Gaussian Process

- Training data:
  - $\begin{bmatrix} x_1 & y_1 \\ . & . \\ . & . \\ . & . \\ x_n & y_n \end{bmatrix} = [X \quad Y]$
- Kernel function:
  - $k(x_n, x_m) = \sigma^2 (1 + \frac{\|x_n - x_m\|^2}{2\alpha\ell^2})^{-\alpha}$

There is a function f could transfer each x_i into coressponding y_i (i.e., $f(x_i) = y_i$).
Assume that $y_i = f(x_i) + \epsilon, where\ \epsilon \sim N(0, \beta)$ and $f \sim N(0, K_n)$.
(i.e., $Y \sim N(f, \beta)$)
On estimate the x_* point, we have formula $\begin{bmatrix} Y \\ y_* \end{bmatrix} \sim N(\begin{bmatrix} Y \\ y_* \end{bmatrix} | 0, K_{n+1})$

After the derivation of probability, we get the

- $\mu(x_*) = k(x, x_*)^T (K_n + \beta I)^{-1} Y$
- $cov(x_*) = k(x_*, x_*) - k(x, x_*)^T (K_n + \beta I)^{-1} k(x, x_*)$

> This form is almost the same as the formula mentioned by Prof. Chiu in the class.
> The tiny difference is that it take the β out of the matrix K, but the course silde takes it into the matrix.

Thus, we could apply the x_* to describe our model.

We notice that the kernel method is decided by some kernel parameters (e.g., σ, α, ℓ), so we need to find the parameters which could have the maximum likelihood.

In my practice, I choose the random value of all parameter between 0 and 10, and call the `scipy.optimize.minimize` to optimize it.
The relative formula is shown below,

$$\begin{aligned} argmax(ln\ p(y \mid \theta)) &= -\frac{1}{2} ln\ |C_\theta| - \frac{1}{2} y^T C_\theta^{-1} - \frac{N}{2} ln\ (2\pi) \\ &\propto -ln\ |C_\theta| - y^T C_\theta^{-1} y \\ &= argmin(\ ln\ |C_\theta| + y^T C_\theta^{-1} y\ ) \end{aligned}$$

Step1

Use `get_K` to compute the covarince matrix, and `get_k_test` use to compute the $k(x, x_*)$ matrix.

```
def rq_kernel(xn, xm, length_scale, scale_mixture, amplitude):
    delta = abs(xn-xm)
```

```
    return amplitude * (1 + delta**2/(2*scale_mixture*(length_scale**2)))**(-
scale_mixture)

def get_K(X, length_scale, scale_mixture, amplitude, beta):
    n = len(X)
    K = np.zeros((n, n), dtype=np.float32)
    for i in range(0, n):
        for j in range(0, n):
            K[i, j] = rq_kernel(X[i], X[j], length_scale, scale_mixture,
amplitude)
            if i==j:
                K[i, j] += 1/beta
    return K

def get_k_test(test, train, length_scale, scale_mixture, amplitude):
    n = len(train)
    K = np.zeros((n, 1), dtype=np.float32)
    for i in range(0, n):
        K[i, 0] = rq_kernel(train[i], test, length_scale, scale_mixture,
amplitude)
    return K
```

Step2

Initial the value of parameters.

```
#given assumption
beta = 5.0

#initial kernel parameter
length_scale = rd.uniform(1, 10.0)
scale_mixture = rd.uniform(1, 10.0)
amplitude = rd.uniform(1, 10.0)
K = get_K(train_x, length_scale, scale_mixture, amplitude, beta)
```

Step3

Add the x_* for the range of [-60, 60], and apply the formula derived above.

```
test_x = np.arange(-60, 60, 0.5)

test_y = np.zeros((len(test_x), 1))
test_var = np.zeros((len(test_x), 1))
for i in range(len(test_x)):
    k = get_k_test(test_x[i], train_x, length_scale, scale_mixture, amplitude)
    test_y[i] = np.matmul(np.matmul(np.transpose(k), np.linalg.inv(K)),
train_y)
    k_new = rq_kernel(test_x[i], test_x[i], length_scale, scale_mixture,
amplitude) + 1/beta
    test_var[i] = k_new - np.matmul(np.matmul(np.transpose(k),
np.linalg.inv(K)), k)
```

Step4

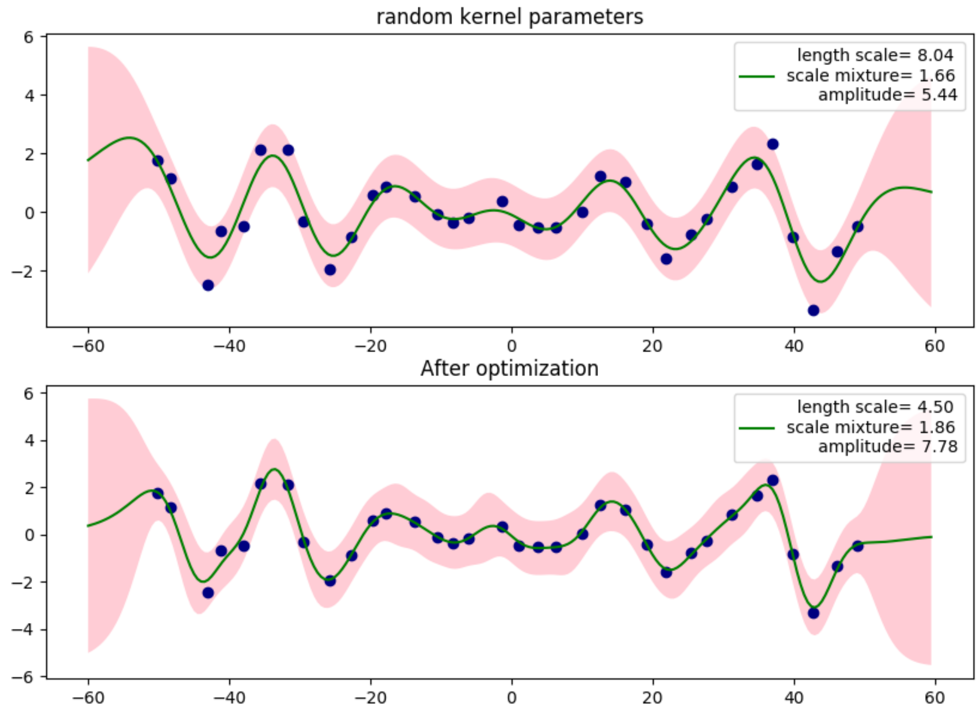
Use `scipy.optimize.minimize` to find the optimized parameter and re-do the gaussian process.

> Found that if using random value of kernel parameters as its initial value, the result after optmizing might be bad for some extremly initial value.

```
#optimize the kernel parameters
def fun(x, args):
    X, Y, beta = args
    K = get_K(X, x[0], x[1], x[2], beta)
    v = np.log(np.linalg.det(K))+np.matmul(np.matmul(np.transpose(train_y),
np.linalg.inv(K)), train_y)
    return v

args = [train_x, train_y, beta]
cons = ({'type': 'ineq', 'fun': lambda x: x[0] - 0.1},
        {'type': 'ineq', 'fun': lambda x: x[1] - 0.1},
        {'type': 'ineq', 'fun': lambda x: x[2] - 0.1})

x0 = np.array((length_scale, scale_mixture, amplitude))
res = minimize(fun, x0, args=[train_x, train_y, beta], method='SLSQP',
constraints=cons)
length_scale, scale_mixture, amplitude = res.x

# re-try Gaussian Process with optimized parameter again
K = get_K(train_x, length_scale, scale_mixture, amplitude, beta)
test_x = np.arange(-60, 60, 0.5)

test_y = np.zeros((len(test_x), 1))
test_var = np.zeros((len(test_x), 1))
for i in range(len(test_x)):
    k = get_k_test(test_x[i], train_x, length_scale, scale_mixture, amplitude)
```

```
    test_y[i] = np.matmul(np.matmul(np.transpose(k), np.linalg.inv(K)),
train_y)
    k_new = rq_kernel(test_x[i], test_x[i], length_scale, scale_mixture,
amplitude) + 1/beta
    test_var[i] = k_new - np.matmul(np.matmul(np.transpose(k),
np.linalg.inv(K)), k)
```

- Result:



Referrence

- https://www.csie.ntu.edu.tw/~cjlin/mlgroup/tutorials/gpr.pdf