Chapter_12-Gaussian_Process

November 26, 2018

1 (Gaussian Process)

###

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

(Gaussian Process Regression) . $T_N = [t_1, t_2, ..., t_N]^T$, T_N t_{N+1} $P(t_{N+1}|T_N)$.

1.0.1 $P(T)$

N $X_n = [x_1, x_2, ..., x_n]^T$ M , M W, $Y_n = [y_1, y_2, ..., y_n]^T$, Y .

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

, $W = [w_1, w_2, ..., w_M]^T$, $K_{nm} = K(x_n, x_m) = \frac{1}{\alpha}\phi(x_n)\phi(x_m)$
x_n t_n, e_n , $t_n = y_n + e_n$. $P(T|Y)$.

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

$P(T)$.

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

1.0.2 $P(t_{N+1})$

T_N t_{N+1} T_{N+1} .

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

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

, .

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

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

1.0.3

.

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

```
In [1]: '''
        @ copyright: AAI lab (http://aailab.kaist.ac.kr/xe2/page_GBex27)
        @ 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]: #     (Tensorflow    parameter )
        def KernelFunctionWithTensorFlow(theta0, theta1, theta2, theta3, X1, X2):
            insideExp = tf.multiply(tf.div(theta1, 2.0), tf.matmul((X1 - X2), tf.transpose(X1 -
            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]: #     (  ,  )
        def KernelFunctionWithoutTensorFlow(theta, X1, X2):
            ret = theta[0] * np.exp(np.multiply(-theta[1] / 2.0, np.dot(np.subtract(X1, X2), np
                    + 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])

            #  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)

#
matCovarianceLinear = []
for i in range(numDataPoints):
    for j in range(numDataPoints):
        kernelEvaluationResult = KernelFunctionWithTensorFlow(theta0, theta1, theta
                                                              tf.slice(obsX, [i, 0]
                                                              tf.slice(obsX, [j, 0]
        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)

#
sumsquarederror = 0.0
for i in range(numDataPoints):
    k = tf.Variable(tf.ones([numDataPoints]))
    for j in range(numDataPoints):
        kernelEvaluationResult = KernelFunctionWithTensorFlow(theta0, theta1, theta
                                                              tf.slice(obsX, [i, 0]
                                                              tf.slice(obsX, [j, 0]
        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,
                                                          tf.slice(obsX, [i, 0], [
                                                          tf.slice(obsX, [i, 0], [

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

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

    #
    predictionMu = tf.matmul(k, tf.matmul(matCovarianceInv, obsY))
    predictionVar = tf.subtract(c, tf.matmul(k, tf.matmul(matCovarianceInv, tf.tran

    #   sum of squared error
    sumsquarederror = tf.add(sumsquarederror, tf.pow(tf.subtract(predictionMu, tf.s

# Training session declaration
```

```
        # Gradient Descent Optimizer  sum of squared error  parameter
        training = tf.train.GradientDescentOptimizer(0.1).minimize(sumsquarederror)

        # Session
        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)

        #   Session
        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: trainin
            trainedTheta.append(sess.run(theta1, feed_dict={obsX: trainingX, obsY: trainin
            trainedTheta.append(sess.run(theta2, feed_dict={obsX: trainingX, obsY: trainin
            trainedTheta.append(sess.run(theta3, feed_dict={obsX: trainingX, obsY: trainin
            trainedBeta = sess.run(beta, feed_dict={obsX: trainingX, obsY: trainingY})

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

        sess.close()

        #  parameter return
        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) #
```

In [7]:
```
trainingX = [] # training  X
trainingY = [] # training  Y

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], [5.749383898293203], [1.1471867
training set of Y:  [[-0.3137204147759303], [-0.30517652007738044], [-0.0727691236929981], [0.9
```

```
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
            k = np.zeros(numPoints)
            for i in range(numPoints):
                #  t_{N+1}  T_N     k
                k[i] = KernelFunctionWithoutTensorFlow(theta, sampleX[i], inputElement)

            #  t_{N+1}     c
            c = KernelFunctionWithoutTensorFlow(theta, inputElement, inputElement) + 1.0 / beta

            # P(t_n+1)
            mu = np.dot(k, np.dot(C_inv, sampleY))
            var = c - np.dot(k, np.dot(C_inv, k))

            return mu[0], var
```

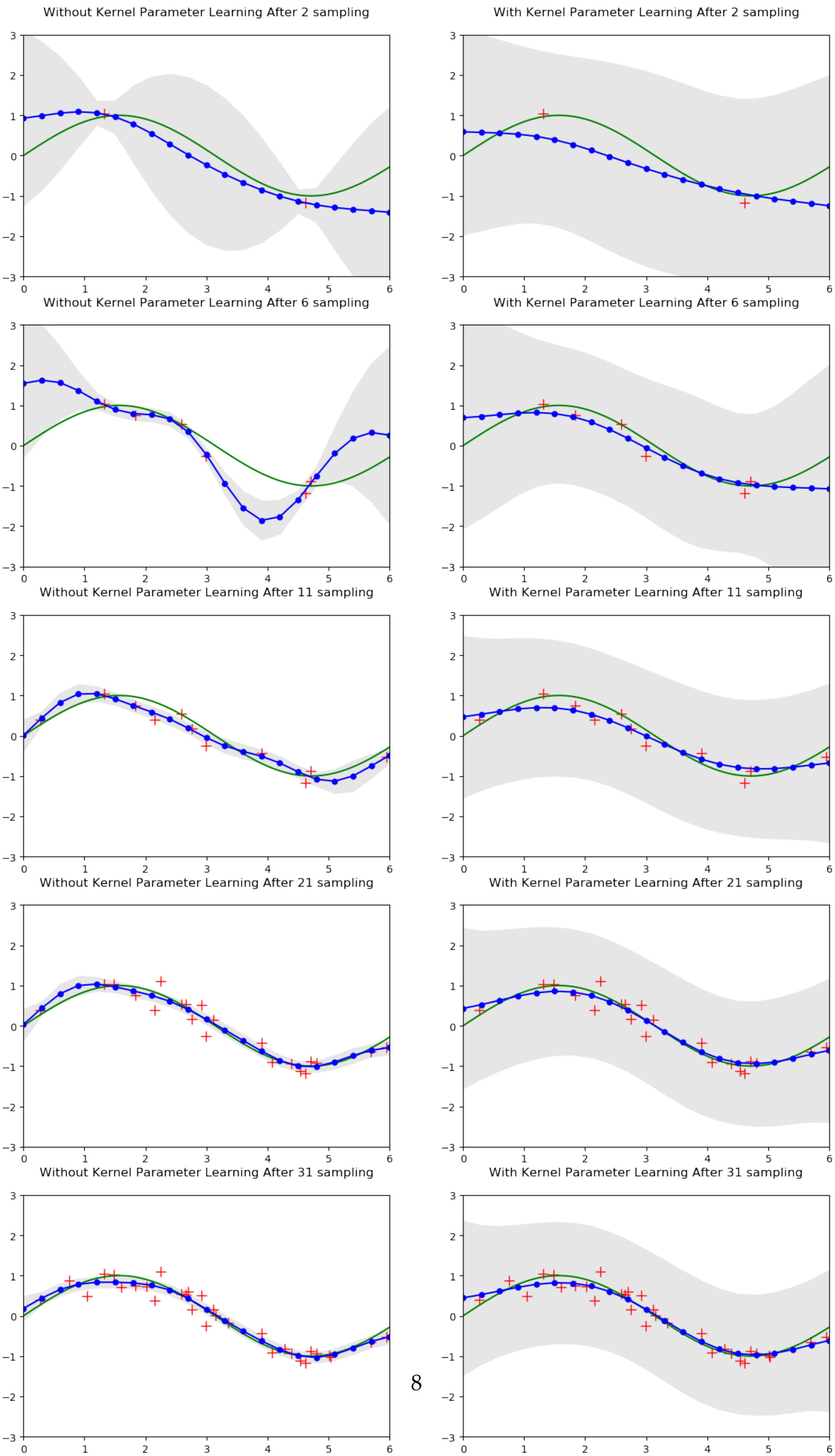
```
In [10]: #
         def KernelCalculation(theta, beta, numPoints, sampleX):
             C = np.zeros((numPoints, numPoints))

             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, s
             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.sqr
                         color=(0.9, 0.9, 0.9))
        plt.plot(sampleX, sampleY, 'r+', markersize=10) # training set  +
        plt.plot(X, trueY, 'g-') #
        plt.plot(inputs, mu_next, 'bo-', markersize=5) #    ,
```

In [12]:
```
## parameter
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()])
    # sampleY  (snr * np.random.randn())
    inputs = np.arange(0, 2 * np.pi, 0.3)

    # parameter
    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(sam
                                                          inputs[itr1])
            mu_next.append(mu)
            sigma2_next.append(var)

        PlottingGaussianProcessRegression(plotN,
                                          'Without Kernel Parameter Learning After %s
                                          inputs, mu_next, sigma2_next, sampleXs, sam
        plotN += 1

    # parameter
    mu_next = []
    sigma2_next = []
```

```
    if itr2 in showVisualization:
        trainedTheta, trainedBeta = KernelHyperParameterLearning(sampleXs, sampleYs)
        C, C_inv = KernelCalculation(trainedTheta, trainedBeta, len(sampleYs), sampleX

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

        PlottingGaussianProcessRegression(plotN, 'With Kernel Parameter Learning Afte
                                          inputs, mu_next, sigma2_next, sampleXs, samp
        plotN += 1

plt.show()
```


Without Kernel Parameter Learning After 2 sampling
With Kernel Parameter Learning After 2 sampling
Without Kernel Parameter Learning After 6 sampling
With Kernel Parameter Learning After 6 sampling
Without Kernel Parameter Learning After 11 sampling
With Kernel Parameter Learning After 11 sampling
Without Kernel Parameter Learning After 21 sampling
With Kernel Parameter Learning After 21 sampling
Without Kernel Parameter Learning After 31 sampling
With Kernel Parameter Learning After 31 sampling

1.0.4

, . tensorflow ( ) .
, + , 0.3 () . .
, .

2

, (input) . .

$$x^* = argmax_{x \in X} f(x)$$

, , (stochastic) . exploitation exploration . 'Acquisition Function' Maximum Probability Improvement Maximum Expected Improvement .

2.0.1 MPI (Maximum Probability Improvement)

D $f(x)$ y_{max}, x $y = f(x)$, y y_{max} margin m , x . .

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

2.0.2 MEI (Maximum Expected Improvement)

Maximum Expected Improvement margin m 0 m x . Maximum Probability of Improvement m . .

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

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

$$\mu = f\left(x \mid D\right), \sigma = K\left(x \mid D\right)$$

$$m = \max\left(0, yy_{max}\right) = \max\left(0, (vu)\,\sigma\right)$$

, Maximum Expected Improvement .

$$MEI\left(x \mid D\right) = argmax_x \int_0^\infty P(y_{max} + m) m dm = \frac{1}{2}\sigma^2 \left(-u\phi(u) + (1+u^2)\Phi(-u)\right)$$

```
In [13]: ## MPI
         def AcquisitionFunctionProbImprovement(sampleX, sampleY, m, Xs, Mus, Sigmas):
             numSamples = len(sampleY)

             idxMax = -1
             Ymax = -1000000

             #  Y      index
```

```
        for itr in range(numSamples):
            if sampleY[itr][0] > Ymax:
                Ymax = sampleY[itr][0]
                idxMax = itr

        #   probability of improvement
        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 Improvement    X
        for itr in range(len(Mus)):
            if probImprovements[itr] > Probmax:
                Probmax = probImprovements[itr]
                idxProbMax = itr

        return Xs[idxProbMax], probImprovements
```

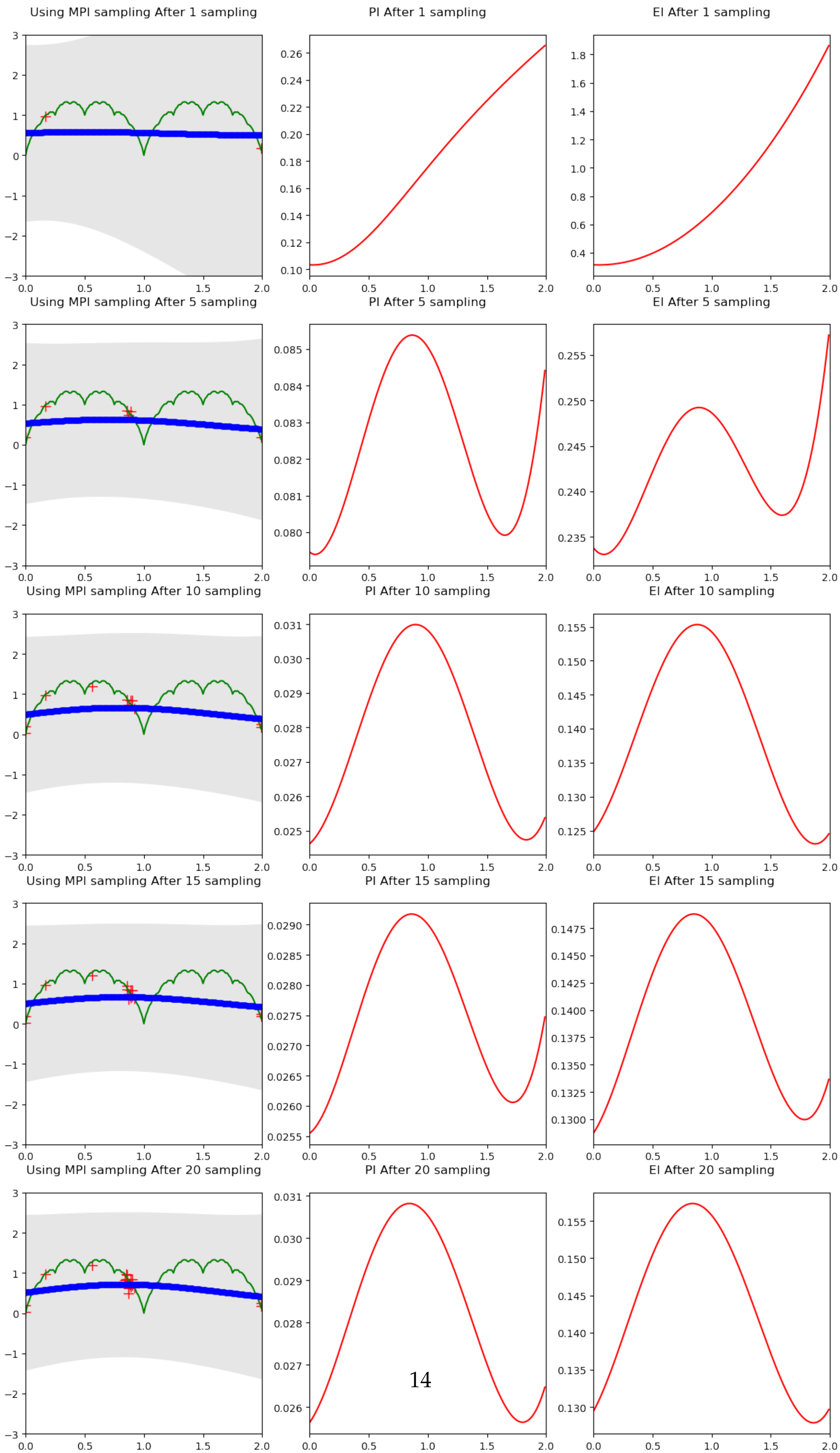
In [14]:
```
#MEI
def AcquisitionFunctionExpectedImprovement(sampleX, sampleY, Xs, Mus, Sigmas):
    numSamples = len(sampleY)

    idxMax = -1
    Ymax = -1000000

    #   Y      index
    for itr in range(numSamples):
        if sampleY[itr][0] > Ymax:
            Ymax = sampleY[itr][0]
            idxMax = itr

    #   expected improvement
    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)) *

    #print("Expected Improvements : ",expectedImprovements)

    idxEIMax = 0
    EImax = -1
    # Expected Improvement    X
    for itr in range(len(Mus)):
```

```
            if expectedImprovements[itr] > EImax:
                EImax = expectedImprovements[itr]
                idxEIMax = itr

        return Xs[idxEIMax], expectedImprovements
```

In [15]:
```
# Bayesian Optimization Result  Acquisition Function
#   MPI(Maximum Probability Improvment) Sampling
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.multipl
    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 = 'ProbImprovement'
#acquisitionFunction = 'ExpectedImprovement'

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

# X
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.multi
        sampleY += S_X
    sampleYs.append([sampleY + snr * np.random.randn()])

# kernel Learning ,
#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
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
                                                              sampleYs, Xs[itr1])
        Mus.append(mu)
        Sigmas.append(var)
    #print("Calculated Results : ",Mus,Sigmas)

    # Acquisition Function
    #print("Calculating Acquisition Values.........")
    if acquisitionFunction == 'ProbImprovement':
        nextX, probImprovments = AcquisitionFunctionProbImprovement(sampleXs, sampleYs
        if itr2 in showVisualization:
            _, expectedImprovments = AcquisitionFunctionExpectedImprovement(sampleXs,
    if acquisitionFunction == 'ExpectedImprovement':
        nextX, expectedImprovments = AcquisitionFunctionExpectedImprovement(sampleXs,
        if itr2 in showVisualization:
            _, probImprovments = AcquisitionFunctionProbImprovement(sampleXs, sampleYs

    #print("Learning Kernel Parameters.........")
    # Acquisition Fuction
    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.multipl
        sampleY += S_X
    sampleYs.append([sampleY + snr * np.random.randn()])
    # kernel Learning ,   parameter
    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)

    # showVisualization  itr2
    if itr2 in showVisualization:
        # sampling visulaize
        plt.subplot(5, 3, plotN)
        plt.xlim([0, 2])
```

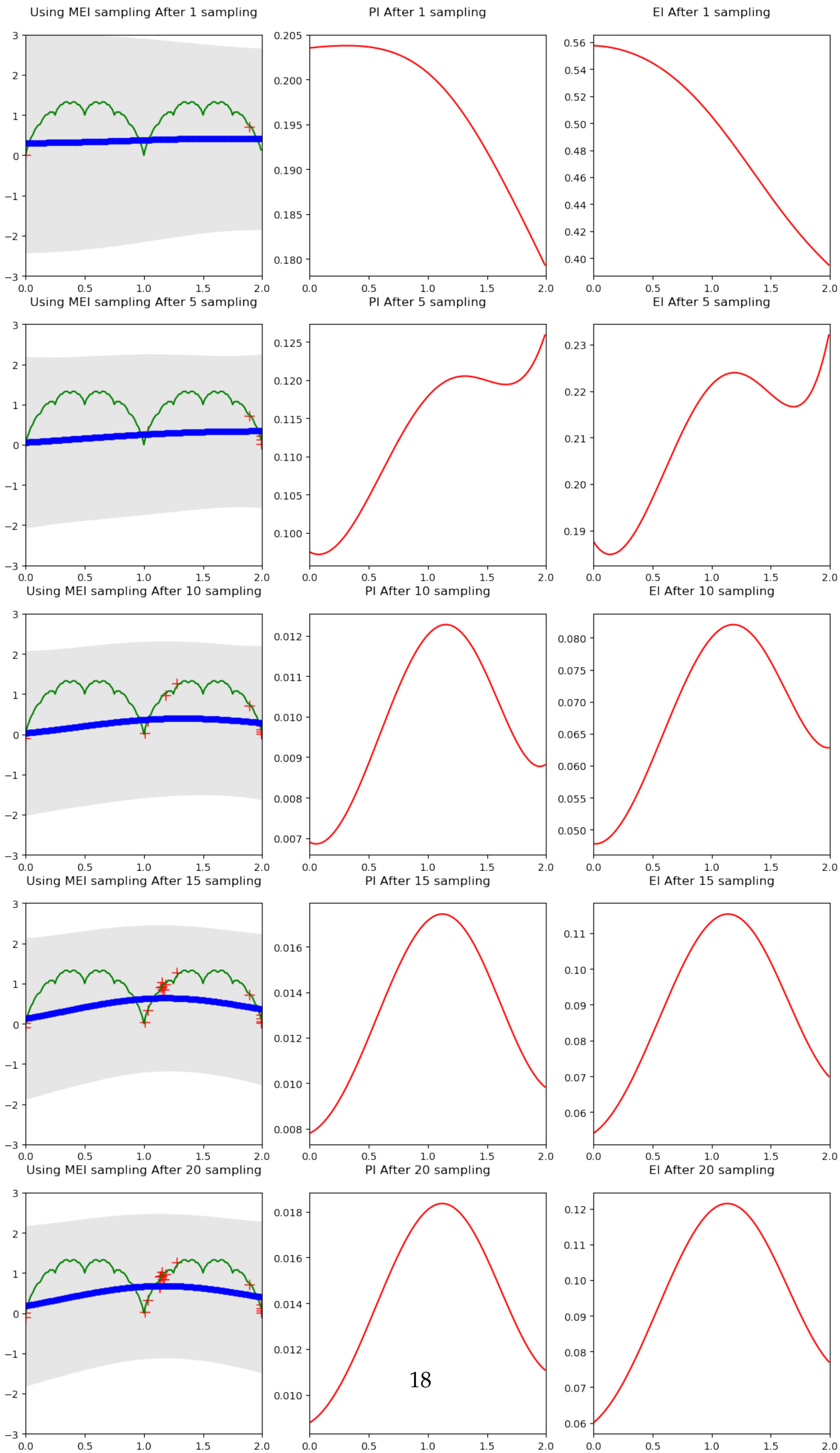
```
        plt.ylim([-3, 3])
        plt.title('Using MPI sampling After %s sampling\n' % (len(sampleYs)-1))
        plt.fill_between(Xs, Mus - 2 * np.sqrt(Sigmas), Mus + 2 * np.sqrt(Sigmas),
                     color=(0.9, 0.9, 0.9))
        plt.plot(sampleXs, sampleYs, 'r+', markersize=10) # training set  +
        plt.plot(X, trueY, 'g-') #
        plt.plot(Xs, Mus, 'bo-', markersize=5) #    ,
        plotN += 1

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

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

plt.show()
```


Using MPI sampling After 1 sampling
PI After 1 sampling
EI After 1 sampling
Using MPI sampling After 5 sampling
PI After 5 sampling
EI After 5 sampling
Using MPI sampling After 10 sampling
PI After 10 sampling
EI After 10 sampling
Using MPI sampling After 15 sampling
PI After 15 sampling
EI After 15 sampling
Using MPI sampling After 20 sampling
PI After 20 sampling
EI After 20 sampling

```
In [16]: # Bayesian Optimization Result  Acquisition Function
         #   MEI(Maximum Expectation Improvement) Sampling
         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.multipl
             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 = 'ProbImprovement'
         acquisitionFunction = 'ExpectedImprovement'

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

         # X
         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.multi
                 sampleY += S_X
             sampleYs.append([sampleY + snr * np.random.randn()])

         # kernel Learning ,
         #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
         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
                                                    sampleYs, Xs[itr1])
    Mus.append(mu)
    Sigmas.append(var)
#print("Calculated Results : ",Mus,Sigmas)

# Acquisition Function
#print("Calculating Acquisition Values.........")
if acquisitionFunction == 'ProbImprovement':
    nextX, probImprovments = AcquisitionFunctionProbImprovement(sampleXs, sampleYs
    if itr2 in showVisualization:
        _, expectedImprovments = AcquisitionFunctionExpectedImprovement(sampleXs,
if acquisitionFunction == 'ExpectedImprovement':
    nextX, expectedImprovments = AcquisitionFunctionExpectedImprovement(sampleXs,
    if itr2 in showVisualization:
        _, probImprovments = AcquisitionFunctionProbImprovement(sampleXs, sampleYs

#print("Learning Kernel Parameters.........")
# Acquisition Fuction
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.multipl
    sampleY += S_X
sampleYs.append([sampleY + snr * np.random.randn()])
# kernel Learning ,   parameter
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)

# showVisualization  itr2
if itr2 in showVisualization:
    # sampling visulaize
    plt.subplot(5, 3, plotN)
    plt.xlim([0, 2])
    plt.ylim([-3, 3])
    plt.title('Using MEI sampling After %s sampling\n' % (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 set   +
        plt.plot(X, trueY, 'g-') #
        plt.plot(Xs, Mus, 'bo-', markersize=5) #     ,
        plotN += 1

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

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

plt.show()
```


Using MEI sampling After 1 sampling
PI After 1 sampling
EI After 1 sampling
Using MEI sampling After 5 sampling
PI After 5 sampling
EI After 5 sampling
Using MEI sampling After 10 sampling
PI After 10 sampling
EI After 10 sampling
Using MEI sampling After 15 sampling
PI After 15 sampling
EI After 15 sampling
Using MEI sampling After 20 sampling
PI After 20 sampling
EI After 20 sampling

2.0.3

Maximum Probability Improvment(MPI) , Maximum Expected Improvment(MEI) . , + , Acquisition Fuction () . . . Probability Improvment(PI) , Expected Improvment(EI) .