```
import numpy as np
from matplotlib import pyplot as plt
import argparse
from q5_1 import *
import ipdb
from scipy.stats import norm
import math
import os
class GPR():
    def __init__(self, var_K):

        self.var_K = var_K
        self.F = None
        self.M = None
        self.M_inv = None
        self.X = None

    def gaussian_kernel(self,x,y):
        return np.exp(-0.5*np.linalg.norm(x-y, ord=2, axis=-1)**2 / self.var_K)

    def objective_func(self,x):
        return x

    def fit(self, X,F):
        XX = np.vstack([X[:,0]]*X.shape[0])
        XXT = np.copy(XX).T.reshape(-1,1)
        XX = XX.reshape(-1,1)

        # M = np.zeros(shape=(X.shape[0],X.shape[0]))
        # for i,x in enumerate(X[:,0]):
            # M[i,:] = self.gaussian_kernel(x*np.ones_like(X), X)

        # ipdb.set_trace()
        M = self.gaussian_kernel(XXT,XX).reshape(X.shape[0],X.shape[0])
        self.M_inv = np.linalg.inv(M+1e-7)
        self.M = M
        self.F = F
        self.X = X

    def predict(self,x):
        v = self.gaussian_kernel(x*np.ones_like(self.X), self.X)
        x_arr = np.array(x).reshape(-1,1)

        # mu = np.matmul(v.T, np.linalg.solve(self.M, self.F))
        # var = self.gaussian_kernel(x_arr,x_arr) - np.matmul(v.T, np.linalg.solve(sel
f.M, v)) + 1e-4

        mu = np.matmul(v.T, np.matmul(self.M_inv, self.F))
        var = self.gaussian_kernel(x_arr,x_arr) - np.matmul(v.T, np.matmul(self.M_inv,
 v))
        if(var<0):
            var = 0.0
        self.mu = mu
        self.var = var
        sample_vals = np.random.normal(mu, var**0.5, 100)

        id_max = np.argmax(sample_vals)
        return mu
        # return sample_vals[id_max]

    def get_mean_std(self):
        X = np.arange(-5,5,0.01)
        means = []
        stds = []
        for x in X:
            v = self.gaussian_kernel(x*np.ones_like(self.X), self.X)
            x_arr = np.array(x).reshape(-1,1)

            # mu = np.matmul(v.T, np.linalg.solve(self.M, self.F))
```

```
            # var = self.gaussian_kernel(x_arr,x_arr) - np.matmul(v.T, np.linalg.solve
(self.M, v)) + 1e-4

            mu = np.matmul(v.T, np.matmul(self.M_inv, self.F))
            var = self.gaussian_kernel(x_arr,x_arr) - np.matmul(v.T, np.matmul(self.M_
inv, v))
            if(var<0):
                var = np.array(0).reshape(var.shape)
            means.append(mu[0]*1.0)
            stds.append(var[0]**0.5)
        return X, np.array(means), np.array(stds)

    def acquisition_func(self, x, ft_max):
        _ = self.predict(x)
        phi = norm.cdf((self.mu-ft_max)/self.var**0.5)
        ei = (self.mu - ft_max) * phi + (self.var**0.5 / (np.sqrt(2*np.pi))) * np.exp(
-(ft_max - self.mu)**2 / (2*self.var))

        return ei

    def optimize_acq_func(self, ft_max):

        X = np.random.uniform(-5,5, 100)
        Y = np.array([self.acquisition_func(x, ft_max) for x in X])

        return X[np.argmax(Y)]



def bayes_optim(model, X, Y, num_iters, dir_name):

    for i in range(num_iters):
        model.fit(X,Y)
        if(i%20==0 or i==98):
            var = model.var_K
            Xs, mean, std = model.get_mean_std()
            plot(var, i+1, Xs, mean, std, X[:,0],Y[:,0], dir_name)
        x_new = model.optimize_acq_func(np.amax(Y))
        y_new = f(x_new)
        # y_new = model.predict(x_new)
        X = np.append(X, np.array(x_new).reshape(-1,1), axis=0)
        Y = np.append(Y, np.array(y_new).reshape(-1,1), axis=0)
        print(x_new, y_new)
    id_max = np.argmax(Y)
    return X[id_max], Y[id_max]

def plot(var, t, Xs, mean, std, Xsamples, Ysamples, dir_name):
    fig = plt.figure()
    plt.plot(Xs, mean, label=r'$\mu_{t}(x)$', color='orangered')
    plt.fill_between(Xs, mean-std, mean+std, facecolor='peachpuff', alpha=0.7)
    plt.scatter(Xsamples, Ysamples)
    plt.title(r'$\sigma^2 =$ {}, t = {}'.format(var, t))
    plt.legend()
    plt.savefig(os.path.join(dir_name,r'var_{}_t_{}.png'.format(var, t)))

    plt.close()

def main():

    # X = np.random.uniform(-5,5,100).reshape(-1,1)
    X = np.array([0]).reshape(-1,1)
    Y = np.array([f(x) for x in X]).reshape(-1,1)
    var_K = 30
    dir_name = os.path.join(os.getcwd(), "q52_var_{}".format(var_K))
    if(not os.path.exists(dir_name)):
        os.mkdir(dir_name)
    num_iters = 100
    gpr_model = GPR(var_K)
```

```
    x_max, f_max = bayes_optim(gpr_model, X, Y, num_iters, dir_name)
    print("({}, {})".format(x_max[0],f_max[0]))

main()
```