \tau \left(1 + \frac{\sqrt{3}r}{\ell}\right) \exp\left(-\frac{\sqrt{3}r}{\ell}\right). \tag{9}$$

We calculated the conditional mean vectors and covariance matrices while varying the hyperparameters τ and σ using our own code in Julia. In Fig. 1, the positive semi-definiteness of the covariance matrices are summarized. The diamond symbols (red) correspond to the case that both K_{XX} and $\Sigma_{\boldsymbol{y}^*|\boldsymbol{y}}$ are non-PSD, and the cross symbols (blue) are assigned if only the $\Sigma_{\boldsymbol{y}^*|\boldsymbol{y}}$ is non-PSD. This is the reason why the RBF relatively easily gives non-singular covariances in our applications. In terms of the length scale, the RBF gives non-PSD matrices easier than Matérn kernel with $\nu = 5/2$ by two orders of magnitude.

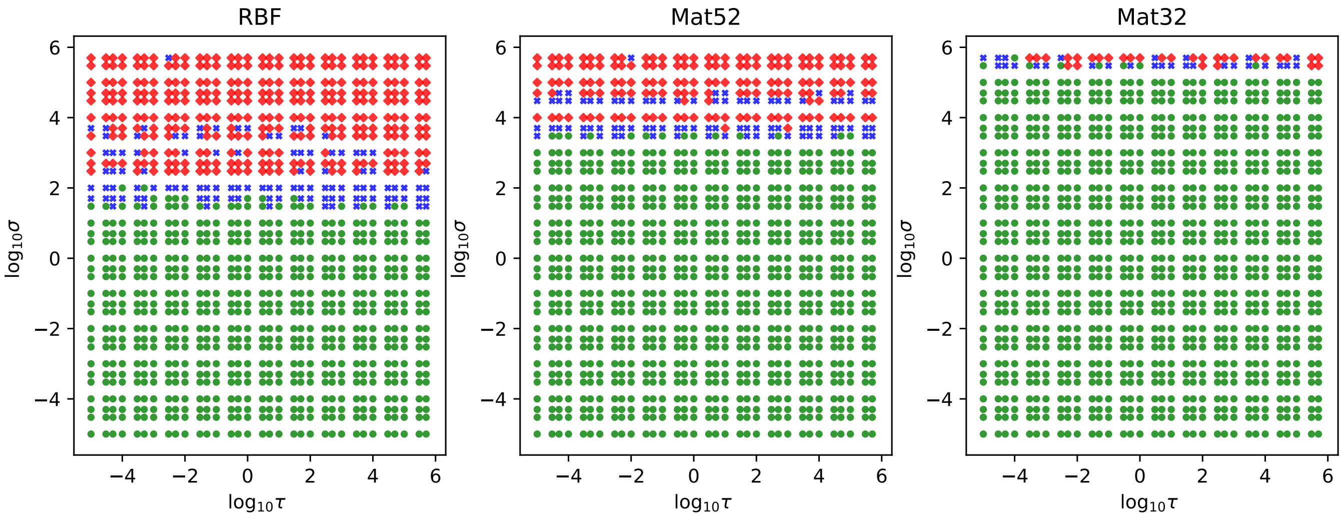

FIG. 1. The summary of PSD properties for the three Kernels. The covariance matrices having PSD are shown by the filled circle. The diamond symbols mean that both K_{XX} and $\Sigma_{\boldsymbol{y}^*|\boldsymbol{y}}$ are non-PSD, and the cross symbols are assigned if only the $\Sigma_{\boldsymbol{y}^*|\boldsymbol{y}}$ is non-PSD.

In the GaussianProcesses.jl package, the ϵ prescription is implemented as follows:

```julia
function make_posdef!(m::AbstractMatrix, chol_factors::AbstractMatrix)
    n = size(m, 1)
    size(m, 2) == n || throw(ArgumentError("Covariance matrix must be square"))
    for _ in 1:10 # 10 chances
        try
            # return m, cholesky(m)
            copyto!(chol_factors, m)
            chol = cholesky!(Symmetric(chol_factors, :U))
            return m, chol
        catch err
            if typeof(err)!=LinearAlgebra.PosDefException
                throw(err)
            end
            # that wasn't (numerically) positive definite,
            # so let's add some weight to the diagonal
            ϵ = 1e-6 * tr(m) / n
            @inbounds for i in 1:n
                m[i, i] += ϵ
            end
        end
    end
    copyto!(chol_factors, m)
    chol = cholesky!(Symmetric(chol_factors, :U))
    return m, chol
end
```

One can see that if the code fails to achieve the Cholesky decomposition of matrix K, this function do

$$K := K + \epsilon I_n, \tag{10}$$

$$\epsilon = 1.\mathrm{e} - 6 \times \mathrm{Tr}(K)/n. \tag{11}$$

As shown later, this procedure sometimes induces significant changes in resultant mean vectors.

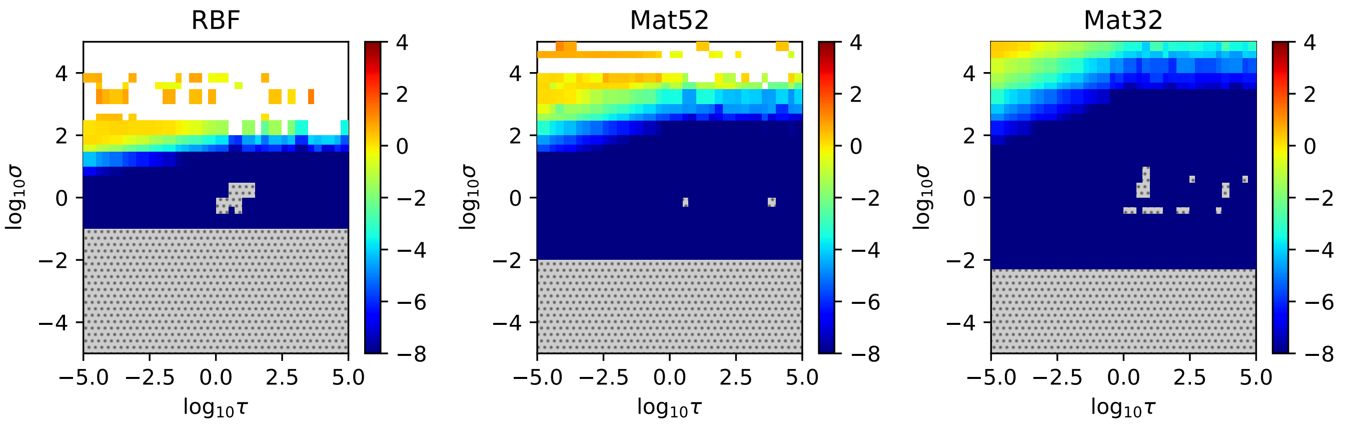

FIG. 2. The plot showing the impact of the ϵ prescription to avoid non-PSD. The colormap shows the Δ in Eq. (12). The hatched region (colored in gray) show the points at which the two codes gives the exactly the same result. The region where the self-made code does not give answer is colored in white.

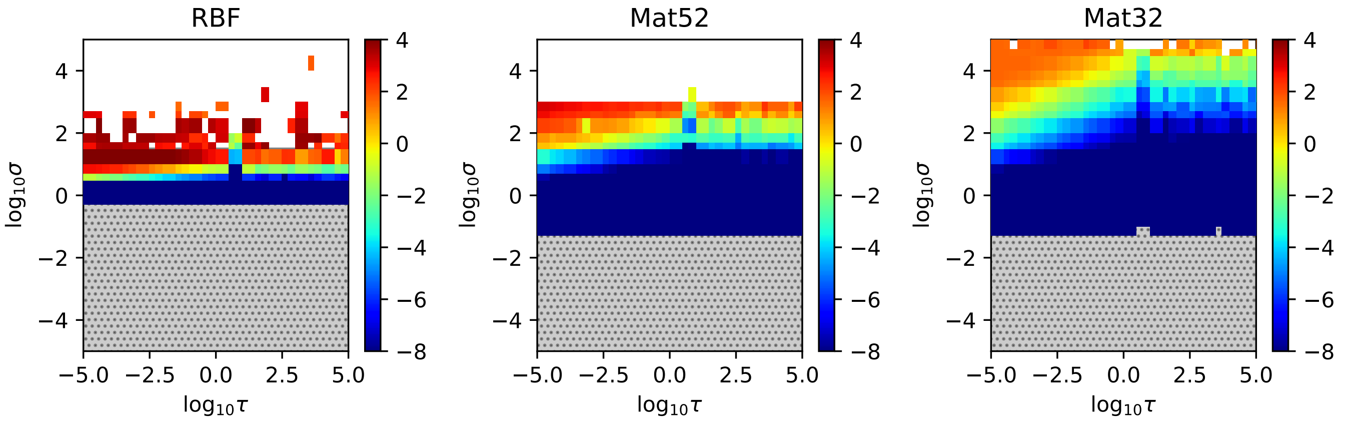

FIG. 3. The counterpart of FIG. 2 for the logarithm distance.

Now we show the effects of the ϵ prescription on the predictions. In FIG. 2 and FIG. 3, the following quantity is shown as the so-called heatmap:

$$\Delta \equiv \log_{10}\left(max(|\boldsymbol{\mu}^{\text{own}}_{\boldsymbol{y}^*|\boldsymbol{y}} - \boldsymbol{\mu}^{\text{lib.}}_{\boldsymbol{y}^*|\boldsymbol{y}}|)\right), \tag{12}$$

where the superscripts own and lib. mean the conditional mean vector calculated by our own code and one using the GaussianProcesses.jl library, respectively. FIG. 3 shows the Δ in the case of Kernels defined by the logarithm distance, i.e. $|x_i - x_j|$ is replaced by $|\ln x_i - \ln x_j|$.

The hatched region (colored in gray) at $\log_{10}\sigma < 1$ means that the results with the two codes are exactly the same. At the points shown by the diamond symbol in FIG. 1, our code without the ϵ prescription cannot not give the mean vectors. Those correspond to the white region $\log_{10}\sigma > 1.5$ appeared in FIG. 2 and FIG. 3. When we go to the parameters at which the posterior covariance become non-PSD, i.e. points shown by the cross symbol in FIG. 1, the Δ become large because of the *immune system* in the library. In some cases, the deviations in mean vectors reach $10^3 - 10^4$, which is obviously non-negligible. As immune systems in vivo, the ϵ prescription sometimes react too much.

One should pay much attention to how the PSD is treated in codes and whether or not one really can neglect the impact of the prescriptions on the predictions, especially when one would like to integrate out the hyperparameter dependence.

Note from physics aspect

In the current case, we assume the y and y^* values to be calculated energies of nuclei by *ab initio* full configuration interaction method. The typical convergence tolerance of the Lanczos method in shell-model codes on the market is 1.e-5 or better[1].

When one would like to incorporate this “tol" information into the kernel as the observation noise[2] like $K + \sigma^2_{\text{obs.}} I$, the ϵ in the package is obviously too large for the current purpose and this must be on the order of 1.e-10. As a result, the possible error coming from the ϵ prescription can be non-negligible as shown in FIG. 2 and FIG. 3.

Summary

We conclude that the Matérn kernel is safer to use than RBF in terms of positive semi-definiteness in many practical situations, and, the ϵ prescription with **any** Kernel function is **not always acceptable**. One should be careful to the prescription done in codes, especially when using some packages.

* s.yoshida@nt.phys.s.u-tokyo.ac.jp

[1] S. Yoshida, Non-parametric bayesian approach to extrapolation problems in configuration interaction methods (2019), arXiv:1907.04974.

[2] S. Yoshida, GitHub repository:IssueOnGPs`https://github.com/SotaYoshida/`.

[3] C. W. Johnson, W. E. Ormand, K. S. McElvain, and H. Shan, Bigstick: A flexible configuration-interaction shell-model code (2018), arXiv:1801.08432.

[4] N. Shimizu, Nuclear shell-model code for massive parallel computation, "kshell" (2013), arXiv:1310.5431.

[5] N. Shimizu, T. Mizusaki, Y. Utsuno, and Y. Tsunoda, Computer Physics Communications **244**, 372 (2019).

[1] In the BIGSTICK [3] and the KSHELL [4, 5], tol = 1.e-5 as default.

[2] In this context, the term *observation* means doing calculation instead of experimental observation.