ML HW5 report

Code explanations

work flow

First of all, get the data in “input.txt”.

```
GP = GaussianProcess(alpha = input_.alpha, lengthscale = input_.lengthscale, variance = input_.variance)
x, y = GP.get_data()
```

Calculate the mean and variance.

```
mean = GP.Cal_mean(x = x, y = y)
variance = GP.Cal_var(x = x, y = y)
```

Optimize the parameter, such as alpha, lengthscale, variance.

```
opt_alpha, opt_lengthscale, opt_variance, error = GP.optimize()

opt_GP = GaussianProcess(alpha = opt_alpha, lengthscale = opt_lengthscale, variance = opt_variance)
```

Use optimal parameter to recalculate the mean and the variance.

```
opt_alpha, opt_lengthscale, opt_variance, error = GP.optimize()

opt_GP = GaussianProcess(alpha = opt_alpha, lengthscale = opt_lengthscale, variance = opt_variance)
```

plotting.

```
para = [str(opt_alpha), str(opt_lengthscale), str(opt_variance)]
plotting(mean, variance, opt_mean, opt_variance, para, x, y)
```

Rational quadratic kernel

$$k(x_a, x_b) = \sigma^2 \left(1 + \frac{\|x_a - x_b\|^2}{2\alpha\ell^2}\right)^{-\alpha}$$

σ^2 is variance
l is lengthscale
alpha is scale-mixture

My code:

```python
def Cal_kernel(self, X1, X2):
    Kernel = np.zeros((len(X1), len(X2)))
    for iter_y in range(len(X1)):
        for iter_x in range(len(X2)):
            term = 1 + (((X1[iter_y] - X2[iter_x]) ** 2) / \
                (2 * self.alpha * (self.lengthscale ** 2)))
            Kernel[iter_y][iter_x] = (self.variance ** 2) * (term ** (- self.alpha))
    return Kernel
```

the Kernel[iter_y][iter_x] is $k(x_a, x_b)$ in the formula

mean

$$\mu(\mathbf{x}^*) = k(\mathbf{x}, \mathbf{x}^*)^\top \mathbf{C}^{-1}\mathbf{y}$$

this formula in picture is cutting form the slide of Prof.Chiu.

C is the covariance matrix which has elements

$$\mathbf{C}(\mathbf{x}_n, \mathbf{x}_m) = k(\mathbf{x}_n, \mathbf{x}_m) + \beta^{-1}\delta_{nm}$$

$k(x, x^*)$: if x and x′ are close to each other (in feature space), y then their y will be also close

beta is hyperparameter.
$delta_{nm}$ is hypermeter, too.

```python
def Cal_mean(self, x, y):
    sample = np.arange(-60, 60, 0.1)

    self.Kernel_xn_xm = self.Cal_kernel(X1 = x, X2 = x)
    self.C_xn_xm = self.Kernel_xn_xm + (1 / self.beta * self.delta)
    self.K_X_Xstar = self.Cal_kernel(X1 = x, X2 = self.plot_x)

    mean = self.K_X_Xstar.T @ (np.linalg.inv(self.C_xn_xm)) @ y
    return mean
```

variance

$$\sigma^2(\mathbf{x}^*) = k^* - k(\mathbf{x}, \mathbf{x}^*)^\top \mathbf{C}^{-1} k(\mathbf{x}, \mathbf{x}^*)$$

k^* is $k(\mathbf{x}^*, \mathbf{x}^*) + \beta^{-1}$

```python
def Cal_var(self, x, y):
    K_Xstar_Xstar = np.zeros(len(self.plot_x))
    variance = np.zeros(len(self.plot_x))

    for iter_pixel in range(len(K_Xstar_Xstar)):
        fucking = 1 + (((self.plot_x[iter_pixel] - self.plot_x[iter_pixel]) ** 2) / \
            (2 * self.alpha * (self.lengthscale ** 2)))
        K_Xstar_Xstar[iter_pixel] = 1 / self.beta + fucking
        variance[iter_pixel] = np.abs(K_Xstar_Xstar[iter_pixel] - \
            self.K_X_Xstar[:,iter_pixel] @ (np.linalg.inv(self.C_xn_xm)) @ self.K_X_Xstar[:,iter_pixel])
    return variance
```

optimize

Using minimize in scipy.optimize optimize the alpha, variance, lengthscale

```python
def optimize(self):
    error = 1000
    inits = np.arange(1, 10, 1)
    opt_alpha, opt_lengthscale = 1.0, 1.0
    for iter_alpha in inits:
        for iter_lengthscale in inits:
            for iter_variance in inits:
                result = minimize(self.Log_Likelihood, x0 = [iter_alpha, iter_lengthscale, iter_variance], method = "SLSQP")
                if result.fun < error:
                    error = result.fun
                    opt_alpha, opt_lengthscale, opt_variance = result.x
    return opt_alpha, opt_lengthscale, opt_variance, error
```

covariance function C with hyper-parameters θ

$$k_\theta(\mathbf{x}_n, \mathbf{x}_m) = \theta_0 \exp\{-\theta_1 \frac{\|\mathbf{x}_n - \mathbf{x}_m\|^2}{2}\} + \theta_2 + \theta_3 \mathbf{x}_n^\top \mathbf{x}_m$$

$$p(\mathbf{y}|\theta) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C}_\theta)$$

$$p(\mathbf{y}|\theta) = \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{C}_\theta)$$

$$\ln p(\mathbf{y}|\theta) = -\frac{1}{2}\ln |\mathbf{C}_\theta| - \frac{1}{2}\mathbf{y}^\top \mathbf{C}_\theta^{-1}\mathbf{y} - \frac{N}{2}\ln (2\pi) \;☞\; \frac{\partial \ln p(\mathbf{y}|\theta)}{\partial \theta}$$

Log likilihood

```python
def Log_Likelihood(self, x0):
    alpha, lengthscale, variance = x0[0], x0[1], x0[2]
    K = self.opt_Cal_kernel(X1 = self.x, X2 = self.x, alpha = alpha, lengthscale = lengthscale, variance = variance)
    lgggg = np.abs(np.linalg.det(K))
    if np.linalg.matrix_rank(K) != len(self.x) or lgggg == 0:
        return 1000

    lg_norm_K = np.log(lgggg)
    inv_K = np.linalg.inv(K)
    ans = 0.5 * (self.y.T @ inv_K @ self.y + lg_norm_K + len(self.x) * np.log(2 * np.pi))
    if ans < 0.001:
        return 1000
    print(ans)
    return ans
```

plotting

```python
def funCtion(x, variance):
    var = np.zeros(len(variance))
    for iter_element in range(len(variance)):
        if variance[iter_element] == 0:
            print("0")
        var[iter_element] = 2 * (variance[iter_element] ** 0.5)
    return var


def plotting(mean, variance, opt_mean, opt_variance, para, x, y):
    plot_x = np.arange(-60, 60, 0.1)
    plt.subplot(121)
    plt.title("Gaussian Process Regression")
    plt.plot(plot_x, mean, 'coral')
    var = funCtion(x = plot_x, variance = variance)
    plt.fill_between(plot_x, mean - var, mean + var, color='aqua')
    plt.scatter(x, y, c = "black")

    plt.subplot(122)
    plt.title("Gaussian Process Regression (Optimized)")
    plt.plot(plot_x, opt_mean, 'coral')
    opt_var = funCtion(x = plot_x, variance = opt_variance)
    plt.fill_between(plot_x, mean - opt_var, mean + opt_var, color='aqua')
    plt.scatter(x, y, c = "black")
    annotated = "alpha : " + para[0] + "\nlengthscale : " + para[1] + "\nvariance : " + para[2]
    plt.text(0.5, -6.1, annotated, size=10, rotation=30.,ha="center", va="bottom", \
        bbox=dict(boxstyle="round", ec=(1., 0.5, 0.5),fc=(1., 0.8, 0.8),))

    plt.show()
```

Settings & Result

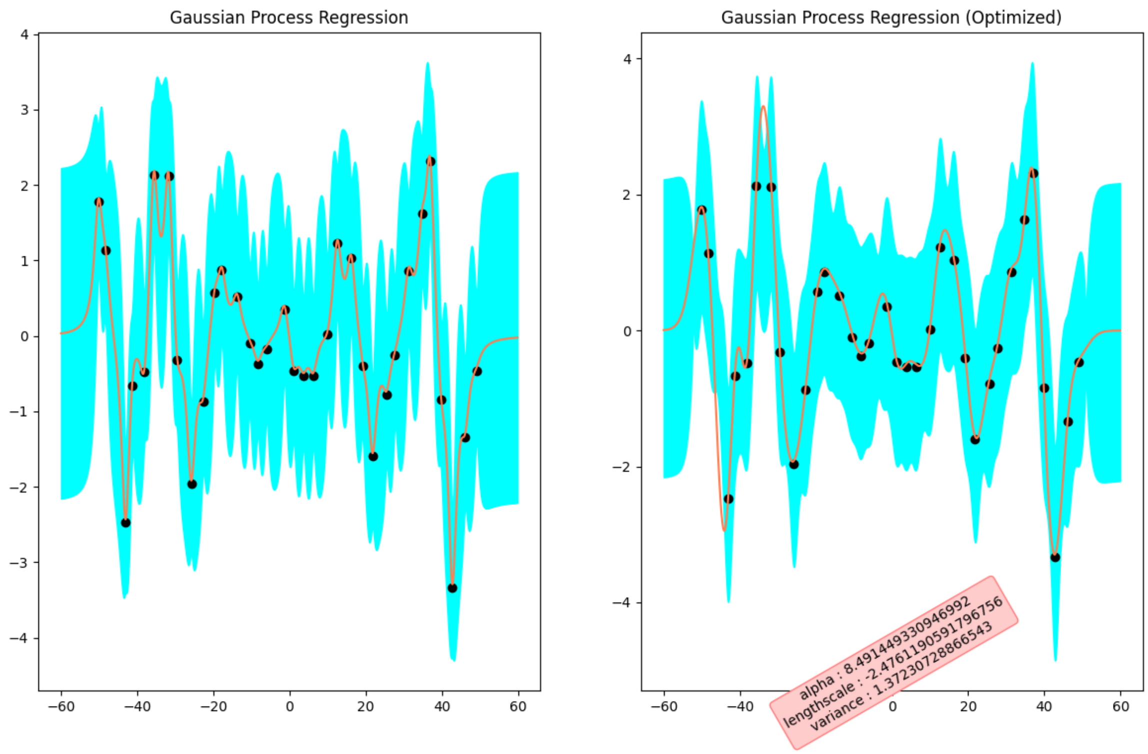
```python
class GaussianProcess(object):
    def __init__(self, alpha, lengthscale, variance):
        self.beta = 5
        self.delta = 1
        self.alpha = alpha
        self.lengthscale = lengthscale
        self.variance = variance
        self.plot_x = np.arange(-60, 60, 0.1)
```

```python
parser = argparse.ArgumentParser()
parser.add_argument("--alpha", type = float, default =  1.0, help = "alpha")
parser.add_argument("--lengthscale", type = float, default =  1.0, help = "lengthscale")
parser.add_argument("--variance", type = float, default = 1.0, help = "variance")
input_ = parser.parse_args()

GP = GaussianProcess(alpha = input_.alpha, lengthscale = input_.lengthscale, variance = input_.variance)
```

I set initial parameter
alpha : 1.0
lengthscale : 1.0
variance : 1.0

After optimizing the parameter, we get
alpha : 8.491449330946992
lengthscale : -2.4761190591796756
variance : 1.37230728866543

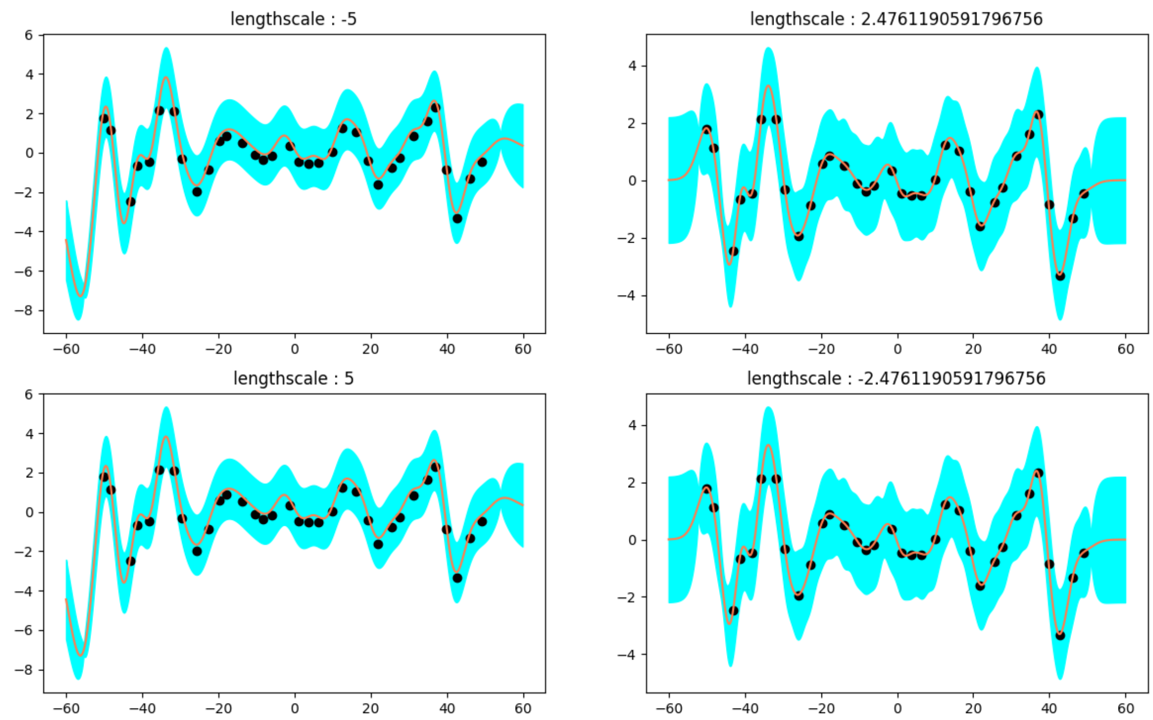


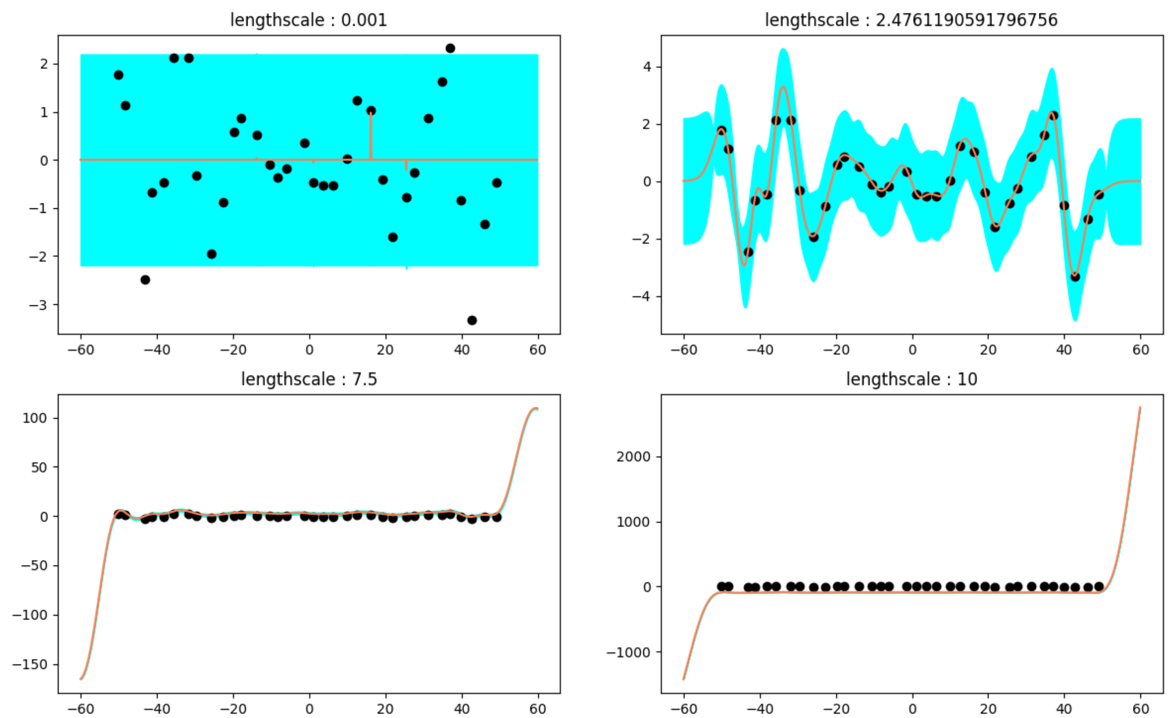
The graph left of picture is the original scatter, mean and variance without optimized parameters.

The graph right of picture is the original scatter, mean and variance with optimized parameters.

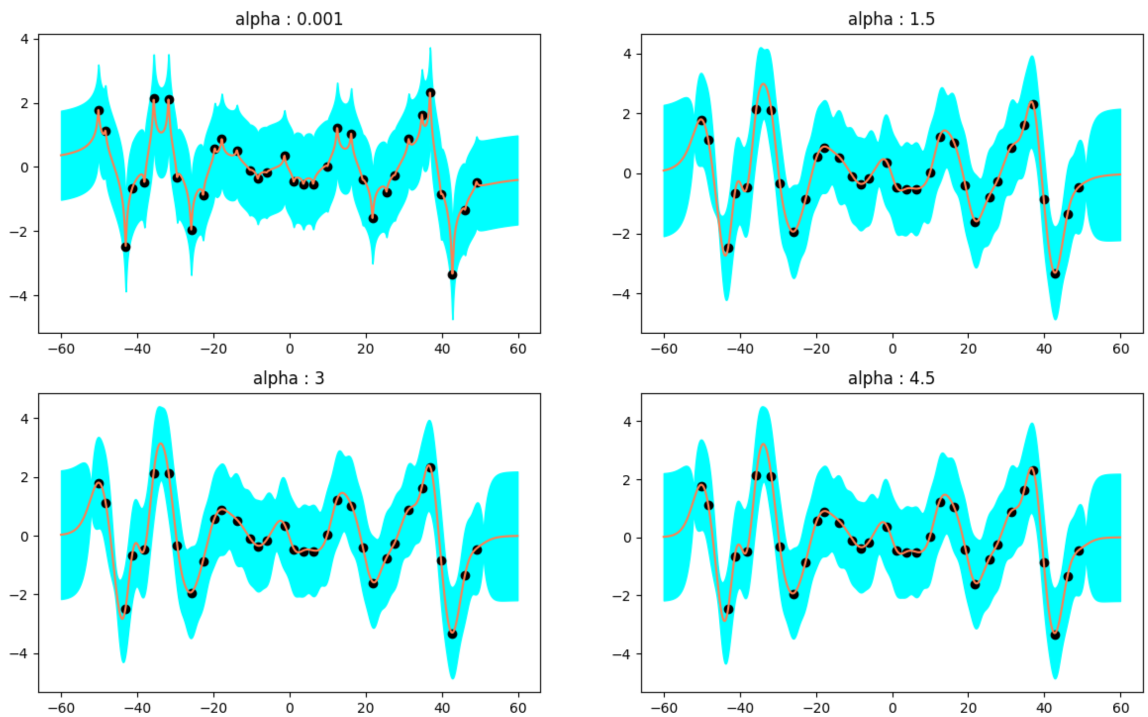
Observations and discussion

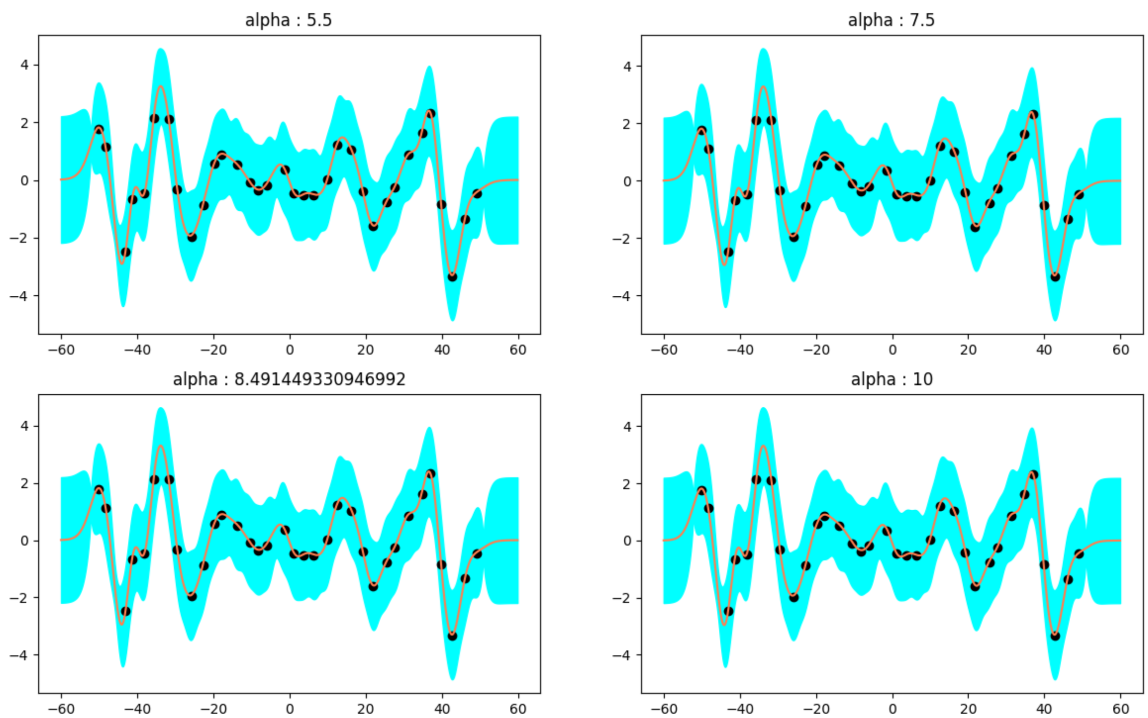
I check the formula in slides. The lengthscale is squared in formula. My optimal lengthscale is negative scalar. However, I think it should be absoluted. It dees not influence the result, thought.





Increasing the lengthscale parameter l increases the overall spread of the covariance.

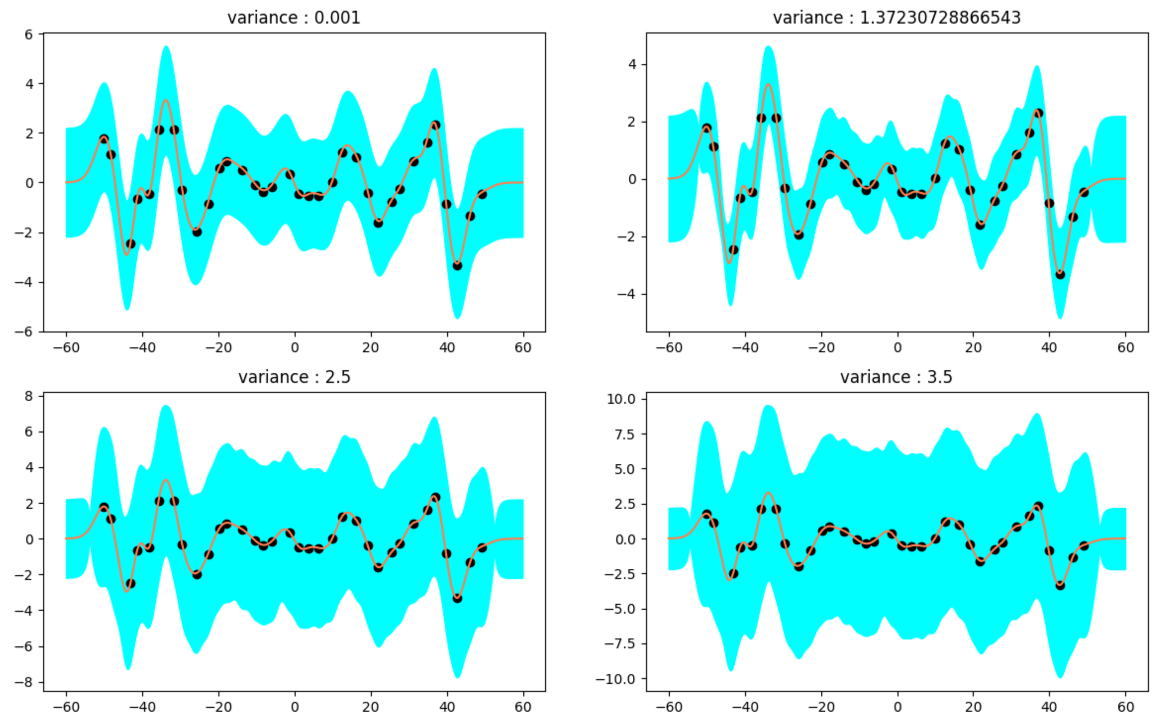


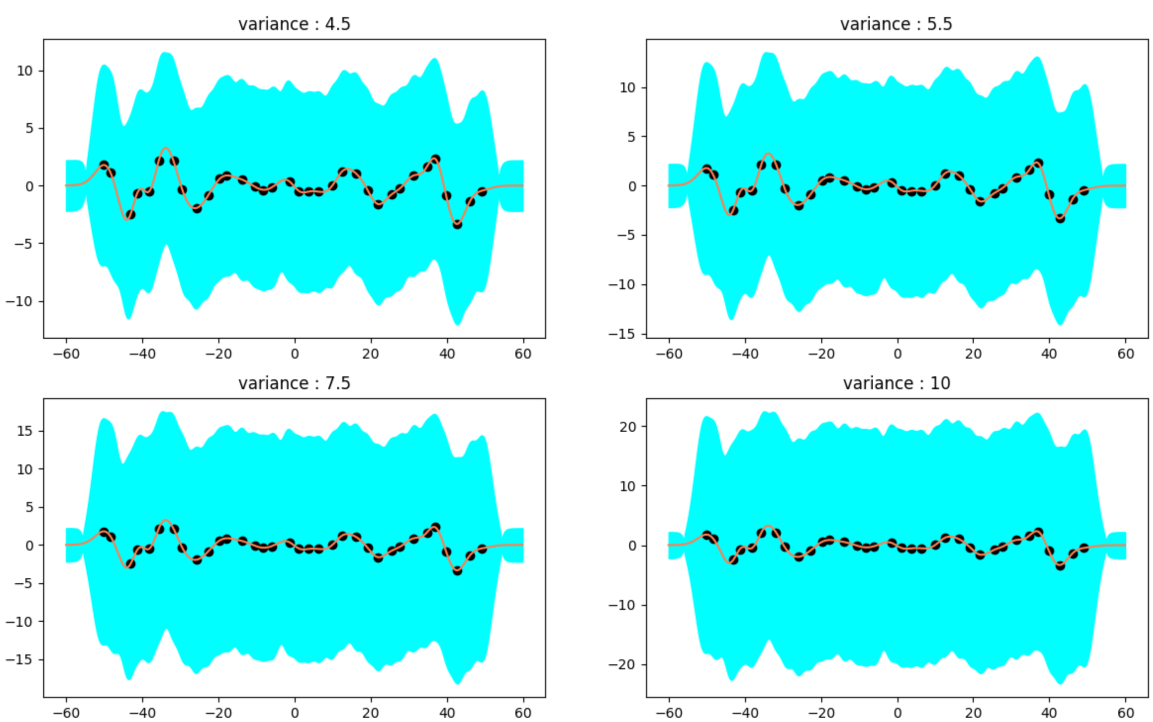


alpha is the scale-mixture.

Decreasing the alpha let more minor local variations while still keeping the longer scale trends.Increasing the alpha to a large value reduces the minor local variations.

When alpha -> ∞ the rational quadratic kernel converges into the exponentiated quadratic kernel.





variance

tags: `MLreport`