Draft version January 25, 2025
Typeset using LaTeX default style in AASTeX631

On the Gaussian Process using JAX/NumPyro

Hajime Kawahara [1,2]

[1] Department of Space Astronomy and Astrophysics, ISAS/JAXA, 3-1-1, Yoshinodai, Sagamihara, Kanagawa, 252-5210 Japan
[2] Department of Astronomy, Graduate School of Science, The University of Tokyo, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113-0033, Japan

ABSTRACT

In this note, we consider the Gaussian process using JAX/NumPyro.

Keywords: JAX – NumPyro

1. GAUSSIAN PROCESS AS A STOCHASTIC MODEL

A Gaussian process is a stochastic process for a random variable $\boldsymbol{x}$ that follows a multivariate normal distribution,

$$\boldsymbol{d} \sim \mathcal{N}(\mathbf{0}, \Sigma), \tag{1}$$

where each component d_i is a time series. The covariance matrix of this multivariate normal distribution is defined as

$$\Sigma_{ij} = K_{ij}(a, \tau) = ak(|t_i - t_j|; \tau) \tag{2}$$

a function of the absolute difference between t_i and t_j. This formulation yields a Gaussian process with a correlation length τ. Various kernel functions $k(t, \tau)$ can be used. For example, the RBF kernel is given by

$$k_{\rm RBF}(t; \tau) = \exp\left(-\frac{t^2}{2\tau^2}\right), \tag{3}$$

and the Matérn 3/2 kernel is defined as

$$k_{\rm M3/2}(t; \tau) = \left(1 + \frac{\sqrt{3}t}{\tau}\right) e^{-\sqrt{3}t/\tau}. \tag{4}$$

Let us implement these kernels and generate some example data. The kernels can be coded in Python as follows:

```python
import numpy as np
def RBF(t, tau):
    Dt = t - np.array([t]).T
    K = np.exp(-(Dt)**2 / 2 / (tau**2))
    return K

def Matern32(t, tau):
    Dt = t - np.array([t]).T
    fac = np.sqrt(3.0) * np.abs(Dt) / tau
    K = (1.0 + fac) * np.exp(-fac)
    return K
```

kawahara@ir.isas.jaxa.jp

Next, we use the ‘scipy.stats.multivariate_normal‘ module to sample from a multivariate normal distribution. The sampling code is as follows:

```
from scipy.stats import multivariate\_normal as smn
np.random.seed(seed=1)  # Set random seed
N = 101
t = np.linspace(0, 10, N)
ave = np.zeros(N)
tau = 0.4
a = 1.0
cov = a * RBF(t, tau)  # Alternatively, use Matern32(t, tau)
di = smn(mean=ave, cov=cov, allow\_singular=True).rvs(1).T
```

To simulate observational noise, we add Gaussian noise with zero mean and a standard deviation σ:

```
sigma = 0.6
d = di + np.random.normal(0.0, sigma, len(di))
```

The generated data, shown in Figure 1, includes the original data without observational noise (solid line).

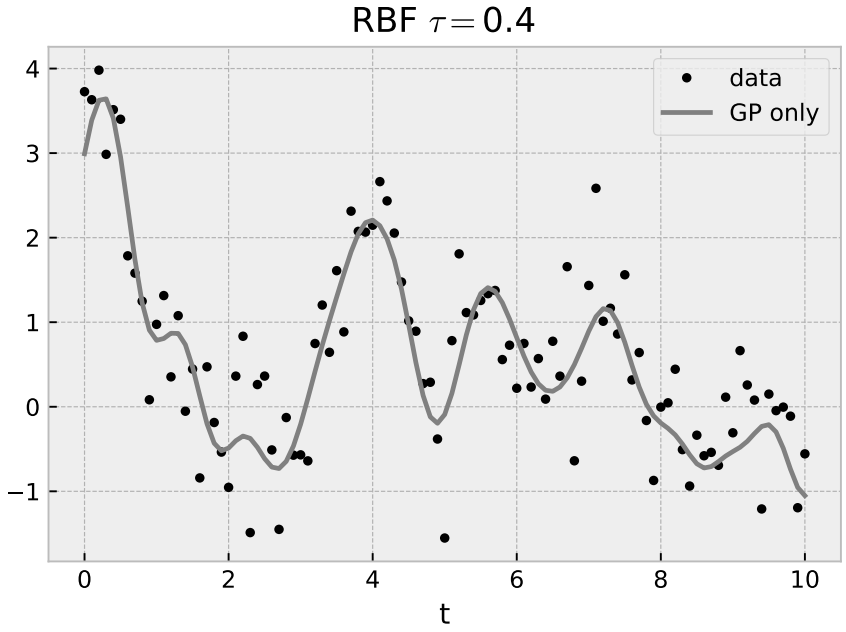

Figure 1. The solid line represents data without observational noise.

Now, we estimate the parameters τ, a, and σ from this data. With Gaussian noise added to the observations, the probabilistic model is expressed as:

$$\boldsymbol{d} \sim \mathcal{N}(\mathbf{0}, \Sigma') \tag{5}$$

$$\Sigma' = K(a, \tau) + \sigma^2 I \tag{6}$$

The following implementation computes Σ':

```
def modelcov(t, tau, a, sigma):
    Dt = t - jnp.array([t]).T
```

```
    K = a * jnp.exp(-(Dt)**2 / 2 / (tau**2)) + jnp.eye(N) * sigma**2
    return K
```

The numpyro model for the Gaussian process using the modelcov function is defined as follows:

```
import jax.numpy as jnp
import numpyro
import numpyro.distributions as dist

def model(t, y):
    sigma = numpyro.sample('sigma', dist.Exponential(1.))
    tau = numpyro.sample('tau', dist.Exponential(1.))
    a = numpyro.sample('a', dist.Exponential(1.))
    cov = modelcov(t, tau, a, sigma)
    numpyro.sample('y', dist.MultivariateNormal(loc=jnp.zeros(N),
    covariance_matrix=cov), obs=y)
```

Here, the prior distributions for τ and σ are set to exponential distributions. To perform Hamiltonian Monte Carlo (HMC) sampling using NUTS in numpyro, we proceed as follows:

```
from numpyro.infer import MCMC, NUTS
from jax import random
rng_key = random.PRNGKey(0)
rng_key, rng_key_ = random.split(rng_key)
num_warmup, num_samples = 1000, 2000
# Run NUTS.
kernel = NUTS(model)
mcmc = MCMC(kernel, num_warmup, num_samples)
mcmc.run(rng_key_, t=t, y=d)
```

The posterior distributions can be visualized using arviz, as shown below:

```
import arviz
refs = {}; refs["sigma"] = sigma; refs["tau"] = tau; refs["a"] = a
arviz.plot_pair(arviz.from_numpyro(mcmc), kind='kde',
    divergences=False, marginals=True, reference_values=refs,
    reference_values_kwargs={'color':"red", "marker":"o", "markersize":12})
```

The resulting visualization of the posterior distributions is shown in Figure 2.

1.1. *Predictive Distribution and Credible Intervals*

The model defined in Equation (5) assumes a mean of $\boldsymbol{x} = \mathbf{0}$. This can be further elucidated by computing the credible intervals. Using the predictive posterior samples, the Highest Posterior Density Interval (HPDI) can be computed as follows:

```
from numpyro.infer import Predictive
from numpyro.diagnostics import hpdi
# Extract posterior samples
posterior_a = mcmc.get_samples()['a']
posterior_tau = mcmc.get_samples()['tau']
posterior_sigma = mcmc.get_samples()['sigma']
```

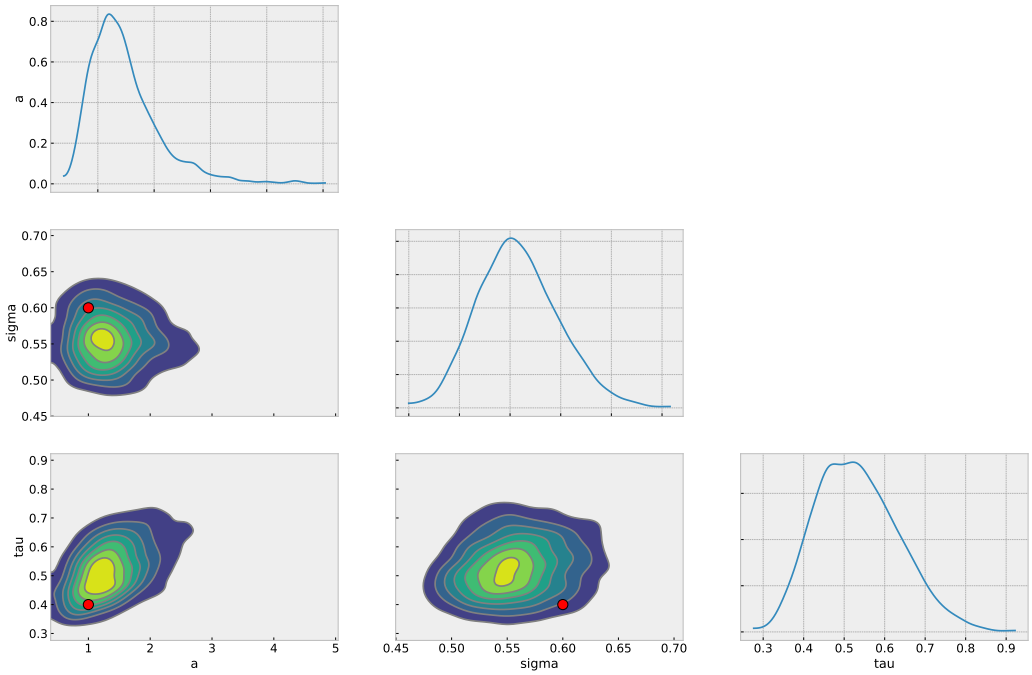

Figure 2.

```
# Predictive model
pred = Predictive(model, {'a': posterior_a, 'tau': posterior_tau,
       'sigma': posterior_sigma}, return_sites=["y"])
predictions = pred(rng_key_, t=t, y=None)
mean_muy = jnp.mean(predictions["y"], axis=0)
hpdi_muy = hpdi(predictions["y"], 0.9)
```

The predicted mean and HPDI can then be plotted as follows:

```
fig, ax = plt.subplots(nrows=1, ncols=1, figsize=(6, 3))
ax.plot(t, d, ".", color="black")
ax.plot(t, mean_muy, color="C0")
ax.fill_between(t, hpdi_muy[0], hpdi_muy[1], alpha=0.3, interpolate=True, color="C0")
plt.xlabel("t")
plt.ylabel("y")
```

The resulting plot, shown in Figure 3, provides an intuitive visualization of the data and the credible intervals.

1.2. *Gaussian Processes as Priors for Model Parameters*

The Gaussian process described above can also be used as a prior distribution for model parameters $\boldsymbol{m}$:

$$p(\boldsymbol{m}) = \mathcal{N}(\mathbf{0}, \Sigma) \tag{7}$$

Observations d_i can be expressed as:

$$d_i = m_i + \epsilon \tag{8}$$

$$\epsilon \sim \mathcal{N}(0, \sigma^2) \tag{9}$$

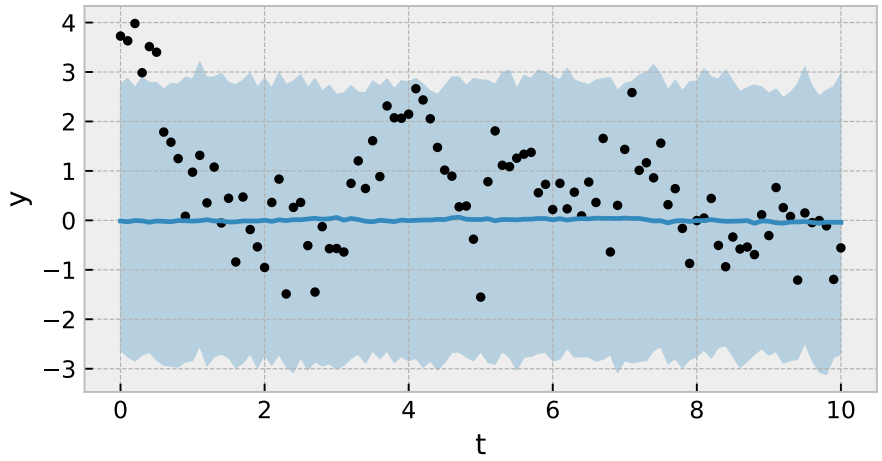

Figure 3.

where ϵ corresponds to observational noise. Alternatively, using the identity transformation for the model $\boldsymbol{g}$, this can be written as:

$$\boldsymbol{g}(\boldsymbol{m}) = \boldsymbol{m} \tag{10}$$

$$\boldsymbol{d} = \boldsymbol{g}(\boldsymbol{m}) + \boldsymbol{\epsilon} \tag{11}$$

$$\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2 I) \tag{12}$$

The likelihood function becomes:

$$p(\boldsymbol{d}|\boldsymbol{m}) = \mathcal{N}(\boldsymbol{d} - \boldsymbol{g}(\boldsymbol{m}), \sigma^2 I) = \mathcal{N}(\boldsymbol{d} - \boldsymbol{m}, \sigma^2 I) \tag{13}$$

The covariance matrix Σ is parameterized as:

$$\Sigma_{ij} = ak(|t_i - t_j|; \tau) \tag{14}$$

where a and τ are hyperparameters. These hyperparameters are assigned prior distributions (hyperpriors). While the posterior distribution of $\boldsymbol{m}$ can be sampled directly using numpyro, it is also known that the posterior distribution can be derived analytically when the hyperparameters are fixed.

The exponent of a multivariate normal distribution can be expressed as:

$$-2 \log \mathcal{N}(\boldsymbol{m}|\boldsymbol{\mu}, \Sigma) = (\boldsymbol{m} - \boldsymbol{\mu})^\top \Sigma^{-1} (\boldsymbol{m} - \boldsymbol{\mu}) = \boldsymbol{m}^\top \Sigma^{-1} \boldsymbol{m} - 2\boldsymbol{m}^T \Sigma^{-1} \boldsymbol{\mu} + \text{const.} \tag{15}$$

If the density $p(\boldsymbol{m})$ is represented as:

$$-2 \log p(\boldsymbol{m}) = \boldsymbol{m}^\top P \boldsymbol{m} - 2\boldsymbol{m}^\top \boldsymbol{q} + \text{const}, \tag{16}$$

then comparing this to the multivariate normal distribution yields:

$$p(\boldsymbol{m}) = \mathcal{N}(\boldsymbol{m}|P^{-1}\boldsymbol{q}, P^{-1}). \tag{17}$$

Now, the posterior distribution is expressed as:

$$p(\boldsymbol{m}|\boldsymbol{d}) \propto p(\boldsymbol{d}|\boldsymbol{m}) p(\boldsymbol{m}) = \mathcal{N}(\boldsymbol{d} - \boldsymbol{m}, \sigma^2 I) \mathcal{N}(\boldsymbol{0}, \Sigma) \tag{18}$$

and by expanding the exponent terms with respect to $\boldsymbol{m}$, we can write:

$$-2\log\left[\mathcal{N}(\boldsymbol{d}-\boldsymbol{m},\sigma^2 I)\mathcal{N}(\boldsymbol{0},\Sigma)\right] = \boldsymbol{m}^\top(\Sigma^{-1}+\sigma^{-2}I)\boldsymbol{m} - 2\sigma^{-2}\boldsymbol{m}^\top\boldsymbol{d} + \text{const.} \tag{19}$$

Additionally, using the relationship:

$$(\Sigma^{-1}+\sigma^{-2}I)^{-1} = \Sigma(I+\sigma^{-2}\Sigma)^{-1} \tag{20}$$

we find:

$$p(\boldsymbol{m}|\boldsymbol{d}) = \mathcal{N}(\Sigma(\sigma^2 I+\Sigma)^{-1}\boldsymbol{d}, \Sigma(I+\sigma^{-2}\Sigma)^{-1}) \tag{21}$$

1.3. *Hyperparameter Sampling Using HMC-NUTS*

The earlier sampling of τ and σ via HMC-NUTS can now be interpreted in this framework. These parameters are hyperparameters, denoted as $\boldsymbol{\theta} = (a,\tau,\sigma)^\top$. Incorporating $\boldsymbol{\theta}$ into the probabilistic model, the marginalized likelihood over $\boldsymbol{m}$ becomes:

$$p(\boldsymbol{d}|\boldsymbol{\theta}) = \frac{p(\boldsymbol{d}|\boldsymbol{m},\boldsymbol{\theta})p(\boldsymbol{m},\boldsymbol{\theta})}{p(\boldsymbol{m}|\boldsymbol{d},\boldsymbol{\theta})} \tag{22}$$

By focusing on the terms involving $\boldsymbol{d}$ in the exponent:

$$-2\log p(\boldsymbol{d}|\boldsymbol{\theta}) = -2\log p(\boldsymbol{d}|\boldsymbol{m},\boldsymbol{\theta}) + 2\log p(\boldsymbol{m}|\boldsymbol{d},\boldsymbol{\theta}) + \text{const} \tag{23}$$

$$= -2\log\mathcal{N}(\boldsymbol{d}-\boldsymbol{m};\sigma^2 I) + 2\log\mathcal{N}(\Sigma(\sigma^2 I+\Sigma)^{-1}\boldsymbol{d}, \Sigma(I+\sigma^{-2}\Sigma)^{-1}) \tag{24}$$

$$= \boldsymbol{d}^\top(\sigma^2 I+\Sigma)^{-1}\boldsymbol{d} + \text{const.} \tag{25}$$

we obtain:

$$p(\boldsymbol{d}|\boldsymbol{\theta}) = \mathcal{N}(\boldsymbol{0}, \Sigma+\sigma^2 I) \tag{26}$$

This matches Equation (5) when $\Sigma = K(\tau)$. Thus, by providing a prior $p(\boldsymbol{\theta})$, the marginalized posterior distribution:

$$p(\boldsymbol{\theta}|\boldsymbol{d}) \propto p(\boldsymbol{d}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{27}$$

is sampled via HMC-NUTS. Figure 3 shows the marginalized posterior distribution of the hyperparameters τ and σ. That is, we sampled

$$\boldsymbol{\theta}_k^\dagger \sim p(\boldsymbol{\theta}|\boldsymbol{d}) \propto p(\boldsymbol{d}|\boldsymbol{\theta})p(\boldsymbol{\theta}). \tag{28}$$

From the joint posterior distribution:

$$p(\boldsymbol{m},\boldsymbol{\theta}|\boldsymbol{d}) = p(\boldsymbol{m}|\boldsymbol{\theta},\boldsymbol{d})p(\theta|\boldsymbol{d}) \tag{29}$$

and the posterior distribution of $\boldsymbol{m}$ given $\boldsymbol{\theta}_k^\dagger$:

$$p(\boldsymbol{m}|\boldsymbol{\theta}_k^\dagger,\boldsymbol{d}) = \mathcal{N}(\boldsymbol{\mu}_k, K_k) \tag{30}$$

$$\boldsymbol{\mu}_k = K(a_k^\dagger,\tau_k^\dagger)((\sigma_k^\dagger)^2 I + K(a_k^\dagger,\tau_k^\dagger))^{-1}\boldsymbol{d} \tag{31}$$

$$K_k = K(a_k^\dagger,\tau_k^\dagger)(I+(\sigma_k^\dagger)^{-2}K(a_k^\dagger,\tau_k^\dagger))^{-1} \tag{32}$$

we sample $(\boldsymbol{m}_k^\dagger,\boldsymbol{\theta}_k^\dagger)$ from $p(\boldsymbol{m},\boldsymbol{\theta}|\boldsymbol{d})$. The implementation for $\boldsymbol{\mu}_k$ and K_k is as follows:

```
def muGP(tau, a, sigma):
    cov = a * RBF(t, tau)
    IKw = sigma**2 * np.eye(N) + cov
    A = scipy.linalg.solve(IKw, d, assume_a="pos")
    return cov @ A

def covGP(tau, a, sigma):
    cov = a * RBF(t, tau)
    IKw = np.eye(N) + cov / sigma**2
    IKw = scipy.linalg.inv(IKw)
    return cov @ IKw
```

Sampling $\boldsymbol{m}$ using the posterior samples of $\boldsymbol{\theta}$ is done as follows:

```
import tqdm
Ns = len(posterior_sigma)
np.random.seed(seed=1)
marr = []
for i in tqdm.tqdm(range(0, Ns)):
    sigmas = float(posterior_sigma[i])
    taus = float(posterior_tau[i])
    a_s = float(posterior_a[i])
    ave = muGP(taus, a_s, sigmas)
    cov = covGP(taus, a_s, sigmas)
    mk = smn(mean=ave, cov=cov, allow_singular=True).rvs(1).T
    marr.append(mk)
marr = np.array(marr)
```

The credible interval for $\boldsymbol{m}$ is then computed as follows:

```
mean_muy = np.mean(marr, axis=0)
hpdi_muy = hpdi(marr, 0.9)
```

Plotting the results, as shown in Figure 4, reveals the 90% credible interval for the model parameter $\boldsymbol{m}$. Note that this interval is for $\boldsymbol{m}$, the model parameter, and does not include observational noise:

```
fig = plt.figure(figsize=(6, 3))
ax = fig.add_subplot(111)
ax.plot(t, d, ".", color="black")
ax.plot(t, mean_muy, color="C1")
ax.fill_between(t, hpdi_muy[0], hpdi_muy[1], alpha=0.3, interpolate=True, color="C1")
plt.xlabel("t")
plt.ylabel("y")
```

1.4. *Predictions at New Data Points*

To predict at $t = t^*$, the posterior distribution of $\boldsymbol{m}^*$ is:

$$p(\boldsymbol{m}^*|\boldsymbol{m}) = \mathcal{N}(K_\times^\top K^{-1}\boldsymbol{m}, K_* - K_\times^\top K^{-1} K_\times) \tag{33}$$

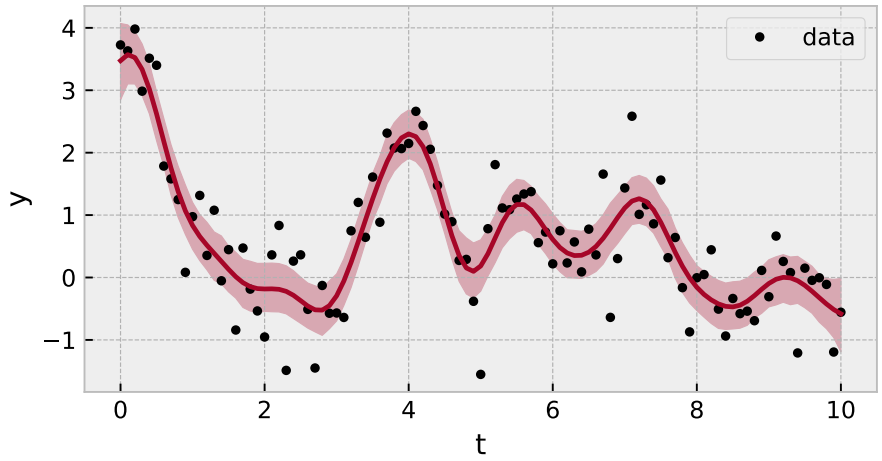

Figure 4.

where

$$K_{ij} = ak(|t_i - t_j|; \tau) \tag{34}$$

$$(K_\times)_{ij} = ak(|t_i - t_j^*|; \tau) \tag{35}$$

$$(K_*)_{ij} = ak(|t_i^* - t_j^*|; \tau). \tag{36}$$

Similarly,

$$p(\boldsymbol{m}^*|\boldsymbol{d}) = \mathcal{N}(K_\times^\top K_\sigma^{-1}\boldsymbol{d}, K_* - K_\times^\top K_\sigma^{-1} K_\times) \tag{37}$$

where

$$(K_\sigma)_{ij} = ak(|t_i - t_j|; \tau) + \sigma^2 \delta_{i,j} \tag{38}$$

If observational noise is included, as

$$\boldsymbol{d}^* = \boldsymbol{m}^* + \boldsymbol{\epsilon}, \tag{39}$$

the predictive distribution becomes:

$$p(\boldsymbol{d}^*|\boldsymbol{d}) = \mathcal{N}(K_\times^\top K_\sigma^{-1}\boldsymbol{d}, K_{*,\sigma} - K_\times^\top K_\sigma^{-1} K_\times) \tag{40}$$

where:

$$(K_{*,\sigma})_{ij} = ak(|t_i^* - t_j^*|; \tau) + \sigma^2 \delta_{i,j} \tag{41}$$

Sampling from the predictive distributions can be implemented as follows:

```
def mucovGPx(t, td, tau, a, sigma):
    cov = a * RBF(t, tau) + sigma**2 * np.eye(N)
    covx = a * RBFx(t, td, tau)
    covxx = a * RBF(td, tau)
    # For predictions including observational noise
    covxx += sigma**2 * np.eye(len(td))
    IKw = cov
    A = scipy.linalg.solve(IKw, d, assume_a="pos")
    IKw = scipy.linalg.inv(IKw)
    return covx @ A, covxx - covx @ IKw @ covx.T  # Mean and covariance
```

This enables predictions and credible intervals to be computed, as shown in Figure 5, with darker regions representing credible intervals for $\boldsymbol{m}^*$ and lighter regions for $\boldsymbol{d}^*$.

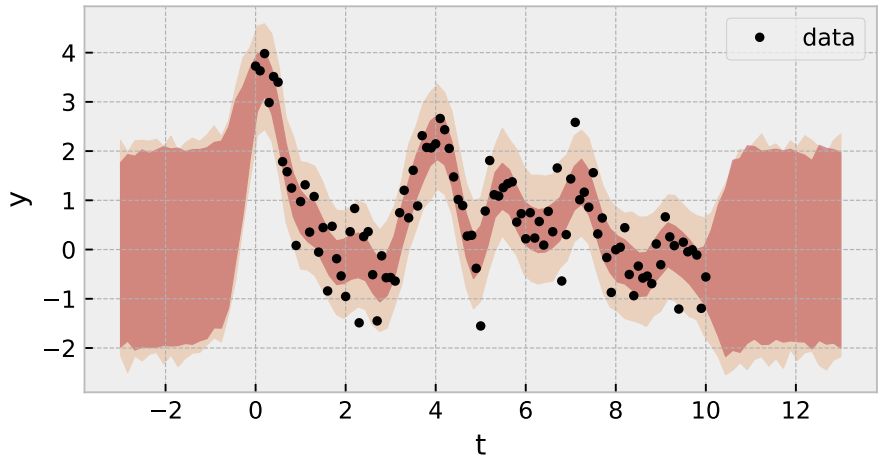

Figure 5.

2. MODEL FITTING WITH GAUSSIAN PROCESS NOISE

So far, we have considered a model generated by a Gaussian process with a zero mean:

$$\boldsymbol{m} \sim \mathcal{N}(\boldsymbol{0}, \Sigma(\boldsymbol{t})). \tag{42}$$

Here, we explore a case where the data is generated by a Gaussian process with a mean function dependent on t. Specifically, we consider:

$$\boldsymbol{m} \sim \mathcal{N}(\boldsymbol{f}(\boldsymbol{t}), \Sigma(\boldsymbol{t})), \tag{43}$$

where $\boldsymbol{f}(\boldsymbol{t})$ is a vectorized version of $f(t)$, applied to each element of $\boldsymbol{t} = (t_0, t_1, \cdots, t_{N-1})$. For $f(t)$, let us assume the following functional form:

$$f(t) = ke^{-(t-T_0)^2/2s^2} \sin(2\pi t/P). \tag{44}$$

While we write this as $f(t)$ for simplicity, we may also use the notation $f(t; \boldsymbol{\theta})$ to explicitly represent its dependence on the parameters $\boldsymbol{\theta} = (T_0, k, s, P)$.

Using the RBF kernel for the covariance matrix in Equation (43), we can generate data as shown in Figure 6. Here, $\tau = 3$, and we observe a relatively smooth trend overlaying $f(t)$.

Fitting such a model using HMC allows us to model noise correlated with the signal $f(t)$, including observational noise, and estimate the parameters of $f(t)$.

If $\boldsymbol{f}(\boldsymbol{t})$ is known, the predictions, both without and with observational noise, are given by:

$$p(\boldsymbol{m}^*|\boldsymbol{d}) = \mathcal{N}(\boldsymbol{f}(\boldsymbol{t}^*) + K_\times^\top K_\sigma^{-1}(\boldsymbol{d} - \boldsymbol{f}(\boldsymbol{t})), K_* - K_\times^\top K_\sigma^{-1} K_\times), \tag{45}$$

$$p(\boldsymbol{d}^*|\boldsymbol{d}) = \mathcal{N}(\boldsymbol{f}(\boldsymbol{t}^*) + K_\times^\top K_\sigma^{-1}(\boldsymbol{d} - \boldsymbol{f}(\boldsymbol{t})), K_{*,\sigma} - K_\times^\top K_\sigma^{-1} K_\times). \tag{46}$$

The parameters $\boldsymbol{\theta} = (T_0, k, s, P)$ of $\boldsymbol{f}(t)$ are sampled using HMC. For example, in the latter case, predictions can be obtained by sampling each $\boldsymbol{\theta}_k^\dagger$ as follows:

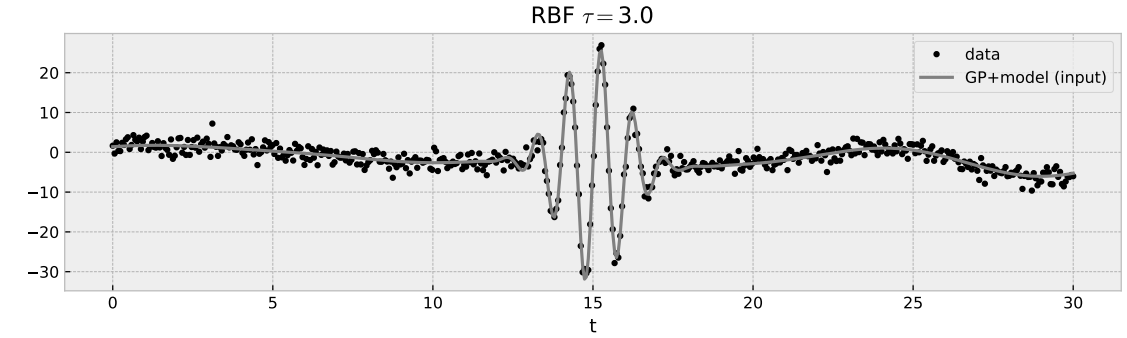

Figure 6. The solid line represents data without observational noise.

$$\boldsymbol{d}_k^* \sim \mathcal{N}(\boldsymbol{f}(\boldsymbol{t}^*;\boldsymbol{\theta}_k^\dagger) + K_\times^\top K_\sigma^{-1}(\boldsymbol{d} - \boldsymbol{f}(\boldsymbol{t};\boldsymbol{\theta}_k^\dagger)), K_{*,\sigma} - K_\times^\top K_\sigma^{-1} K_\times). \tag{47}$$

Thus, we can perform Gaussian process fitting that incorporates the model $f(t)$, as shown in Figure 7.

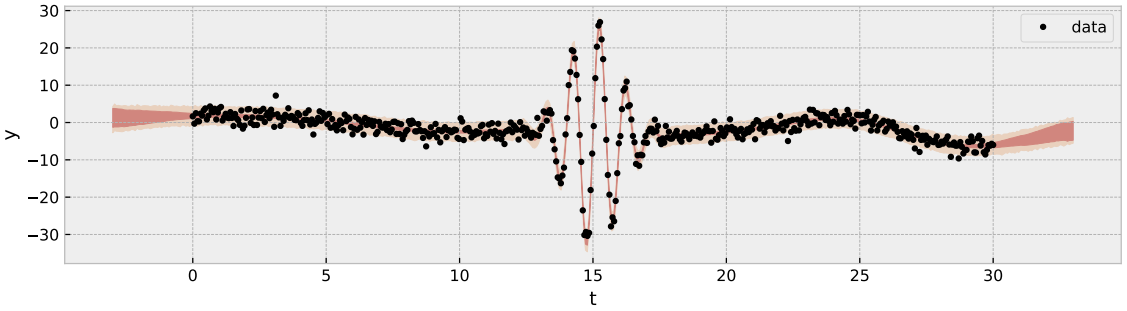

Figure 7.

Ref: Appendix in Kawahara et al. (2022)

REFERENCES

Kawahara, H., Kawashima, Y., Masuda, K., et al. 2022,
ApJS, 258, 31, doi: 10.3847/1538-4365/ac3b4d