HW3

Lakshya Tiwari

1222259194

PROBLEM 1

The Optimization problem can be formulated as $\underset{A_{12},A_{21}}{min} \sum_i (\hat{p}_i - p_i^{obs})^2$

In [1]:
```
# Importing Libraries

import torch as t
from torch.autograd import Variable
import numpy as np
```

In [2]:
```
# Calculation Of Saturation Pressure

p_sat14=10**(7.43155 - 1554.679/(20+240.337))
p_satw=10**(8.071 - 1730.63/(20+233.426))

p_g=[ 28.1 , 34.4 , 36.7 , 36.9 , 36.8 , 36.7 , 36.5 , 35.4 , 32.9 , 27.7 , 17.5
x_g=[ 0.0 , 0.1 , 0.2 , 0.3 , 0.4 , 0.5 , 0.6 , 0.7 , 0.8 , 0.9 , 1.0 ]
```

In [3]:
```
# Saturation Pressure

print("Saturation Pressure of 1,4 dioxane is", p_sat14)
print("Saturation Pressure of water is", p_satw)
```

```
Saturation Pressure of 1,4 dioxane is 28.824099527405245
Saturation Pressure of water is 17.460784103526855
```

In [4]:
```
#Functions

# Pressure
def p(a,x1):
    x2=1-x1
    return x1 * t.exp(a[0]* ( (a[1]*x2)/(a[0]*x1 + a[1]*x2) )**2 ) * p_satw +\
                            x2 * t.exp(a[1]* ( (a[0]*x1)/(a[0]*x1 + a[1]*x2)

# Total sum
def TS(a):
    t=0
    for i in range(len(x_g)):
        xi=x_g[i]
        P=p(a,xi)
        t += (P-p_g[i])**2
    return t

# Step
def l(a):
    s=0.1
    while TS(a-s*a.grad) > TS(a)-s*(0)*np.matmul(a.grad, a.grad):
        s=.25*s
    return s
```

In [5]:
```
# Assuming initial values for A12 and A21 as 2 & 1.
a = Variable(t.tensor([2.0, 1.0]), requires_grad=True)

# Error
e = 150

# Gradient Descent
while e > 0.15:
    obj=TS(a)
    obj.backward()
    step=l(a)
    e = t.linalg.norm(a.grad)
    with t.no_grad():
        a -= step * a.grad
        a.grad.zero_()

print('Final value of a is ' + str(a.data.numpy()))
print('The value of objective function is ' + str(obj.data.numpy()))
```

```
Final value of a is [1.9594604 1.6896328]
The value of objective function is 0.67127305
```

In [6]:
```
# Comparison between given data and modelled data
print('Given P        Modelled P')
for i in range(0,11):
    print(str(p_g[i]) + '         ' + str(p(a.data,x_g[i]).item()))
```

```
Given P        Modelled P
28.1        28.824098587036133
34.4        34.64482498168945
36.7        36.4518280029297
36.9        36.86505126953125
36.8        36.87146759033203
36.7        36.747562408447266
36.5        36.3885498046875
35.4        35.38287353515625
32.9        32.94468688964844
27.7        27.723770141601562
17.5        17.460784912109375
```

The modelled pressures fit very well with the values given in the data.

PROBLEM 2

In [7]:
```
# Importing Libraries

import numpy as np
import sklearn.gaussian_process as gp
from scipy.stats import norm
from scipy.optimize import minimize
```

In [8]:
```
# EXPECTED IMPROVEMENT function.

def expected_improvement(x, gaussian_process, evaluated_loss, greater_is_better=F

    x_to_predict = x.reshape(-1, n_params)

    mu, sigma = gaussian_process.predict(x_to_predict, return_std=True)

    if greater_is_better:
        loss_optimum = np.max(evaluated_loss)
    else:
        loss_optimum = np.min(evaluated_loss)

    scaling_factor = (-1) ** (not greater_is_better)

    # In case sigma equals zero
    with np.errstate(divide='ignore'):
        Z = scaling_factor * (mu - loss_optimum) / sigma
        expected_improvement = scaling_factor * (mu - loss_optimum) * norm.cdf(Z)
        expected_improvement[sigma == 0.0] == 0.0

    return -1 * expected_improvement
```

In [9]:
```
# HYPERPARAMETER function.

def sample_next_hyperparameter(acquisition_func, gaussian_process, evaluated_loss
                               bounds=(0, 10), n_restarts=25):
    best_x = None
    best_acquisition_value = 1
    n_params = bounds.shape[0]

    for starting_point in np.random.uniform(bounds[:, 0], bounds[:, 1], size=(n_r

        res = minimize(fun=acquisition_func,
                       x0=starting_point.reshape(1, -1),
                       bounds=bounds,
                       method='L-BFGS-B',
                       args=(gaussian_process, evaluated_loss, greater_is_better,

        if res.fun < best_acquisition_value:
            best_acquisition_value = res.fun
            best_x = res.x

    return best_x
```

In [10]:
```
# BAYESIAN OPTIMIZATION function.

def bayesian_optimisation(n_iters, sample_loss, bounds, x0=None, n_pre_samples=5,
                          gp_params=None, random_search=False, alpha=1e-5, epsilo

    x_list = []
    y_list = []

    n_params = bounds.shape[0]

    if x0 is None:
        for params in np.random.uniform(bounds[:, 0], bounds[:, 1], (n_pre_sample
            x_list.append(params)
            y_list.append(sample_loss(params))
    else:
        for params in x0:
            x_list.append(params)
            y_list.append(sample_loss(params))

    xp = np.array(x_list)
    yp = np.array(y_list)

    # Create the GP
    if gp_params is not None:
        model = gp.GaussianProcessRegressor(**gp_params)
    else:
        kernel = gp.kernels.Matern()
        model = gp.GaussianProcessRegressor(kernel=kernel,
                                            alpha=alpha,
                                            n_restarts_optimizer=10,
                                            normalize_y=True)

    for n in range(n_iters):

        model.fit(xp, yp)

        # Sample next hyperparameter
        if random_search:
            x_random = np.random.uniform(bounds[:, 0], bounds[:, 1], size=(randon
            ei = -1 * expected_improvement(x_random, model, yp, greater_is_better
            next_sample = x_random[np.argmax(ei), :]
        else:
            next_sample = sample_next_hyperparameter(expected_improvement, model,
                                                     greater_is_better=True, bour

        # Duplicates will break the GP. In case of a duplicate, we will randomly
        if np.any(np.abs(next_sample - xp) <= epsilon):
            next_sample = np.random.uniform(bounds[:, 0], bounds[:, 1], bounds.sh

        # Sample loss for new set of parameters
        cv_score = sample_loss(next_sample)

        # Update lists
        x_list.append(next_sample)
        y_list.append(cv_score)
```

```
        # Update xp and yp
        xp = np.array(x_list)
        yp = np.array(y_list)

    return xp, yp
```

In [11]:
```
# Setup

# Bounds
b = np.array([[-3, 3],[-2,2]])
# Function
def fun(x):
    x1=x[0]
    x2=x[1]
    return -1*((4 - 2.1*x1**2 + (x1**4)/3)*x1**2 + x1*x2 + (-4 + 4*(x2**2))*x2**2

import warnings
warnings.filterwarnings('ignore')

# Bayesian Optimisation
z1, z2 = bayesian_optimisation(100, fun, b, n_pre_samples=5, random_search=100000
```

In [12]:
```
# Solution
print('Value of X1 and X2 is \t', z1[100])
print('Value of minimized function is \t', z2[100])
```

```
Value of X1 and X2 is 	 [ 0.04588308 -0.76349676]
Value of minimized function is 	 0.9991121097405781
```