

Gaussian Process

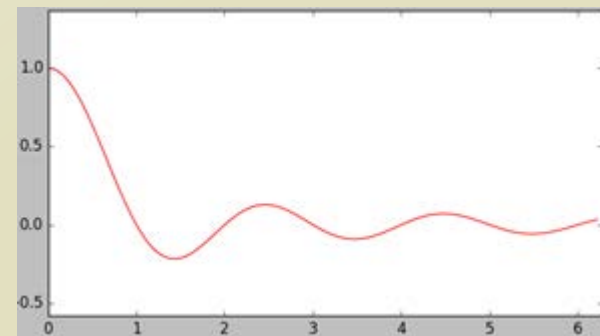
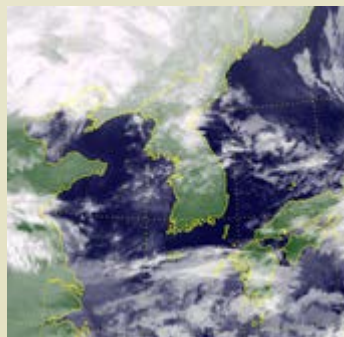
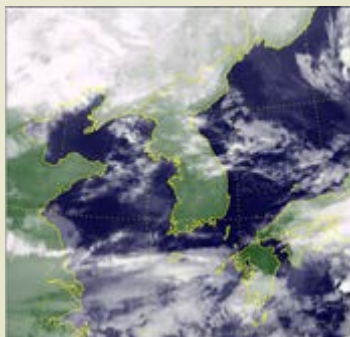
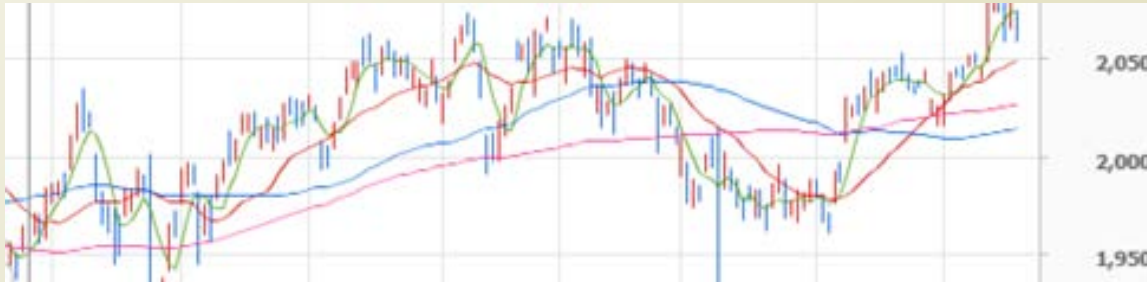
Il-Chul Moon
Dept. of Industrial and Systems Engineering
KAIST

icmoon@kaist.ac.kr

SIMPLE CONTINUOUS DOMAIN ANALYSIS

Continuous Domain Data

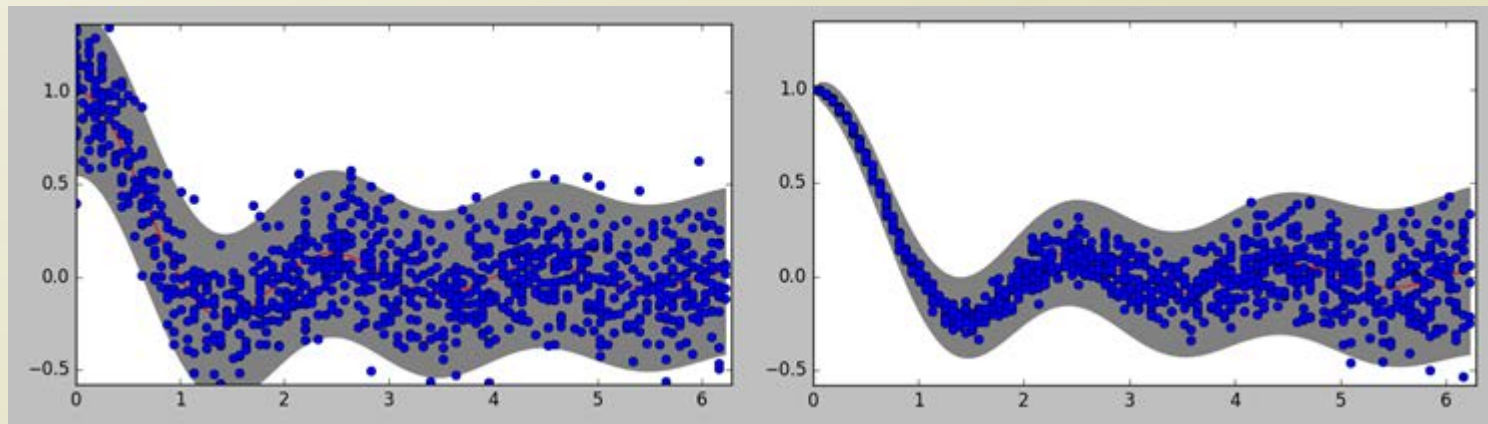
- Real-world, many continuous domain
 - Time, Space, Spatio-Temporal....
 - Discrete time vs. **Continuous time**
- How to analyze such dataset?
 - Estimation on the underlying function (ex, Autoregression)
 - Prediction on the unexplored point (ex, Extrapolation with autoregression)



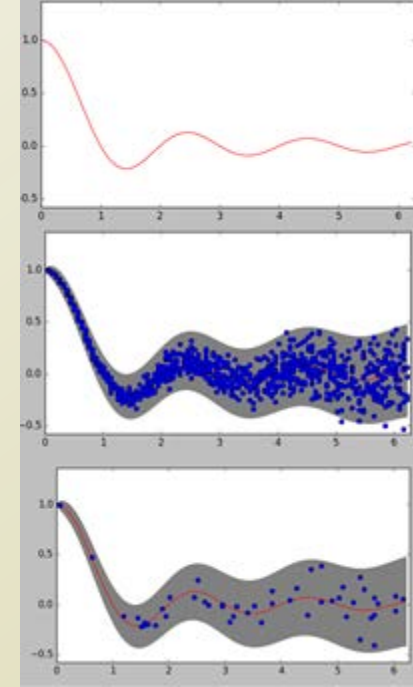
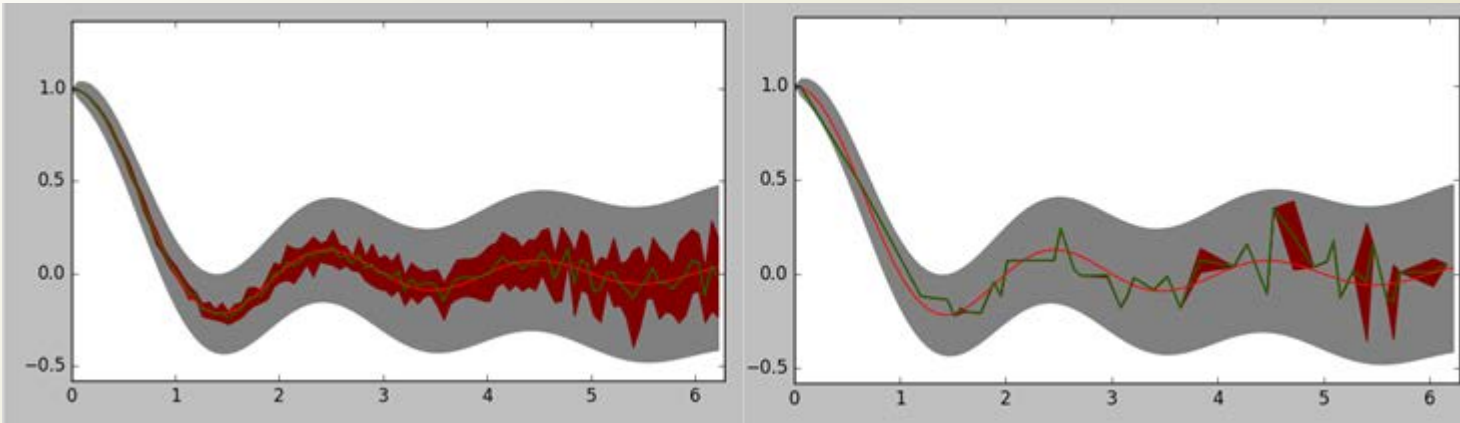
Underlying Function and Observations



- Simple temporal line does not say much
 - Two cases of different observations from the same temporal line
- An observation dataset can be explained with two temporal functions
 - Function in two continuous domain
 - Under the assumption that the observation's noise is generated from a Gaussian distribution
 - Mean function
 - Variance function, or precision function
- Previously, mean and variance was a value



Simple Analyses without Domain Correlation



- Estimating the mean function without the domain correlation
 - Calculating the mean and the precision of Y with the same X
- Very unlikely in the real world
 - Continuous domain \rightarrow No multiple observations with the same X
- No utilization of the domain information
 - Yesterday's observations might have some information on today's latent function

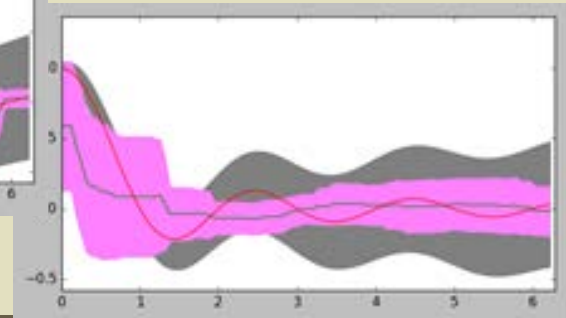
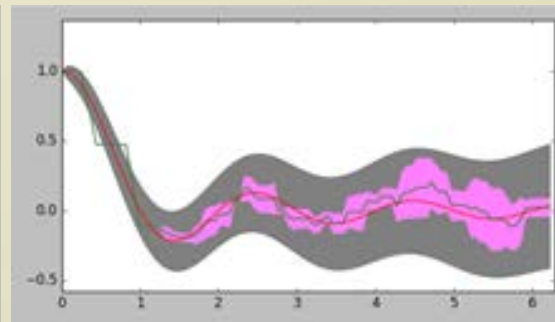
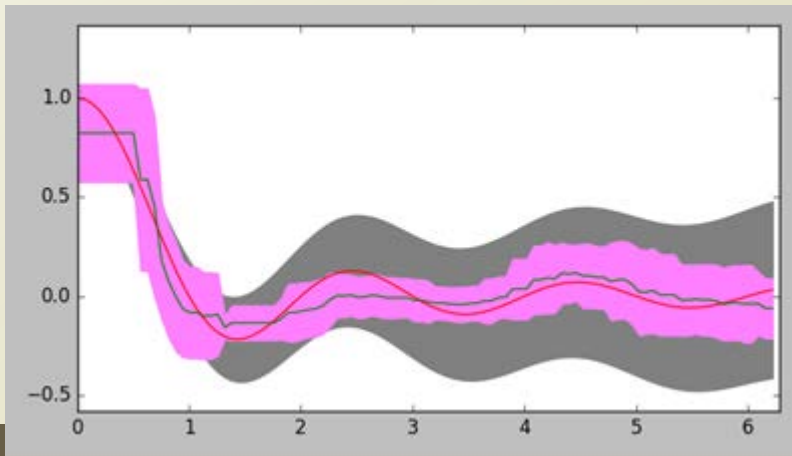
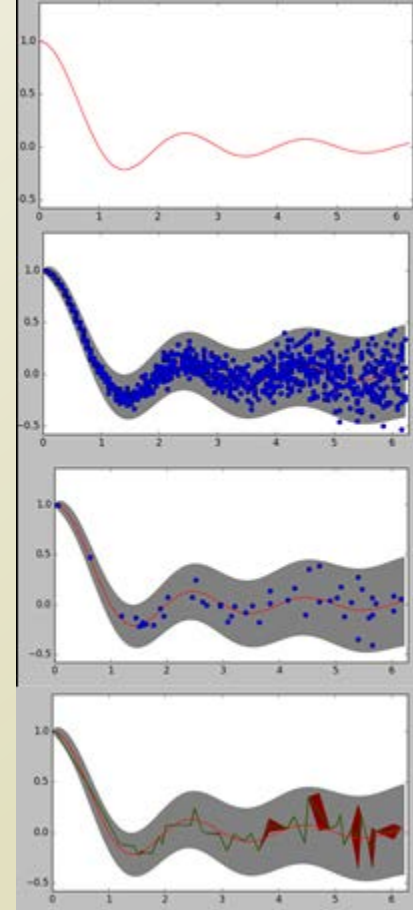
Simple Analyses with Domain Correlation

- Estimating the mean function with the domain correlation
 - Calculating the mean and the precision of Y with the correlated X
- Moving average with time-window $[w_{low}, w_{high}]$ and Dataset, D

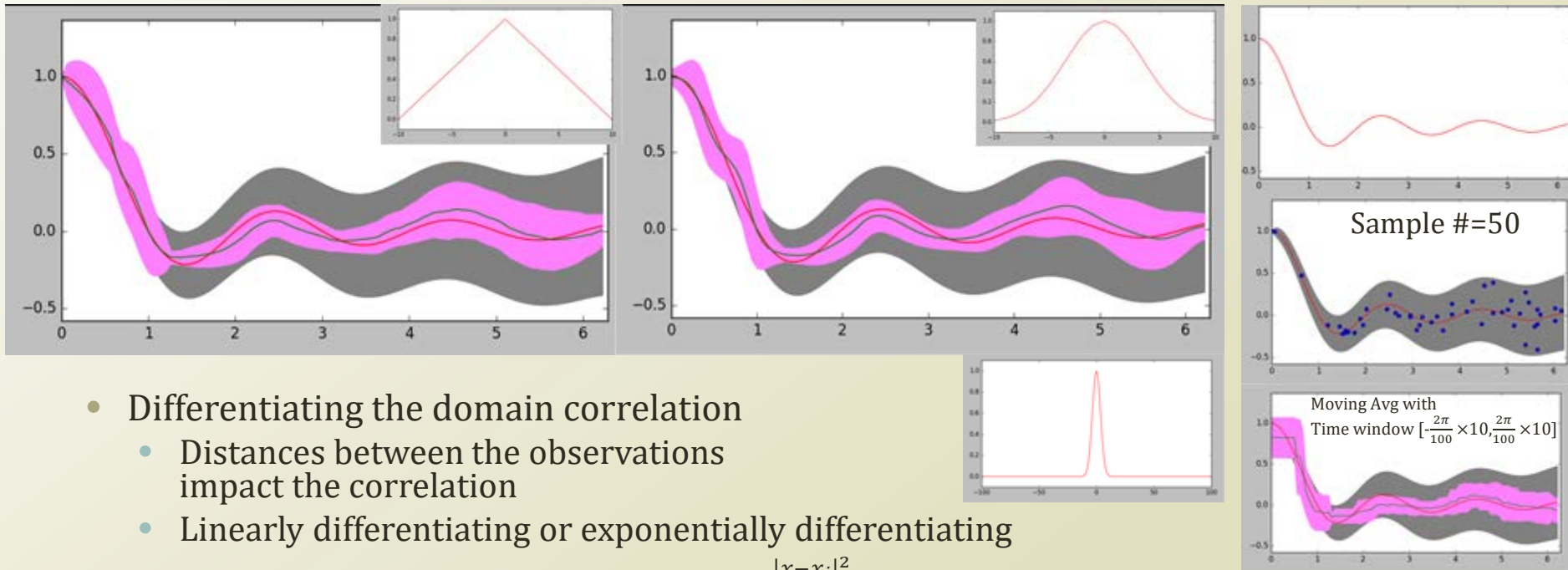
$$MA(x) = \frac{1}{N} \sum_{x_i \in W, D} y_i$$

$$W = [x - w_{low}, x + w_{high}], N = |\{x_i | x_i \in W, D\}|$$

- Simple moving average because it does not differentiate yesterday and 10 days ago



Simple Analyses with Differentiated Domain Correlation



- Differentiating the domain correlation
 - Distances between the observations impact the correlation
 - Linearly differentiating or exponentially differentiating
 - Squared Exponential : $k(x, x_i) = \exp(-\frac{|x-x_i|^2}{L^2})$
- Moving average with time-window $[w_{low}, w_{high}]$ and Dataset, D

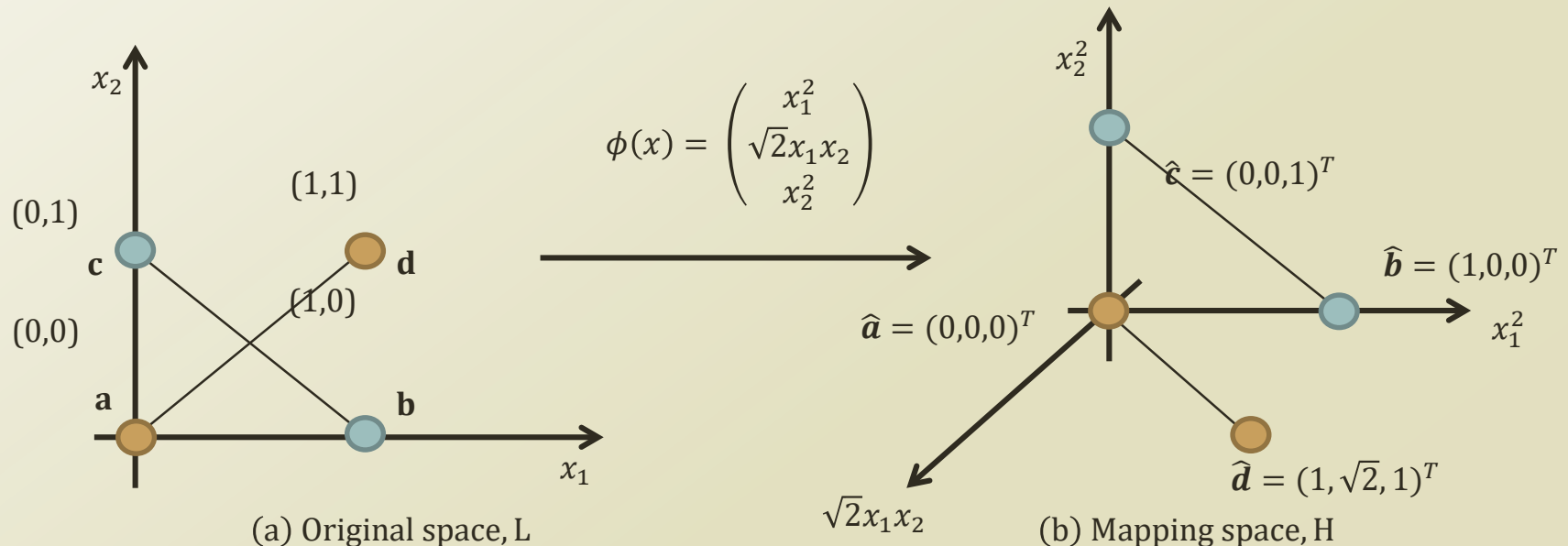
$$MA(x) = \frac{1}{\sum_{x_i \in W, D} k(x, x_i)} \sum_{x_i \in W, D} k(x, x_i) y_i, W = [x - w_{low}, x + w_{high}]$$

- How to determine such differentiation? Can we make a complex differentiation?

DERIVATION OF GAUSSIAN PROCESS

Detour: Mapping Functions

- Suppose that there are non-linearly separable data sets...
- The non-linear separable case can be linearly separable when we increase the basis space
 - Standard basis: $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3, \dots, \mathbf{e}_n \rightarrow$ Linearly independent and generate \mathbf{R}^n
- Expanding the Basis through Space mapping function $\phi : L \rightarrow H$
 - Or, transformation function, etc...
- Any problem????
 - Feature space becomes bigger and bigger....



Linear Regression with Basis Function

- Linear regression : $y(x) = w^T \phi(x)$
 - w : weight vector of M dimension
 - Or, $Y = \Phi w$
 - Φ : called a design matrix revealing the relation of the weight vector and the input vector
 - $\Phi_{nk} = \phi_k(x_n)$
- Previously, w is modeled as deterministic values
 - Now, w is considered to be also probabilistically distributed values
 - $P(w) = N(w|0, \alpha^{-1}I)$
 - Normal distribution with zero mean and α precision (or, α^{-1} variance)
- Now, w probability distribution $\rightarrow Y$ probability distribution
 - $E[Y] = E[\Phi w] = \Phi E[w] = 0$
 - $cov[Y] = E[(Y - 0)(Y - 0)^T] = E[YY^T]$
$$= E[\Phi w w^T \Phi^T] = \Phi E[w w^T] \Phi^T = \frac{1}{\alpha} \Phi \Phi^T$$
- $K_{nm} = k(x_n, x_m) = \frac{1}{\alpha} \phi(x_n)^T \phi(x_m)$
 - K : Gram matrix, k : kernel function
- $P(Y) = N(Y|0, K)$

Detour: Kernel Function

- The kernel calculates the inner product of two vectors in a different space (preferably without explicitly representing the two vectors in the different space)
 - $K(\mathbf{x}_i, \mathbf{x}_j) = \varphi(\mathbf{x}_i) \cdot \varphi(\mathbf{x}_j)$
- Some common kernels are following :
 - Polynomial(homogeneous)
 - $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j)^d$
 - Polynomial(inhomogeneous)
 - $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i \cdot \mathbf{x}_j + 1)^d$
 - Gaussian kernel function, a.k.a. Radial Basis Function
 - $k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\gamma \|\mathbf{x}_i - \mathbf{x}_j\|^2\right)$
 - For $\gamma > 0$. Sometimes parameterized using $\gamma = \frac{1}{2\sigma^2}$
 - Hyperbolic tangent, a.k.a. Sigmoid Function
 - $k(\mathbf{x}_i, \mathbf{x}_j) = \tanh(\kappa \mathbf{x}_i \cdot \mathbf{x}_j + c)$
 - For some(not every) $\kappa > 0$ and $c < 0$

Detour: Polynomial Kernel Function

- Imagine we have
 - $\mathbf{x} = \langle x_1, x_2 \rangle$ and $\mathbf{z} = \langle z_1, z_2 \rangle$
 - Polynomial Kernel Function of degree 1
 - $K(\langle x_1, x_2 \rangle, \langle z_1, z_2 \rangle) = \langle x_1, x_2 \rangle \cdot \langle z_1, z_2 \rangle = x_1 z_1 + x_2 z_2 = \mathbf{x} \cdot \mathbf{z}$
 - Polynomial Kernel Function of degree 2
 - $K(\langle x_1, x_2 \rangle, \langle z_1, z_2 \rangle) = \langle x_1^2, \sqrt{2}x_1x_2, x_2^2 \rangle \cdot \langle z_1^2, \sqrt{2}z_1z_2, z_2^2 \rangle$
 - $= x_1^2 z_1^2 + 2x_1x_2z_1z_2 + x_2^2 z_2^2 = (x_1 z_1 + x_2 z_2)^2 = (\mathbf{x} \cdot \mathbf{z})^2$
 - Polynomial Kernel Function of degree 3
 - $K(\langle x_1, x_2 \rangle, \langle z_1, z_2 \rangle) = (\mathbf{x} \cdot \mathbf{z})^3$
 - Polynomial Kernel Function of degree n
 - $K(\langle x_1, x_2 \rangle, \langle z_1, z_2 \rangle) = (\mathbf{x} \cdot \mathbf{z})^n$
- Do we need to express and calculate the transformed coordinate values for \mathbf{x} and \mathbf{z} to know the polynomial kernel of K ?
 - Do we need to convert the feature spaces to exploit the linear separation in the high order?
 - **Condition: only the inner product is computable with this trick**

Modeling Noise with Gaussian Distribution

- $P(Y) = N(Y|0, K)$
 - $K_{nm} = k(x_n, x_m) = \frac{1}{\alpha} \phi(x_n)^T \phi(x_m)$
- $t_n = y_n + e_n$
 - t_n : Observed value with noise
 - y_n : Latent, error-free value
 - e_n : Error term distributed by following the Gaussian distribution
- $P(t_n|y_n) = N(t_n|y_n, \beta^{-1})$
 - β : Hyper-parameter of the error precision (or, variance considering the invert)
- $P(T|Y) = N(T|Y, \beta^{-1}I_N)$
 - $T = (t_1, \dots, t_N)^T, Y = (y_1, \dots, y_N)^T$
 - Assuming that the error terms are independent
- $P(T) = \int P(T|Y)P(Y)dY = \int N(T|Y, \beta^{-1}I_N)N(Y|0, K)dY$

Marginal Gaussian Distribution

- $P(T) = \int P(T|Y)P(Y)dY = \int N(T|Y, \beta^{-1}I_N)N(Y|0, K)dY$
- $P(T|Y)P(Y) = P(T, Y) = P(Z)$
- $\ln P(Z) = \ln P(Y) + \ln P(T|Y)$

$$= -\frac{1}{2}(Y-0)^TK^{-1}(Y-0) - \frac{1}{2}(T-Y)^T\beta I_N(T-Y) + \text{const.}$$

$$= -\frac{1}{2}Y^TK^{-1}Y - \frac{1}{2}(T-Y)^T\beta I_N(T-Y) + \text{const.}$$
- Second order term of $\ln P(Z)$
 - $-\frac{1}{2}Y^TK^{-1}Y - \frac{\beta}{2}T^TT + \frac{\beta}{2}TY + \frac{\beta}{2}YT - \frac{\beta}{2}Y^TY$

$$= -\frac{1}{2}\begin{pmatrix} Y \\ T \end{pmatrix}^T \begin{pmatrix} K^{-1} + \beta I_N & -\beta I_N \\ -\beta I_N & \beta I_N \end{pmatrix} \begin{pmatrix} Y \\ T \end{pmatrix} = -\frac{1}{2}Z^TRZ$$
 - R becomes the precision matrix of Z
 - $M = (K^{-1} + \beta I_N - \beta I_N(\beta I_N)^{-1}\beta I_N)^{-1} = K$
 - $R^{-1} = \begin{pmatrix} K & K\beta I_N(\beta I_N)^{-1} \\ (\beta I_N)^{-1}\beta I_N K & (\beta I_N)^{-1} + (\beta I_N)^{-1}\beta I_N K\beta I_N(\beta I_N)^{-1} \end{pmatrix}$

$$= \begin{pmatrix} K & K \\ K & (\beta I_N)^{-1} + K \end{pmatrix}$$
- First order term of $\ln P(Z) \rightarrow \text{None}$
- $P(Z) = N(Z|0, R^{-1})$

$$N(x|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}} \frac{1}{|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)\right)$$

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix}^{-1} = \begin{pmatrix} M & -MBD^{-1} \\ -D^{-1}CM & D^{-1} + D^{-1}CMBD^{-1} \end{pmatrix}$$

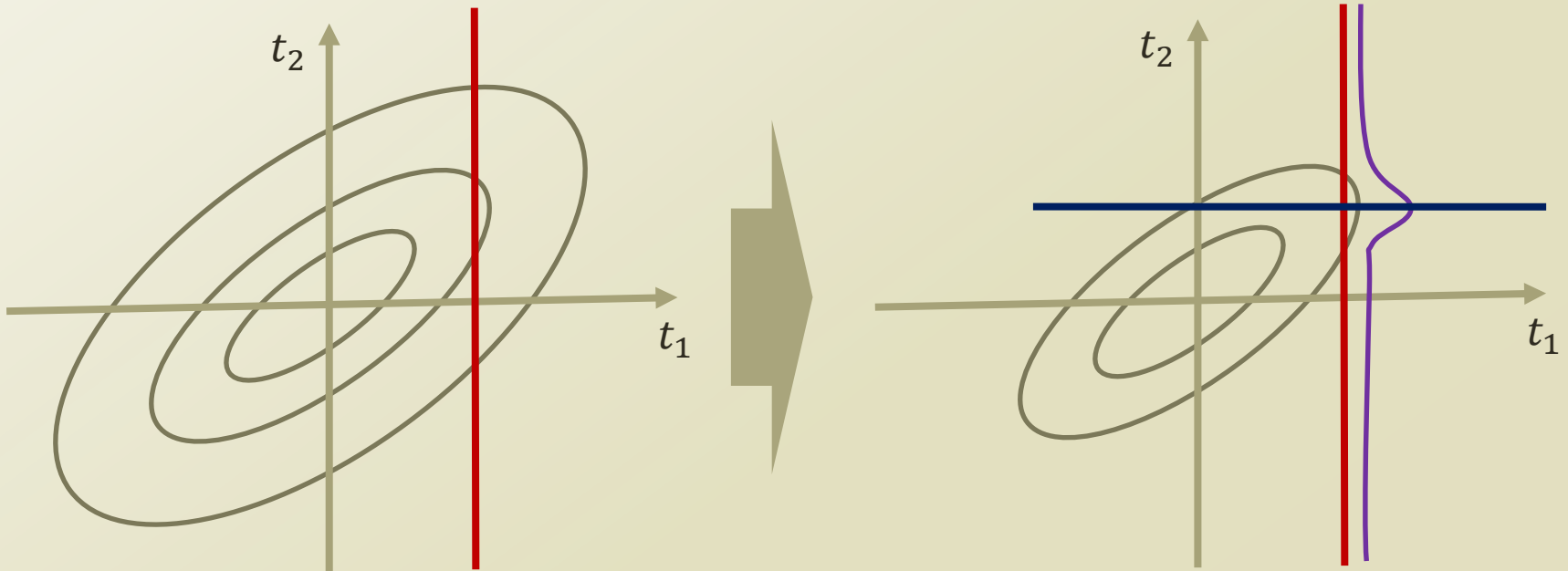
$$M = (A - BD^{-1}C)^{-1}$$

Marginal and Conditional Distribution of P(T)

- $P(T) = \int P(T|Y)P(Y)dY = \int N(T|Y, \beta^{-1}I_N)N(Y|0, K)dY$
 - $P(T|Y)P(Y) = P(Y, T) = P(Z)$
 - $P(Y, T) = N(Y, T|(0 \quad 0), \begin{pmatrix} K & K \\ K & (\beta I_N)^{-1} + K \end{pmatrix})$
 - Precision Matrix = $\begin{pmatrix} K^{-1} + \beta I_N & -\beta I_N \\ -\beta I_N & \beta I_N \end{pmatrix}$
- Two theorems on multivariate normal distributions
 - Given $X = [X_1 \quad X_2]^T, \mu = [\mu_1 \quad \mu_2]^T, \Sigma = \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix}$
 - $P(X_1) = N(X_1|\mu_1, \Sigma_{11})$
 - $P(X_1|X_2) = N(X_1|\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(X_2 - \mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})$
- $P(T) = N(T|0, (\beta I_N)^{-1} + K)$
 - $K_{nm} = k(x_n, x_m) = \frac{1}{\alpha} \phi(x_n)^T \phi(x_m)$
 - One example $\rightarrow k(x_n, x_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \|x_n - x_m\|^2\right) + \theta_2 + \theta_3 x_n^T x_m$
- Our ultimate question as a regression problem is
 - $P(t_{N+1}|T_N)=? \rightarrow P(T_{N+1})=!$

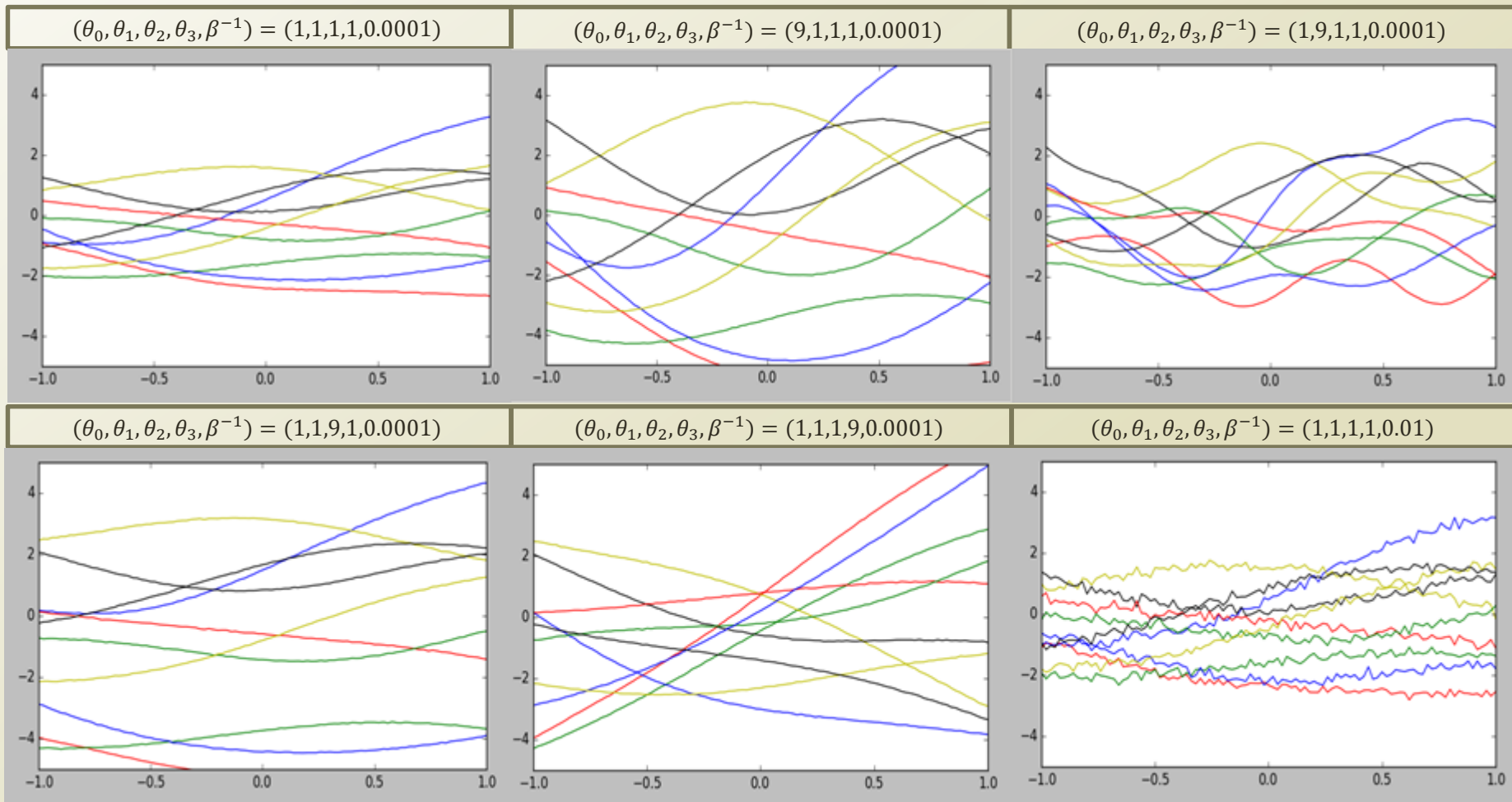
Prediction from Covariance

- $P(T) = N(T|0, (\beta I_N)^{-1} + K)$
 - T is a multi-dimensional random variable
 - $T = \langle t_1, t_2, \dots, t_n \rangle$
- What-if t_1 is already given?
 - $P(t_2|t_1) = ?$
 - The mean of $P(T)$ is zero, but t_1 can deviate from zero if sampled
 - Then, given non-zero t_1 , would the mean of t_2 becomes zero?
- The covariance structure provides the prediction of a dimension given another dimension



Sampling of P(T)

- $P(T) = N(T|0, (\beta I_N)^{-1} + K)$
 - $K_{nm} = k(x_n, x_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \|x_n - x_m\|^2\right) + \theta_2 + \theta_3 x_n^T x_m$
- Sampling T of 101 dimensions when points
 - when $x_n = [-1, -0.98 \dots, 0.98, 1]$ in $[-1, 1]$



Mean and Covariance of $P(t_{N+1}|T_N)$

- $P(T) = N(T|0, (\beta I_N)^{-1} + K)$

- $K_{nm} = k(x_n, x_m)$

- $P(T_{N+1}) = N(T|0, cov)$

$$cov = \begin{bmatrix} K_{11} + \beta^{-1} & K_{12} & \cdots & K_{1N} & K_{1(N+1)} \\ K_{21} & K_{22} + \beta^{-1} & \cdots & K_{2N} & K_{2(N+1)} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ K_{N1} & K_{N2} & \cdots & K_{NN} + \beta^{-1} & K_{N(N+1)} \\ K_{(N+1)1} & K_{(N+1)2} & \cdots & K_{(N+1)N} & K_{(N+1)(N+1)} + \beta^{-1} \end{bmatrix}$$

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

- Future distribution given the past data

- Remember the theorem introduced earlier

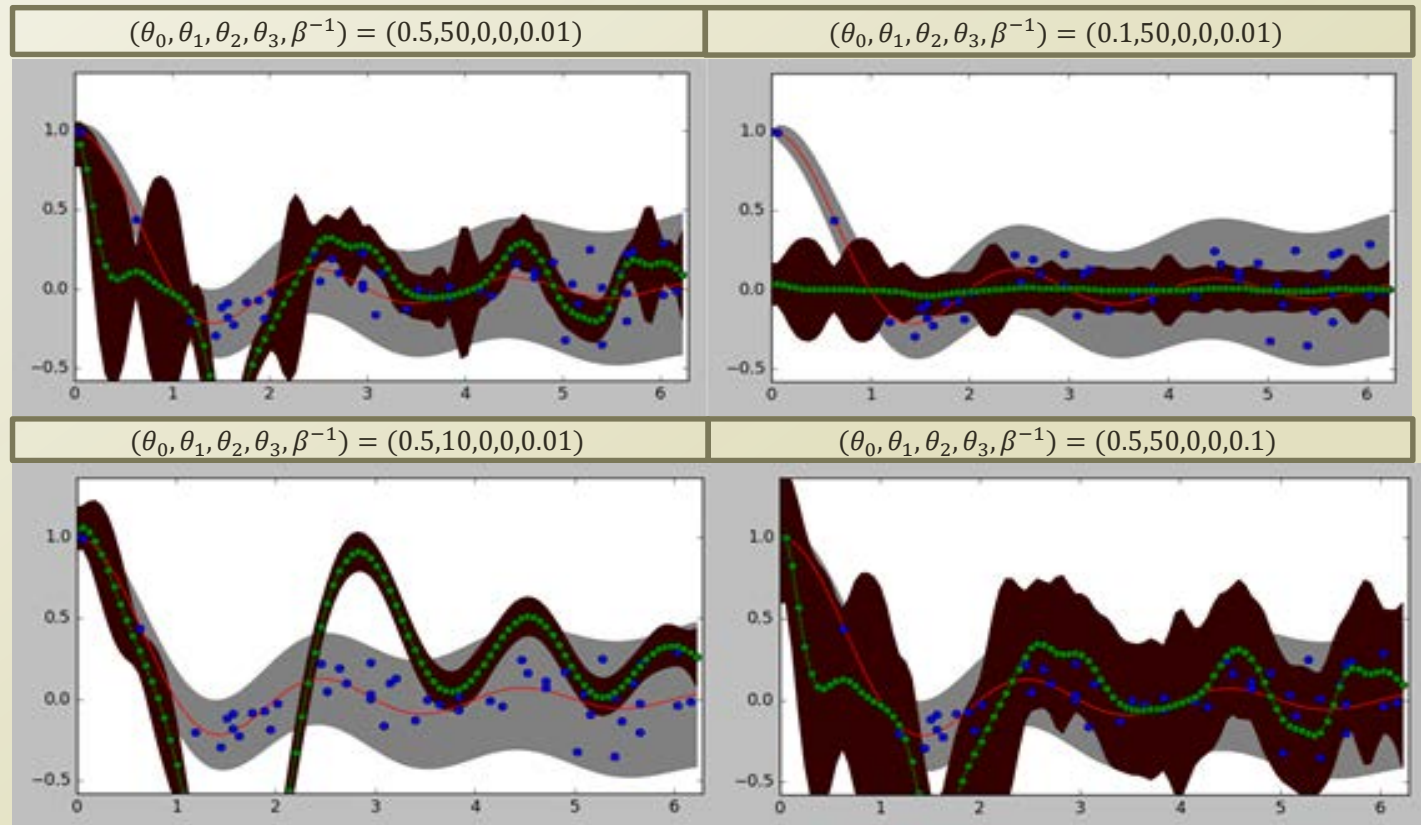
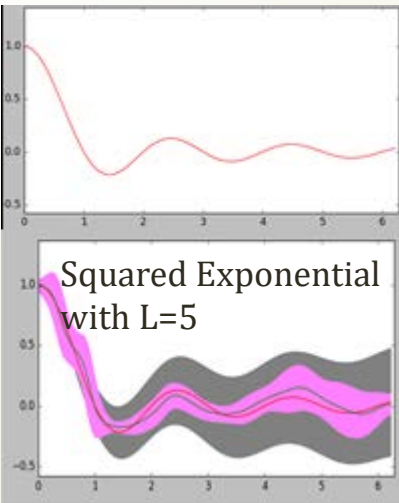
- $P(X_1|X_2) = N(X_1|\mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(X_2 - \mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21})$

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

- $\mu_{t_{N+1}} = k^T cov_N^{-1}T_N, \sigma^2_{t_{N+1}} = c - k^T cov_N^{-1}k$

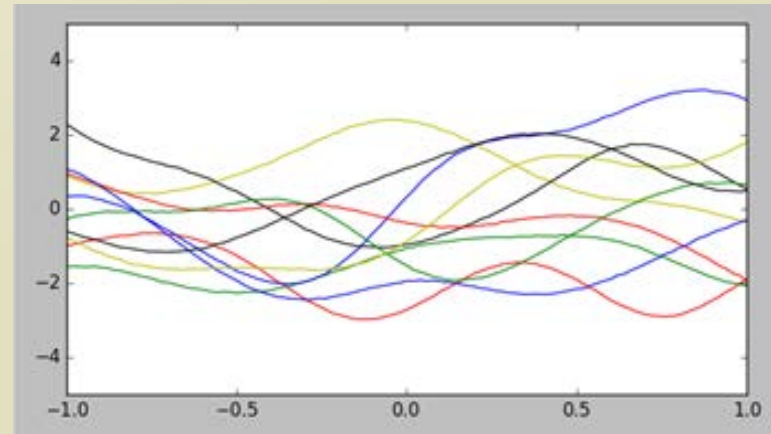
Gaussian Process Regression

- $P(t_{N+1}|T_N) = N(t_{N+1}|k^T cov_N^{-1}T_N, c - k^T cov_N^{-1}k)$
- Gaussian process regression
 - Models the predictive distribution given the past records, $P(t_{N+1}|T_N)$
 - Mean of the predictive distribution could be the most likely point estimation of the prediction
- $K_{nm} = k(x_n, x_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} ||x_n - x_m||^2\right) + \theta_2 + \theta_3 x_n^T x_m$



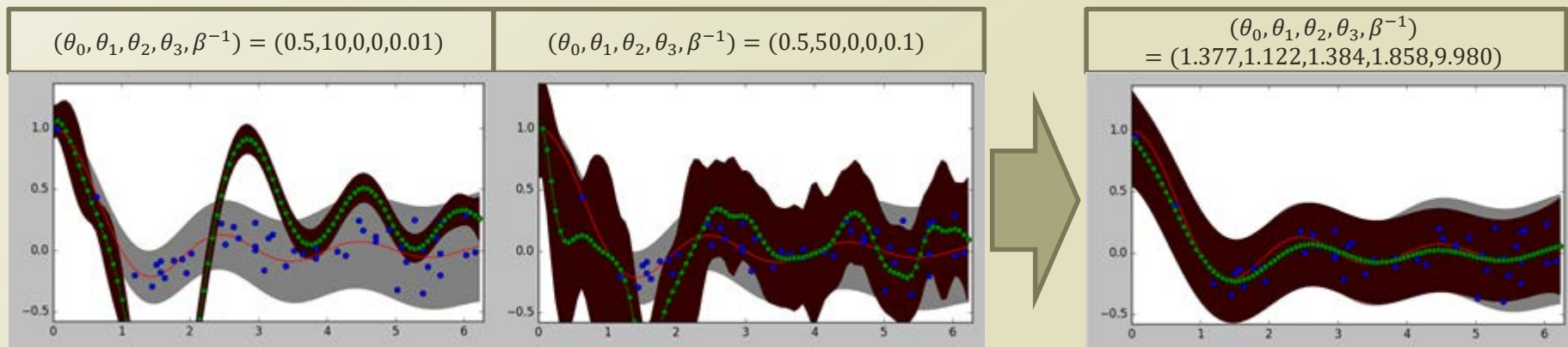
Random Process

- Random process, a.k.a. stochastic process, is
 - An infinite indexed collection of random variables, $\{X(t)|t \in T\}$
 - Index parameter : t
 - Can be time, space....
 - A function, $X(t, \omega)$, where $t \in T$ and $\omega \in \Omega$
 - Outcome of the underlying random experiment : ω
 - Fixed $t \rightarrow X(t, \omega)$ is a random variable over Ω
 - Fixed $\omega \rightarrow X(t, \omega)$ is a deterministic function of t , a sample function
- Example of random process
 - Gaussian process
 - $P(T) = N(T|0, (\beta I_N)^{-1} + K)$
 - $K_{nm} = k(x_n, x_m)$
$$= \theta_0 \exp\left(-\frac{\theta_1}{2} \|x_n - x_m\|^2\right) + \theta_2 + \theta_3 x_n^T x_m$$
 - Fixed t , a random variable following a Gaussian distribution
 - Fixed ω , a deterministic curve of t



Hyper-parameters of Gaussian Process Regression

- $K_{nm} = k(x_n, x_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \|x_n - x_m\|^2\right) + \theta_2 + \theta_3 x_n^T x_m$
- $P(T) = N(T|0, (\beta I_N)^{-1} + K) = N(T|0, C)$
 - Actually, $P(T|\theta)$
 - Need to learn $\theta \rightarrow$ Going back to the linear regression parameter optimization
 - $P(x|\mu, \Sigma) = (2\pi)^{-k/2} |\Sigma|^{-1/2} \exp(-\frac{1}{2} (x - \mu)' \Sigma^{-1} (x - \mu))$
- $\frac{\partial}{\partial \theta_i} \log P(T|\theta) = -\frac{1}{2} \text{Tr}\left(C_N^{-1} \frac{\partial C_N}{\partial \theta_i}\right) + \frac{1}{2} T^T C_N^{-1} \frac{\partial C_N}{\partial \theta_i} C_N^{-1} T$
 - Find θ to $\frac{\partial}{\partial \theta_i} P(T|\theta) = 0$
 - No closed form solution \rightarrow Need approximation; and Long derivation...
 - Or, we can use a probabilistic programming framework, i.e. Theano, TensorFlow....



Probabilistic Programming for Hyperparameter Learning of GP (1)

- $K_{nm} = k(x_n, x_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \|x_n - x_m\|^2\right) + \theta_2 + \theta_3 x_n^T x_m$
- $P(T) = N(T|0, (\beta I_N)^{-1} + K) = N(T|0, C)$

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

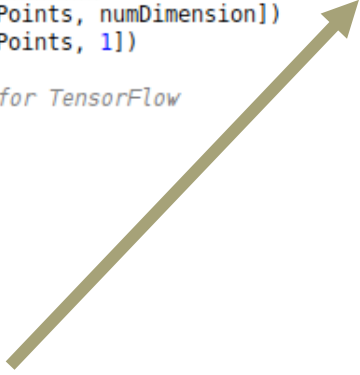
    # Input and Output Data Declaration for Tensorflow
    obsX = tf.placeholder(tf.float32, [numDataPoints, numDimension])
    obsY = tf.placeholder(tf.float32, [numDataPoints, 1])

    # Learning Parameter Variable Declaration 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)

    # Kernel Build
    matCovarianceLinear = []
    for i in range(numDataPoints):
        for j in range(numDataPoints):
            kernelEvaluationResult = KernelFunctionWithTensorFlow(theta0, theta1, theta2, theta3,
                                                                    tf.slice(obsX, [i, 0], [1, numDimension]),
                                                                    tf.slice(obsX, [j, 0], [1, numDimension]))

            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)
```



```
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
    thirdTerm = tf.multiply(theta3, tf.matmul(X1, tf.transpose(X2)))
    ret = tf.add(tf.add(firstTerm, secondTerm), thirdTerm)
    return ret
```

Probabilistic Programming for Hyperparameter Learning of GP (2)

- $K_{nm} = k(x_n, x_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} \|x_n - x_m\|^2\right) + \theta_2 + \theta_3 x_n^T x_m$
- $P(T) = N(T|0, (\beta I_N)^{-1} + K) = N(T|0, C)$
 $\mu_{t_{N+1}} = k^T \text{cov}_N^{-1} T_N, \sigma^2_{t_{N+1}} = c - k^T \text{cov}_N^{-1} k$

```
# Prediction
sumsquarederror = 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, numDimension]),
                                                                tf.slice(obsX, [j, 0], [1, numDimension]))

        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))

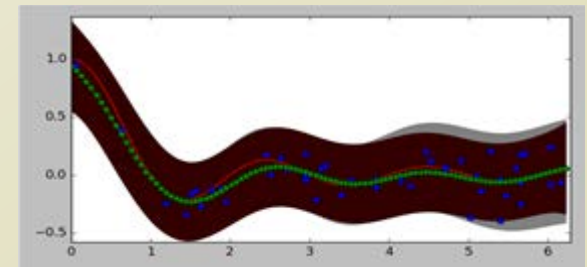
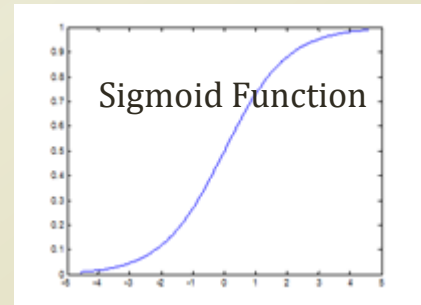
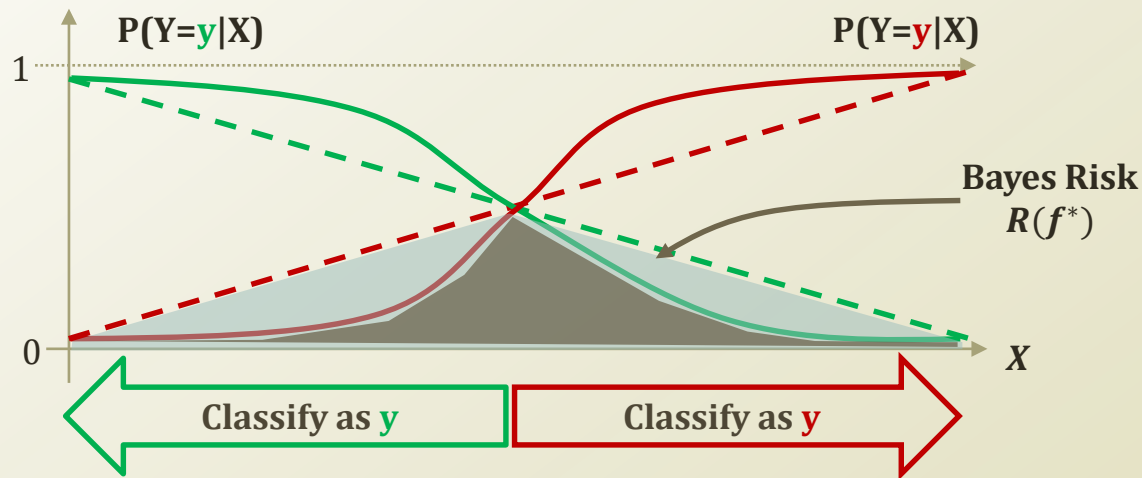
    k = tf.reshape(k, [1, numDataPoints])

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

    sumsquarederror = tf.add(sumsquarederror, tf.pow(tf.subtract(predictionMu, tf.slice(obsY, [i, 0], [1, 1])), 2))

# Training session declaration
training = tf.train.GradientDescentOptimizer(0.1).minimize(sumsquarederror)
```

Gaussian Process Classifier



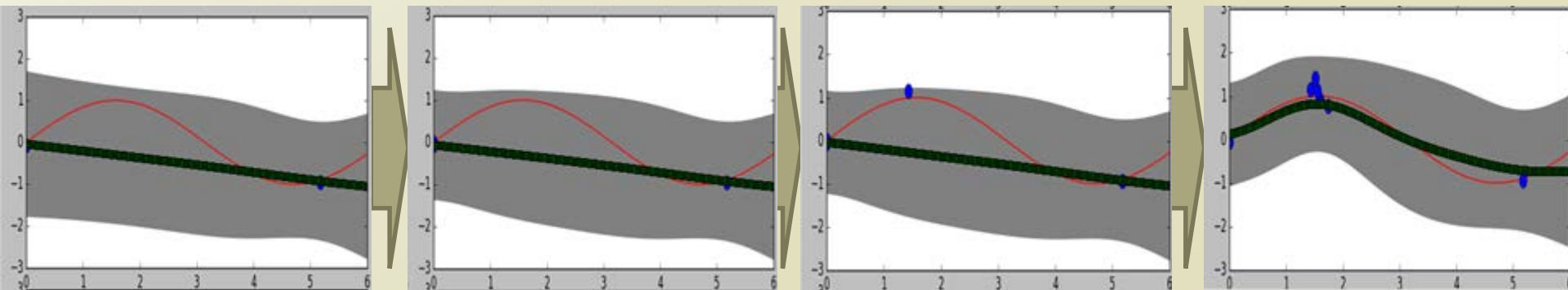
- Logistic regression
 - Sigmoid function(logistic function) + linear regression

$$P(y = 1|x) = \frac{1}{1 + e^{-\theta^T x}}$$

- Gaussian process classifier
 - Sigmoid function(logistic function) + Gaussian process regression
 - Gaussian process : $f(x; \theta) \rightarrow$ Gaussian process classifier : $y = \sigma(f(x; \theta))$
 - If $t \in \{0,1\}$, then the objective function to optimize
 - $P(t|\theta) = \sigma(f(x; \theta))^t (1 - \sigma(f(x; \theta)))^{1-t}$

Bayesian Optimization with Gaussian Process

- Imagine we have a sequence of experiments that we can set the input as we want
 - The experiment result should be maximized
 - We don't know the underlying function generating the experiment results
 - The result and the input are continuous
 - The result have a stochastic element
- Previous approaches include search methods
 - Grid search : no learning of underlying function
 - Fixed sampling inputs
 - Binary search : learning of constraints, not the function
 - Adaptively change sampling inputs
- Integration of learning underlying function and selecting the next sampling input



Acquisition Function:

Maximum Probability of Improvement

- Acquisition function
 - Gaussian process provides the predicted mean and the predicted std. on any point
 - Any point \rightarrow Next sampling
 - Predicted mean \rightarrow potential optimized value
 - Predicted std. \rightarrow potential risk of getting a value deviating from the mean
 - Need a policy for sampling, and this policy is the acquisition function
- Maximum probability of improvement
 - Selects a sampling input with the highest probability of improving the current optimized value, y_{max} , with some margin, m

$$\begin{aligned} MPI(x|D) &= \operatorname{argmax}_x P(y \geq (1+m)y_{max} | x, D), \quad y \sim N(\mu, \sigma^2) \\ &= \operatorname{argmax}_x P\left(\frac{y - \mu}{\sigma} \geq \frac{(1+m)y_{max} - \mu}{\sigma}\right) \\ &= \operatorname{argmax}_x \left\{ 1 - \Phi\left(\frac{(1+m)y_{max} - \mu}{\sigma}\right) \right\} \\ &= \operatorname{argmax}_x \Phi\left(\frac{\mu - (1+m)y_{max}}{\sigma}\right) \end{aligned}$$

Acquisition Function:

Maximum Expected Improvement

- Maximum expected improvement
 - A problem of maximum probability of improvement is
 - Introducing another hyperparameter, m
 - Why not take an expectation over the range of m which is from 0 to infinite
- Assumption

$$\chi_R(\tilde{v}) = \begin{cases} 1, & \tilde{v} \in R \\ 0, & \tilde{v} \notin R \end{cases}$$

- $y = f(x), y_{\max} = \max_{m=1, \dots, n} f(x_m), u = \frac{y_{\max} - \mu}{\sigma}, v = \frac{y - \mu}{\sigma}, \mu = f(x|\mathcal{D}), \sigma = K(x|\mathcal{D})$
- $MEI(x|D) = \operatorname{argmax}_x \int_0^\infty P(y \geq y_{\max} + m) m \, dm$
- $\int_0^\infty P(y \geq y_{\max} + m) m \, dm = \int_0^\infty P\left(\frac{y - \mu}{\sigma} \geq \frac{y_{\max} - \mu + m}{\sigma}\right) m \, dm = \int_0^\infty P\left(v \geq u + \frac{m}{\sigma}\right) m \, dm$
- $= \int_0^\infty \int_{u + \frac{m}{\sigma}}^\infty \phi(\tilde{v}) d\tilde{v} m \, dm = \int_0^\infty \int_0^\infty \chi_{[u + \frac{m}{\sigma}, \infty)}(\tilde{v}) \phi(\tilde{v}) d\tilde{v} m \, dm = \int_0^\infty \int_0^\infty m \chi_{[u + \frac{m}{\sigma}, \infty)}(\tilde{v}) \phi(\tilde{v}) dv \, dm =$
 $\int_0^\infty \int_0^\infty m \chi_{[u + \frac{m}{\sigma}, \infty)}(\tilde{v}) \phi(\tilde{v}) dm \, dv = \int_0^\infty \left\{ \int_0^\infty m \chi_{[u + \frac{m}{\sigma}, \infty)}(\tilde{v}) dm \right\} \phi(\tilde{v}) d\tilde{v} =$
 $\int_0^\infty \left\{ \int_0^\infty m \chi_{\{0 \leq m \leq \sigma(\tilde{v} - u)\}}(\tilde{v}) dm \right\} \phi(\tilde{v}) d\tilde{v} = \int_0^\infty \left\{ \int_0^\infty m \chi_{\{0 \leq m \leq \sigma(\tilde{v} - u)\} \cap \{0 \leq \sigma(\tilde{v} - u)\}}(\tilde{v}) dm \right\} \phi(\tilde{v}) d\tilde{v} =$
 $\int_0^\infty \left\{ \int_0^\infty m \chi_{\{0 \leq m \leq \sigma(\tilde{v} - u)\}}(\tilde{v}) \chi_{\{0 \leq \sigma(\tilde{v} - u)\}}(\tilde{v}) dm \right\} \phi(\tilde{v}) d\tilde{v} = \int_0^\infty \left\{ \int_0^\infty m \chi_{\{0 \leq m \leq \sigma(\tilde{v} - u)\}}(\tilde{v}) dm \right\} \chi_{\{0 \leq \sigma(\tilde{v} - u)\}}(\tilde{v}) \phi(\tilde{v}) d\tilde{v} =$
 $\int_u^\infty \left\{ \int_0^{\sigma(\tilde{v} - u)} m dm \right\} \phi(\tilde{v}) d\tilde{v} = \int_u^\infty \frac{1}{2} \sigma^2 (\tilde{v} - u)^2 \phi(\tilde{v}) d\tilde{v} = \frac{1}{2} \sigma^2 \left[\int_u^\infty \tilde{v}^2 \phi(\tilde{v}) d\tilde{v} - 2u \int_u^\infty \tilde{v} \phi(\tilde{v}) d\tilde{v} + \right.$
 $\left. u^2 \int_u^\infty \phi(\tilde{v}) d\tilde{v} \right] = \frac{1}{2} \sigma^2 [-u\phi(u) + (1 + u^2)\Phi(-u)]$
 - $\therefore \int_u^\infty \tilde{v}^2 \phi(\tilde{v}) d\tilde{v} = \int_u^\infty \frac{1}{\sqrt{2\pi}} \tilde{v}^2 \exp\left(-\frac{\tilde{v}^2}{2}\right) d\tilde{v} = \left[-\frac{1}{\sqrt{2\pi}} \tilde{v} \exp\left(-\frac{\tilde{v}^2}{2}\right)\right]_u^\infty + \int_u^\infty \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\tilde{v}^2}{2}\right) d\tilde{v} = \left[-\frac{1}{\sqrt{2\pi}} \tilde{v} \exp\left(-\frac{\tilde{v}^2}{2}\right)\right]_u^\infty +$
 $\int_{-\infty}^{-u} \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\tilde{v}^2}{2}\right) d\tilde{v} = u\phi(u) + \Phi(-u)$
 - $\int_u^\infty \tilde{v} \phi(\tilde{v}) d\tilde{v} = \int_u^\infty \frac{1}{\sqrt{2\pi}} \tilde{v} \exp\left(-\frac{\tilde{v}^2}{2}\right) dv = \left[-\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\tilde{v}^2}{2}\right)\right]_u^\infty = \phi(u)$

Bayesian Optimization Result

Sampling, Learned Function...

Prob. Of Improvement

Expected Improvement

- A case of Bayesian optimization
 - Sampling based upon the maximum expected improvement

