

가우시안 프로세스 (Gaussian Process)

Moon Il-chul(icmoon@kaist.ac.kr); Kim Hye-mi(khm0308@kaist.ac.kr); Na Byeong-hu(wp03052@kaist.ac.kr)

본 코드의 목적은 가우시안 프로세스 리그레션(Gaussian Process Regression)을 통해 샘플링한 데이터를 바탕으로 실제함수를 추정하는 것입니다. 이는 관찰된 포인트 집합을 다변수 정규분포를 따르는 $T_N = [t_1, t_2, \dots, t_N]^T$ 라고 가정하여, T_N 을 바탕으로 t_{N+1} 의 확률분포인 $P(t_{N+1}|T_N)$ 을 찾음으로써 달성을 할 수 있습니다.

$P(T)$ 의 확률분포

먼저 연속 도메인 위 N 개의 포인트 집합 $X_n = [x_1, x_2, \dots, x_n]^T$ 이 M 차원으로 매핑될 때, M 차원의 웨이트 벡터를 W , 매핑된 잠재함수의 값을 $Y_n = [y_1, y_2, \dots, y_n]^T$ 이라 할 때, 추정하는 잠재함수 Y 의 확률분포를 구하면 다음과 같습니다.

$$P(Y) = N(W| 0, K)$$

$$\text{이때, } W = [w_1, w_2, \dots, w_M]^T, K_{nm} = K(x_n, x_m) = \frac{1}{\alpha} \phi(x_n) \phi(x_m)$$

여기서 하나의 포인트 x_n 에 대해 실제 관측된 값을 t_n , 가우시안 에러를 e_n 이라고 하면, $t_n = y_n + e_n$ 이 성립합니다. 이에 따라 $P(T| Y)$ 를 구하면 다음과 같습니다.

$$P(T| Y) = N(T| Y, \beta^{-1} I_N)$$

여기서 다변수 정규분포의 성질을 이용하여 단일분포 $P(T)$ 를 구하면 다음과 같은 식으로 나타납니다.

$$P(T) = N\left(T \middle| 0, \frac{1}{\beta} I_N + K\right)$$

$P(t_{N+1})$ 의 확률분포

T_N 에 t_{N+1} 을 추가한 T_{N+1} 의 확률분포는 다음과 같습니다.

$$P(T_{N+1}) = P(T_N, t_{N+1}) = N\left(T \middle| 0, \begin{pmatrix} \frac{1}{\beta} I_N + K & k_{1(N+1)} \\ k_{(N+1)1} & k_{(N+1)(N+1)} + \frac{1}{\beta} \end{pmatrix}\right)$$

$$cov_{N+1} = \begin{bmatrix} cov_N & k \\ k^T & c \end{bmatrix}$$

다변수 정규분포의 특징을 이용하면, 위의 분포를 통해 아래의 조건분포를 구할 수 있습니다.

$$P(t_{N+1}|T_N) = N(t_{N+1} \middle| 0 + k^T cov_N^{-1}(T_N - 0), c - k^T cov_N^{-1}k)$$

따라서 $\mu_{t_{N+1}} | T_N = k^T cov_N^{-1} T_N$, $\sigma_{t_{N+1}} | T_N = c - k^T cov_N^{-1} k$ 입니다.

커널함수

본 코드는 아래와 같은 커널함수를 이용하여 공분산행렬을 구성하고 데이터 값을 예측합니다.

$$K_{nm} = k(x_n, x_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \|x_n - x_m\|^2\right) + \theta_2 + \theta_3 x_n^T x_m$$

In [1]:

```
///
@ copyright: AAI lab (http://aailab.kaist.ac.kr/xe2/page_GBox27)
@ author: Moon Il-chul: icmoon@kaist.ac.kr
@ annotated by Kim Hye-mi: khm0308@kaist.ac.kr; Na Byeong-hu: wp03052@kaist.ac.kr
///

import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from scipy.stats import norm
from scipy import linalg
```

In [2]:

```
%matplotlib inline
```

In [3]:

```
# 본 코드에서 사용하는 커널함수 (Tensorflow를 사용하여 데이터를 바탕으로 parameter를 학습함)
def KernelFunctionWithTensorFlow(theta0, theta1, theta2, theta3, X1, X2):
    insideExp = tf.multiply(tf.div(theta1, 2.0), tf.matmul((X1 - X2), tf.transpose(X1 - X2)))
    firstTerm = tf.multiply(theta0, tf.exp(-insideExp))
    secondTerm = theta2
    thridTerm = tf.multiply(theta3, tf.matmul(X1, tf.transpose(X2)))
    ret = tf.add(tf.add(firstTerm, secondTerm), thridTerm)
    return ret
```

In [4]:

```
# 본 코드에서 사용하는 커널함수 (위와 식은 동일하나, 학습하지 않음)
def KernelFunctionWithoutTensorFlow(theta, X1, X2):
    ret = theta[0] * np.exp(np.multiply(-theta[1] / 2.0, np.dot(np.subtract(X1, X2), np.subtract(X1, X2))))
        + theta[2] + theta[3] * np.dot(np.transpose(X1), X2)
    return ret
```

In [5]:

```

def KernelHyperParameterLearning(trainingX, trainingY):
    tf.reset_default_graph()
    numDataPoints = len(trainingY)
    numDimension = len(trainingX[0])

    # input & output (for TensorFlow)
    obsX = tf.placeholder(tf.float32, [numDataPoints, numDimension])
    obsY = tf.placeholder(tf.float32, [numDataPoints, 1])

    # 학습률 parameter (for TensorFlow)
    theta0 = tf.Variable(1.0)
    theta1 = tf.Variable(1.0)
    theta2 = tf.Variable(1.0)
    theta3 = tf.Variable(1.0)
    beta = tf.Variable(1.0)

    # 커널 함수를 이용하여 공분산 행렬을 구성함
    matCovarianceLinear = []
    for i in range(numDataPoints):
        for j in range(numDataPoints):
            kernelEvaluationResult = KernelFunctionWithTensorFlow(theta0, theta1, theta2, theta3
            ,
                           tf.slice(obsX, [i, 0], [1, num
                           Dimension]),
                           tf.slice(obsX, [j, 0], [1, num
                           Dimension]))
            if i != j:
                matCovarianceLinear.append(kernelEvaluationResult)
            if i == j:
                matCovarianceLinear.append(kernelEvaluationResult + tf.div(1.0, beta))

    matCovarianceCombined = tf.convert_to_tensor(matCovarianceLinear, dtype=tf.float32)
    matCovariance = tf.reshape(matCovarianceCombined, [numDataPoints, numDataPoints])
    matCovarianceInv = tf.matrix_inverse(matCovariance)

    # 공분산 행렬을 바탕으로 예측값과 예측값의 분산을 구함
    sumssquarederror = 0.0
    for i in range(numDataPoints):
        k = tf.Variable(tf.ones([numDataPoints]))
        for j in range(numDataPoints):
            kernelEvaluationResult = KernelFunctionWithTensorFlow(theta0, theta1, theta2, theta3
            ,
                           tf.slice(obsX, [i, 0], [1, num
                           Dimension]),
                           tf.slice(obsX, [j, 0], [1, num
                           Dimension]))
            indices = tf.constant([j])
            tempTensor = tf.Variable(tf.zeros([1]))
            tempTensor = tf.add(tempTensor, kernelEvaluationResult)
            tf.scatter_update(k, tf.reshape(indices, [1, 1]), tempTensor)

            c = tf.Variable(tf.zeros([1, 1]))
            kernelEvaluationResult = KernelFunctionWithTensorFlow(theta0, theta1, theta2, theta3,
                                                               tf.slice(obsX, [i, 0], [1, numDimension]),
                                                               tf.slice(obsX, [i, 0], [1, numDimension]))

            c = tf.add(tf.add(c, kernelEvaluationResult), tf.div(1.0, beta))

```

```

k = tf.reshape(k, [1, numDataPoints])

# 예측값 및 예측값의 표준편차
predictionMu = tf.matmul(k, tf.matmul(matCovarianceInv, obsY))
predictionVar = tf.subtract(c, tf.matmul(k, tf.matmul(matCovarianceInv, tf.transpose(k
)))))

# 예측값과 실제값의 sum of squared error를 구함
sumsquarederror = tf.add(sumsquarederror, tf.pow(tf.subtract(predictionMu, tf.slice(obsY
, [i, 0], [1, 1])), 2))

# Training session declaration
# Gradient Descent Optimizer를 활용하여 sum of squared error를 최소화하는 parameter를 구함
training = tf.train.GradientDescentOptimizer(0.1).minimize(sumsquarederror)

# Session을 초기화함
gpu_options = tf.GPUOptions(per_process_gpu_memory_fraction=0.333)
sess = tf.Session(config=tf.ConfigProto(gpu_options=gpu_options))
init = tf.global_variables_initializer()
sess.run(init)

### Check PSD
def isPSD(A, tol=1e-8):
    E,V = linalg.eigh(A)
    return np.all(E > -tol)

# 학습을 위해 Session을 실행
for i in range(100):
    sess.run(training, feed_dict={obsX: trainingX, obsY: trainingY})

    trainedTheta = []
    trainedTheta.append(sess.run(theta0, feed_dict={obsX: trainingX, obsY: trainingY}))
    trainedTheta.append(sess.run(theta1, feed_dict={obsX: trainingX, obsY: trainingY}))
    trainedTheta.append(sess.run(theta2, feed_dict={obsX: trainingX, obsY: trainingY}))
    trainedTheta.append(sess.run(theta3, feed_dict={obsX: trainingX, obsY: trainingY}))
    trainedBeta = sess.run(beta, feed_dict={obsX: trainingX, obsY: trainingY})

    #print("----- Iteration ", i, " -----")
    #print("Sum of Squared Error : ", sess.run(sumsquarederror, feed_dict={obsX: trainingX, o
bsY: trainingY}))
    #print("Theta : ", trainedTheta)
    #print("Beta : ", trainedBeta)

sess.close()

# 학습된 parameter를 return함
return trainedTheta, trainedBeta

```

In [6]:

```

snr = 0.2 # signal to noise ratio
numObservePoints = 5
numInputDimension = 1

X = np.arange(0, 2 * np.pi, 0.1)
numTruePoints = X.shape[0]
trueY = np.sin(X) # 실제함수

```

In [7]:

```

trainingX = [] # training에 이용할 X 데이터
trainingY = [] # training에 이용할 Y 데이터

for itr2 in range(numObservePoints):
    sampleX = 2 * np.pi * np.random.random()
    trainingX.append([sampleX])
    trainingY.append([np.sin(sampleX) + snr * np.random.randn()])
numPoints = len(trainingX)

print("training set of X: ", trainingX)
print("training set of Y: ", trainingY)

```

```

training set of X: [[3.391299497541918], [6.089790468622297], [5.74938389829320
3], [1.1471867004267506], [2.750539925566083]]
training set of Y: [[-0.3137204147759303], [-0.30517652007738044], [-0.0727691236
929981], [0.9031585416912293], [0.33506478937059775]]

```

In [8]:

```

trainedTheta, trainedBeta = KernelHyperParameterLearning(trainingX, trainingY)
print("----- Trained Result -----")
print("Theta : ", trainedTheta)
print("Beta : ", trainedBeta)

```

```

----- Trained Result -----
Theta : [0.8447998, 0.8332223, 1.0954661, 0.95990527]
Beta : 1.4701568

```

In [9]:

```

def PredictionGaussianProcessRegression(theta, beta, C_inv, numPoints, sampleX, sampleY, inputElement):
    k = np.zeros(numPoints)
    for i in range(numPoints):
        # 추가한  $t_{N+1}$ 과 기존  $T_N$ 으로 인해 추가되는 공분산 행렬의 k행렬 영역
        k[i] = KernelFunctionWithoutTensorFlow(theta, sampleX[i], inputElement)

    # 추가한  $t_{N+1}$ 로 인해 변경되는 공분산 행렬의 c영역
    c = KernelFunctionWithoutTensorFlow(theta, inputElement, inputElement) + 1.0 / beta

    #  $P(t_{n+1})$  확률분포의 평균과 표준편차
    mu = np.dot(k, np.dot(C_inv, sampleY))
    var = c - np.dot(k, np.dot(C_inv, k))

    return mu[0], var

```

In [10]:

```
# 커널 함수를 이용하여 공분산 행렬을 나타냄
def KernelCalculation(theta, beta, numPoints, sampleX):
    C = np.zeros((numPoints, numPoints))

    for i in range(numPoints):
        for j in range(numPoints):
            C[i, j] = KernelFunctionWithoutTensorFlow(theta, sampleX[i], sampleX[j])
            if i == j:
                C[i, j] += 1.0 / beta

    return C, np.linalg.inv(C)
```

In [11]:

```
def PlottingGaussianProcessRegression(plotN, strTitle, inputs, mu_next, sigma2_next, sampleX, sampleY):
    plt.subplot(5, 2, plotN)
    plt.xlim([0, 6])
    plt.ylim([-3, 3])
    plt.title(strTitle)

    plt.fill_between(inputs, mu_next - 2 * np.sqrt(sigma2_next), mu_next + 2 * np.sqrt(sigma2_ne
xt),
                     color=(0.9, 0.9, 0.9))
    plt.plot(sampleX, sampleY, 'r+', markersize=10) # training set은 빨강색 +로 표시함
    plt.plot(X, trueY, 'g-') # 실제함수는 초록색 실선으로 나타냄
    plt.plot(inputs, mu_next, 'bo-', markersize=5) # 예측값은 파란색 점으로 나타나며, 실선으로
    #이어짐
```

In [12]:

```

## 커널 parameter가 학습되지 않은 경우와 학습된 경우를 비교함
sampleXs = []
sampleYs = []

showVisualization = [1, 5, 10, 20, 30]
numMaxPoints = 50
theta = np.array([1, 1, 1, 1])
beta = 300
plt.figure(1, figsize=(14, 25), dpi=100)
plotN = 1

for itr2 in range(numMaxPoints):
    sampleX = 2 * np.pi * np.random.random()
    sampleXs.append([sampleX])
    sampleYs.append([np.sin(sampleX) + snr * np.random.randn()])
    # sampleY는 실제 함수에 노이즈(snr * np.random.randn())가 더해진 값으로 우리가 실제로 얻게 되는 디지털 값을 의미함
    inputs = np.arange(0, 2 * np.pi, 0.3)

    # parameter를 학습하지 않은 경우
    mu_next = []
    sigma2_next = []
    if itr2 in showVisualization:
        C, C_inv = KernelCalculation(theta, beta, len(sampleYs), sampleXs)

        for itr1 in range(len(inputs)):
            mu, var = PredictionGaussianProcessRegression(theta, beta, C_inv, len(sampleYs), sampleXs, sampleYs,
                                                            inputs[itr1])
            mu_next.append(mu)
            sigma2_next.append(var)

    PlottingGaussianProcessRegression(plotN,
                                       'Without Kernel Parameter Learning After %s sampling' % (len(sampleYs)),
                                       inputs, mu_next, sigma2_next, sampleXs, sampleYs)
    plotN += 1

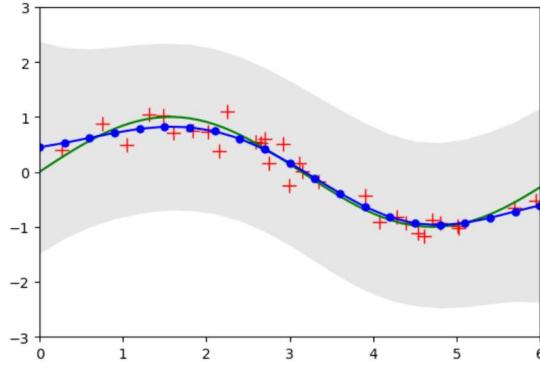
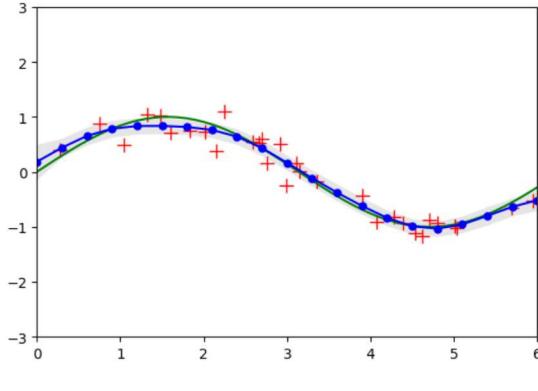
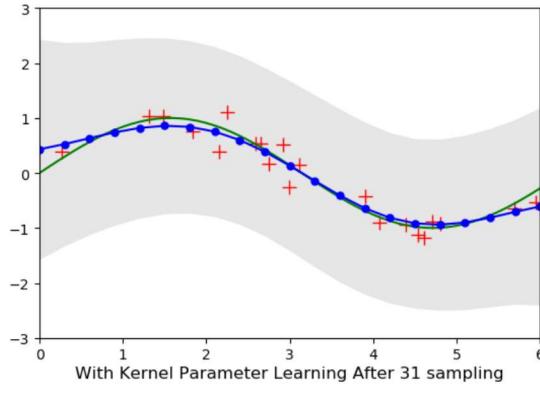
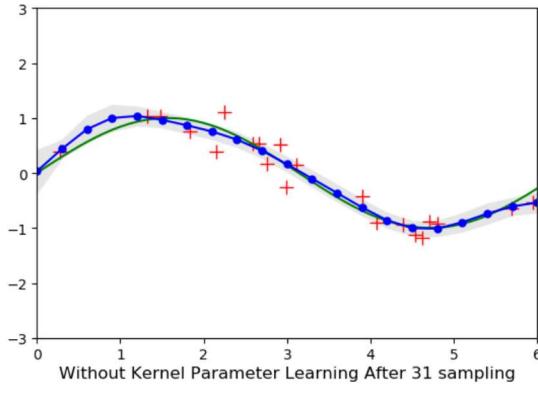
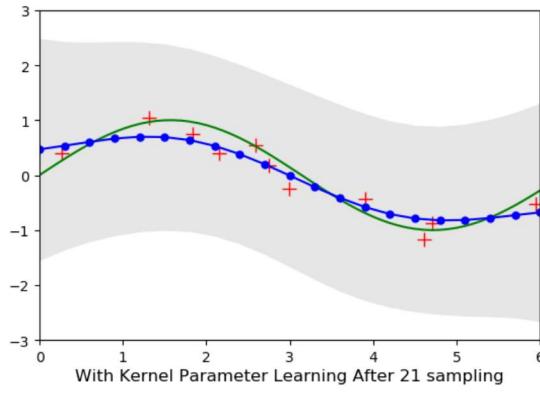
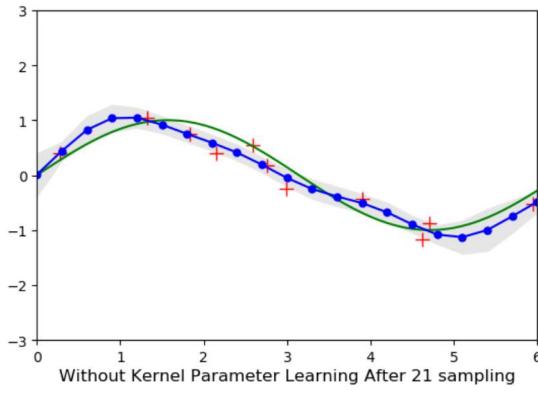
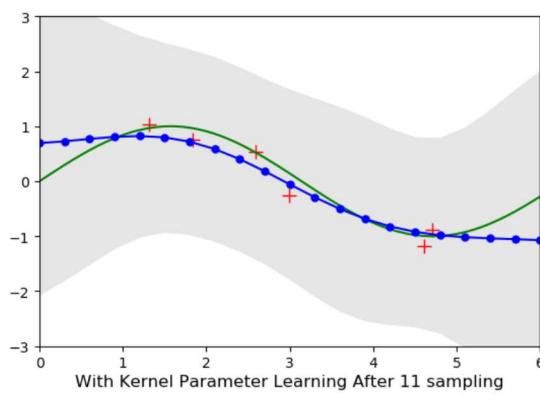
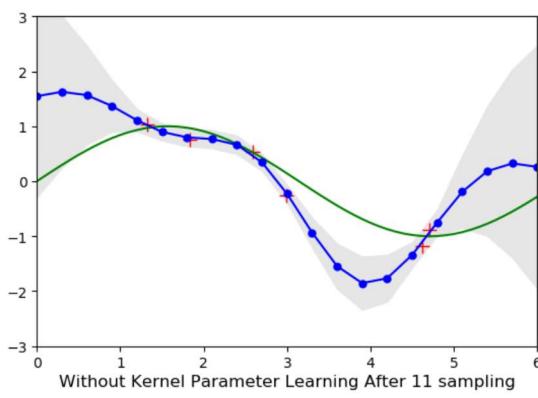
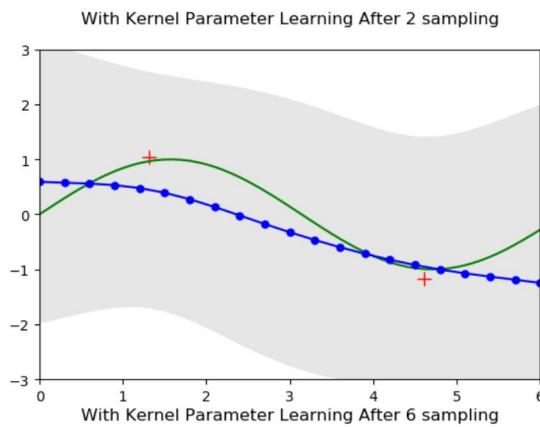
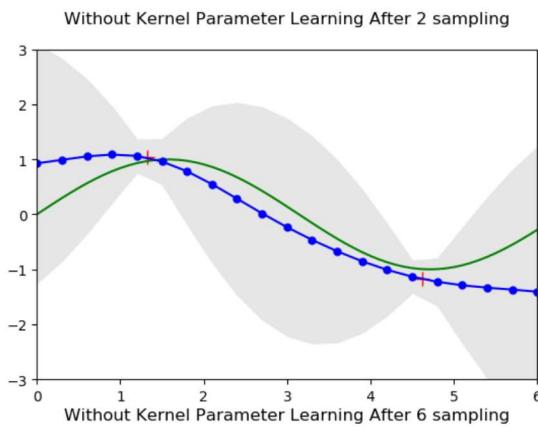
    # parameter를 학습한 경우
    mu_next = []
    sigma2_next = []
    if itr2 in showVisualization:
        trainedTheta, trainedBeta = KernelHyperParameterLearning(sampleXs, sampleYs)
        C, C_inv = KernelCalculation(trainedTheta, trainedBeta, len(sampleYs), sampleXs)

        for itr1 in range(len(inputs)):
            mu, var = PredictionGaussianProcessRegression(trainedTheta, trainedBeta, C_inv, len(sampleYs), sampleXs,
                                                            sampleYs, inputs[itr1])
            mu_next.append(mu)
            sigma2_next.append(var)

    PlottingGaussianProcessRegression(plotN, 'With Kernel Parameter Learning After %s sampling' % (len(sampleYs)),
                                       inputs, mu_next, sigma2_next, sampleXs, sampleYs)
    plotN += 1

plt.show()

```



결과해석부

왼쪽열의 그래프는 학습을 하지 않았을 때, 오른쪽 열의 그래프는 학습 후의 가우시안 프로세스 리그레션 결과입니다. tensorflow를 활용하여 변수(평균과 분산)를 학습하였을 때의 결과가 실제함수를 더 잘 예측하고 있음을 확인할 수 있습니다.

그래프에서 초록색 실선은 실제함수를, 빨강색 + 점은 샘플링 값을, 파랑색 실선은 0.3 간격으로 찍힌 포인트 집합에서 가우시안 프로세스 리그레션을 바탕으로 구한 예측값(평균값)을 연결한 선입니다. 파랑색 실선 주위에 표시된 영역은 각 지점에서 예측된 함수의 분산을 나타냅니다.

위 그래프를 통해 샘플링 된 지점의 분산이 감소하는 것을 확인할 수 있으며, 샘플링 횟수가 증가할수록 가우시안 프로세스 리그레션을 통해 추정한 함수가 실제함수와 유사해짐을 확인할 수 있습니다.

베이지안 최적화

베이지안 최적화는 순차적이고 입력값을 정할 수 있는 실험을 진행한다고 가정할 때, 결과값을 최대 혹은 최소로 만드는 입력값(input)을 찾는 것을 목표로 합니다. 이를 수식으로 나타내면 아래와 같습니다.

$$x^* = \operatorname{argmax}_{x \in X} f(x)$$

베이지안 최적화는 우리가 결과값을 결정하는 잠재함수를 모른다는 점, 결과값과 입력값이 연속적이라는 점, 결과값이 확률적(stochastic)으로 발생한다는 점이 특징입니다. 베이지안 최적화는 분산을 줄여주게끔 샘플링 하 *exploitation* 과정과 평균이 커지는 지점을 반복하여 샘플링하는 *exploration* 과정을 반복하여 잠재함수를 실제함수와 유사하게 학습시킵니다. 여기서 적절한 샘플링 방식을 결정해주는 함수가 ‘Acquisition Function’으로 크게 Maximum Probability Improvement와 Maximum Expected Improvement가 있습니다.

MPI (Maximum Probability Improvement)

기존의 데이터 D 가 가지고 있는 함수 $f(x)$ 의 최대값을 y_{max} , 새로운 샘플링 포인트 x 를 잡았을 때의 함수값 $y = f(x)$ 라고 할 때, y 가 y_{max} 보다 margin m 만큼 혹은 더 크게 할 확률 중, 가장 높은 확률을 가지게 하는 x 를 구하는 방법입니다. 이는 아래와 같은 수식으로 표현됩니다.

$$MPI(x|D) = \operatorname{argmax}_x \phi\left(\frac{\mu - (1+m)y_{max}}{\sigma}\right)$$

MEI (Maximum Expected Improvement)

Maximum Expected Improvement는 margin m 을 0에서 무한대까지 순차적으로 변화시킴으로써 m 에 대한 평균값이 가장 큰 x 를 구합니다. Maximum Probability of Improvement에서 m 의 최적값을 구해야하는 점을 보완한 것입니다. 여기서는 다음과 같은 가정을 합니다.

$$y = f(x), y_{max} = \max_{m=1,\dots,n} f(x_n)$$

$$u = \frac{y_{max} - \mu}{\sigma}, v = \frac{u - \mu}{\sigma}$$

$$\mu = f(x | D), \sigma = K(x | D)$$

$$m = \max(0, y - y_{max}) = \max(0, (v - u) \sigma)$$

이 때, Maximum Expected Improvement는 아래와 같은 수식으로 표현됩니다.

$$MEI(x | D) = \operatorname{argmax}_x \int_0^\infty P(y_{max} + m) mdm = \frac{1}{2} \sigma^2 (-u\phi(u) + (1+u^2)\Phi(-u))$$

In [13]:

```

## MPI
def AcquisitionFunctionProblImprovement(sampleX, sampleY, m, Xs, Mus, Sigmas):
    numSamples = len(sampleY)

    idxMax = -1
    Ymax = -1000000

    # 샘플 Y의 최대값과 0을 값으로 가지는 index를 찾음
    for itr in range(numSamples):
        if sampleY[itr][0] > Ymax:
            Ymax = sampleY[itr][0]
            idxMax = itr

    # 각 샘플에서의 probability of improvement를 구함
    probImprovements = []
    for itr in range(len(Mus)):
        probImprovements.append((Mus[itr] - (1 + m) * Ymax) / np.sqrt(Sigmas[itr]))
    probImprovements[itr] = norm.cdf(probImprovements[itr])

    #print("Prob Improvements : ",probImprovements)

    idxProbMax = 0
    Probmax = -1
    # Probability of Improvement를 최대화하는 샘플 X를 찾음
    for itr in range(len(Mus)):
        if probImprovements[itr] > Probmax:
            Probmax = probImprovements[itr]
            idxProbMax = itr

    return Xs[idxProbMax], probImprovements

```

In [14]:

```
#MEI
def AcquisitionFunctionExpectedImprovement(sampleX, sampleY, Xs, Mus, Sigmas):
    numSamples = len(sampleY)

    idxMax = -1
    Ymax = -1000000

    # 샘플 Y의 최대값과 이를 갖으로 가지는 index를 찾음
    for itr in range(numSamples):
        if sampleY[itr][0] > Ymax:
            Ymax = sampleY[itr][0]
            idxMax = itr

    # 각 샘플에서의 expected improvement를 구함
    expectedImprovements = []
    for itr in range(len(Mus)):
        u = (Ymax - Mus[itr]) / np.sqrt(Sigmas[itr])
        expectedImprovements.append(Sigmas[itr] * (-u * norm.pdf(u) + (1 + pow(u,2)) * norm.cdf(-u)))

    #print("Expected Improvements : ",expectedImprovements)

    idxEIMax = 0
    EImax = -1
    # Expected Improvement을 최대화하는 샘플 X를 찾음
    for itr in range(len(Mus)):
        if expectedImprovements[itr] > EImax:
            EImax = expectedImprovements[itr]
            idxEIMax = itr

    return Xs[idxEIMax], expectedImprovements
```

In [15]:

```
# Bayesian Optimization Result와 각 Acquisition Function 함수의 변화 비교
# 본 코드에서는 MPI(Maximum Probability Improvement)로 Sampling을 진행함
snr = 0.1
X = np.arange(0, 2, 0.01)
numTruePoints = X.shape[0]
trueY = np.zeros(numTruePoints)
for i in range(8):
    S_X = np.multiply(np.divide(np.abs(np.multiply(X, pow(2,i)) - np.around(np.multiply(X, pow(2,i)))), pow(2,i)),2)
    trueY = np.add(trueY,S_X)

sampleXs = []
sampleYs = []

showVisualization = [0, 4, 9, 14, 19]
numTrials = showVisualization[4]+1
trainedTheta = np.array([1, 1, 1, 1])
trainedBeta = 1
m = 1
kernelLearning = True

acquisitionFunction = 'ProblImprovement'
#acquisitionFunction = 'ExpectedImprovement'

plt.figure(1, figsize=(14, 25), dpi=100)
plotN = 1

#초기 X값 설정
for itr2 in range(1):
    sampleX = 2 * np.random.random()
    sampleXs.append([sampleX])
    sampleY = 0
    for i in range(8):
        S_X = 2 * np.divide(np.abs(np.multiply(sampleX, pow(2,i)) - np.around(np.multiply(sampleX, pow(2,i)))), pow(2,i))
        sampleY += S_X
    sampleYs.append([sampleY + snr * np.random.randn()])

# kernel Learning을 시킬경우, 아래의 조건문을 실행함
#print("Learning Kernel Parameters.....")
if kernelLearning == True:
    trainedTheta, trainedBeta = KernelHyperParameterLearning(sampleXs, sampleYs)
    C, C_inv = KernelCalculation(trainedTheta, trainedBeta, len(sampleYs), sampleXs)
#print("Trained Kernel Parameters : ", trainedTheta, trainedBeta)

# Bayesian Optimization 수행
for itr2 in range(numTrials):

    Xs = np.arange(0, 2, 0.01)
    Mus = []
    Sigmas = []

    #print("Calculating Predicted Values.....")
    for itr1 in range(len(Xs)):
        mu, var = PredictionGaussianProcessRegression(trainedTheta, trainedBeta, C_inv, len(sampleYs), sampleXs,
                                                       sampleYs, Xs[itr1])
        Mus.append(mu)
        Sigmas.append(var)
```

```

#print("Calculated Results : ",Mus,Sigmas)

# Acquisition Function에 따라 위에서 구한 파라미터를 대입하여 다음 샘플링 포인트를 찾아줌
#print("Calculating Acquisition Values.....")
if acquisitionFunction == 'ProblImprovement':
    nextX, probImprovements = AcquisitionFunctionProblImprovement(sampleXs, sampleYs, m, Xs, M
us, Sigmas)
    if itr2 in showVisualization:
        _, expectedImprovements = AcquisitionFunctionExpectedImprovement(sampleXs, sampleYs,
Xs, Mus, Sigmas)
    if acquisitionFunction == 'ExpectedImprovement':
        nextX, expectedImprovements = AcquisitionFunctionExpectedImprovement(sampleXs, sampleYs,
Xs, Mus, Sigmas)
        if itr2 in showVisualization:
            _, probImprovements = AcquisitionFunctionProblImprovement(sampleXs, sampleYs, m, Xs, M
us, Sigmas)

#print("Learning Kernel Parameters.....")
# Acquisition Fuction을 통해 구한 다음 샘플링 포인트를 샘플에 추가함
sampleXs.append([nextX])
sampleY = 0
for i in range(8):
    S_X = 2 * np.divide(np.abs(np.multiply(nextX, pow(2,i)) - np.around(np.multiply(nextX, p
ow(2,i)))), pow(2,i))
    sampleY += S_X
sampleYs.append([sampleY + snr * np.random.randn()])
# kernel Learning을 시킬경우, 업데이트된 샘플로 parameter를 다시 학습시킴
if kernelLearning == True:
    trainedTheta, trainedBeta = KernelHyperParameterLearning(sampleXs, sampleYs)
    C, C_inv = KernelCalculation(trainedTheta, trainedBeta, len(sampleYs), sampleXs)
#print("Trained Kernel Parameters : ", trainedTheta, trainedBeta)

#print("iteration, probing point : ",itr2, " ",nextX)

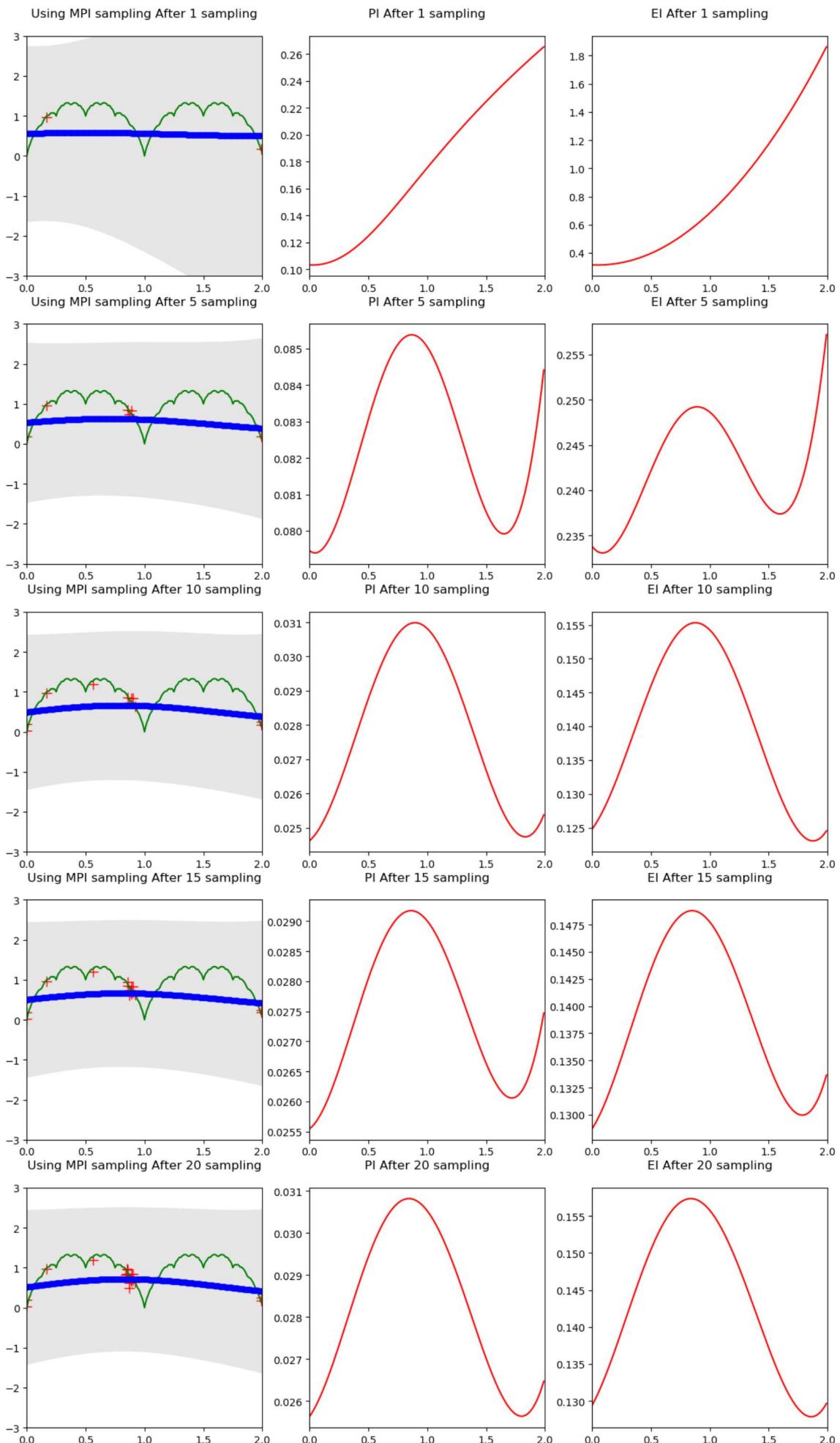
# showVisualization에 해당하는 itr2인 경우 아래와 같은 그래프를 그려줌
if itr2 in showVisualization:
    # sampling visualize
    plt.subplot(5, 3, plotN)
    plt.xlim([0, 2])
    plt.ylim([-3, 3])
    plt.title('Using MPI sampling After %s sampling' % (len(sampleYs)-1))
    plt.fill_between(Xs, Mus - 2 * np.sqrt(Sigmas), Mus + 2 * np.sqrt(Sigmas),
                     color=(0.9, 0.9, 0.9))
    plt.plot(sampleXs, sampleYs, 'r+', markersize=10) # training set은 빨강색 +로 표시함
    plt.plot(X, trueY, 'g-') # 실제값'수는 초록색 실선으로 나타냄
    plt.plot(Xs, Mus, 'bo-', markersize=5) # 예측값은 파란색 점으로 나타나며, 실선으로 0/1이
    plotN += 1

    # MEI
    plt.subplot(5, 3, plotN)
    plt.xlim([0, 2])
    plt.title('EI After %s sampling' % (len(sampleYs)-1))
    plt.plot(Xs, probImprovements, 'r-')
    plotN += 1

    # MEL
    plt.subplot(5, 3, plotN)
    plt.xlim([0, 2])
    plt.title('EI After %s sampling' % (len(sampleYs)-1))
    plt.plot(Xs, expectedImprovements, 'r-')

```

```
plotN += 1  
plt.show()
```



In [16]:

```

# Bayesian Optimization Result와 각 Acquisition Function 함수의 변화 비교
# 본 코드에서는 MEI(Maximum Expectation Improvement)로 Sampling을 진행함
snr = 0.1
X = np.arange(0, 2, 0.01)
numTruePoints = X.shape[0]
trueY = np.zeros(numTruePoints)
for i in range(8):
    S_X = np.multiply(np.divide(np.abs(np.multiply(X, pow(2,i)) - np.around(np.multiply(X, pow(2,i)))), pow(2,i)),2)
    trueY = np.add(trueY,S_X)

sampleXs = []
sampleYs = []

showVisualization = [0, 4, 9, 14, 19]
numTrials = showVisualization[4]+1
trainedTheta = np.array([1, 1, 1, 1])
trainedBeta = 1
m = 1
kernelLearning = True

# acquisitionFunction = 'Probability'
acquisitionFunction = 'ExpectedImprovement'

plt.figure(1, figsize=(14, 25), dpi=100)
plotN = 1

# 초기 X값 설정
for itr2 in range(1):
    sampleX = 2 * np.random.random()
    sampleXs.append([sampleX])
    sampleY = 0
    for i in range(8):
        S_X = 2 * np.divide(np.abs(np.multiply(sampleX, pow(2,i)) - np.around(np.multiply(sampleX, pow(2,i)))), pow(2,i))
        sampleY += S_X
    sampleYs.append([sampleY + snr * np.random.randn()])

# kernel Learning을 시킬경우, 아래의 조건문을 실행함
#print("Learning Kernel Parameters.....")
if kernelLearning == True:
    trainedTheta, trainedBeta = KernelHyperParameterLearning(sampleXs, sampleYs)
    C, C_inv = KernelCalculation(trainedTheta, trainedBeta, len(sampleYs), sampleXs)
#print("Trained Kernel Parameters : ", trainedTheta, trainedBeta)

# Bayesian Optimization 수행
for itr2 in range(numTrials):

    Xs = np.arange(0, 2, 0.01)
    Mus = []
    Sigmas = []

    #print("Calculating Predicted Values.....")
    for itr1 in range(len(Xs)):
        mu, var = PredictionGaussianProcessRegression(trainedTheta, trainedBeta, C_inv, len(sampleYs), sampleXs,
                                                       sampleYs, Xs[itr1])
        Mus.append(mu)
        Sigmas.append(var)

```

```

#print("Calculated Results : ",Mus,Sigmas)

# Acquisition Function에 따라 위에서 구한 파라미터를 대입하여 다음 샘플링 포인트를 찾아줌
#print("Calculating Acquisition Values.....")
if acquisitionFunction == 'ProblImprovement':
    nextX, probImprovements = AcquisitionFunctionProblImprovement(sampleXs, sampleYs, m, Xs, M
us, Sigmas)
    if itr2 in showVisualization:
        _, expectedImprovements = AcquisitionFunctionExpectedImprovement(sampleXs, sampleYs,
Xs, Mus, Sigmas)
    if acquisitionFunction == 'ExpectedImprovement':
        nextX, expectedImprovements = AcquisitionFunctionExpectedImprovement(sampleXs, sampleYs,
Xs, Mus, Sigmas)
        if itr2 in showVisualization:
            _, probImprovements = AcquisitionFunctionProblImprovement(sampleXs, sampleYs, m, Xs, M
us, Sigmas)

#print("Learning Kernel Parameters.....")
# Acquisition Fuction을 통해 구한 다음 샘플링 포인트를 샘플에 추가함
sampleXs.append([nextX])
sampleY = 0
for i in range(8):
    S_X = 2 * np.divide(np.abs(np.multiply(nextX, pow(2,i)) - np.around(np.multiply(nextX, p
ow(2,i)))), pow(2,i))
    sampleY += S_X
sampleYs.append([sampleY + snr * np.random.randn()])
# kernel Learning을 시킬경우, 업데이트된 샘플로 parameter를 다시 학습시킴
if kernelLearning == True:
    trainedTheta, trainedBeta = KernelHyperParameterLearning(sampleXs, sampleYs)
    C, C_inv = KernelCalculation(trainedTheta, trainedBeta, len(sampleYs), sampleXs)
#print("Trained Kernel Parameters : ", trainedTheta, trainedBeta)

#print("iteration, probing point : ",itr2, " ",nextX)

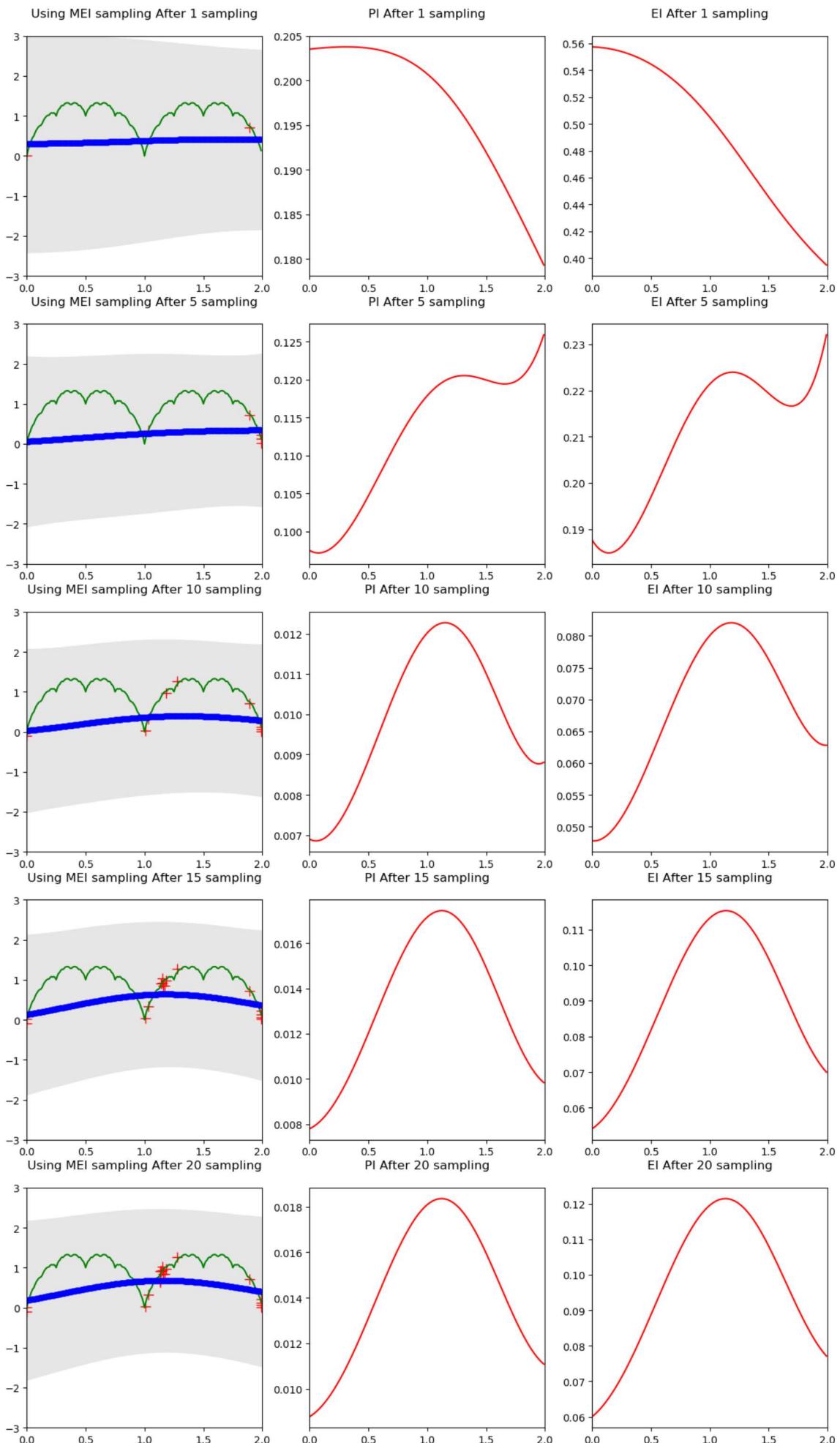
# showVisualization에 해당하는 itr2인 경우 아래와 같은 그래프를 그려줌
if itr2 in showVisualization:
    # sampling visualize
    plt.subplot(5, 3, plotN)
    plt.xlim([0, 2])
    plt.ylim([-3, 3])
    plt.title('Using MEI sampling After %s sampling' % (len(sampleYs)-1))
    plt.fill_between(Xs, Mus - 2 * np.sqrt(Sigmas), Mus + 2 * np.sqrt(Sigmas),
                     color=(0.9, 0.9, 0.9))
    plt.plot(sampleXs, sampleYs, 'r+', markersize=10) # training set은 빨강색 +로 표시함
    plt.plot(X, trueY, 'g-') # 실제값은 초록색 실선으로 나타냄
    plt.plot(Xs, Mus, 'bo-', markersize=5) # 예측값은 파란색 점으로 나타나며, 실선으로 0/1이
    plotN += 1

    # MPI
    plt.subplot(5, 3, plotN)
    plt.xlim([0, 2])
    plt.title('PI After %s sampling' % (len(sampleYs)-1))
    plt.plot(Xs, probImprovements, 'r-')
    plotN += 1

    # MEI
    plt.subplot(5, 3, plotN)
    plt.xlim([0, 2])
    plt.title('EI After %s sampling' % (len(sampleYs)-1))
    plt.plot(Xs, expectedImprovements, 'r-')

```

```
    plotN += 1  
    plt.show()
```



결과해석부

위 단락은 Maximum Probability Improvement(MPI)를 기준으로 샘플링을 진행한 과정이고, 아래 단락은 Maximum Expected Improvement(MEI)를 기준으로 샘플링을 진행한 결과입니다. 각 그래프의 첫 번째 열에서 초록색 실선은 실제함수를, 빨강색 + 점은 샘플링 값을, 파랑색 실선은 Acquisition Function을 통해 구한 샘플의 포인트 집합에서 가우시안 프로세스 리그레션을 바탕으로 구한 예측값(평균값)을 연결한 선입니다. 파랑색 실선 주위에 표시된 영역은 각 지점에서 예측된 함수의 분산을 나타냅니다. 위 그래프의 첫 번째 열을 통해 샘플링 횟수가 증가할수록 샘플링 포인트가 실제 함수의 최댓값과 가까워지는 것을 확인할 수 있습니다. 그래프의 두 번째 열은 샘플링 횟수에 따른 Probability Improvement(PI) 함수의 변화, 세 번째 열은 샘플링 횟수에 따른 Expected Improvement(EI) 함수의 변화를 나타낸 그래프입니다.