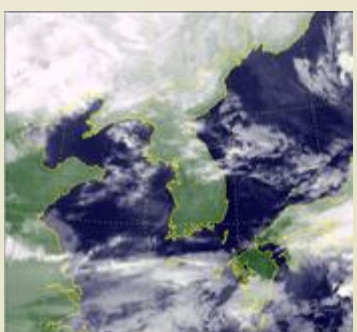
Gaussian Process

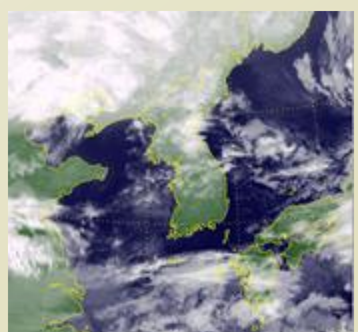
Il-Chul Moon
Dept. of Industrial and Systems Engineering
KAIST
icmoon@kaist.ac.kr

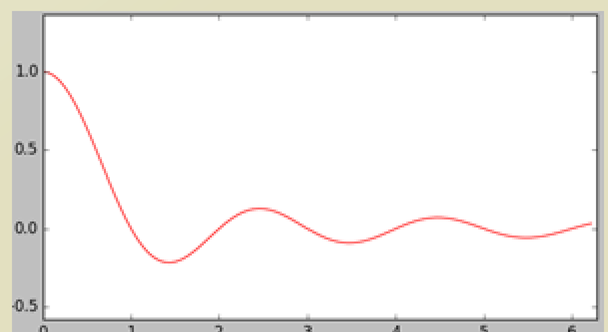
SIMPLE CONTINUOUS DOMAIN ANALYSIS

Continuous Domain Data

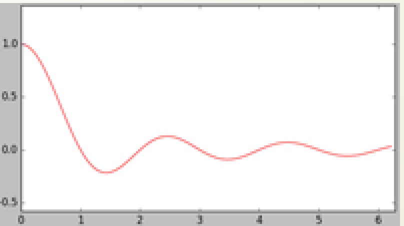
- Real-world, many continuous domain
  - Time, Space, Spatio-Temporal….
    - Discrete time vs. **Continuous time**
- How to analyze such dataset?
  - Estimation on the underlying function (ex, Autoregression)
  - Prediction on the unexplored point (ex, Extrapolation with autoregression)



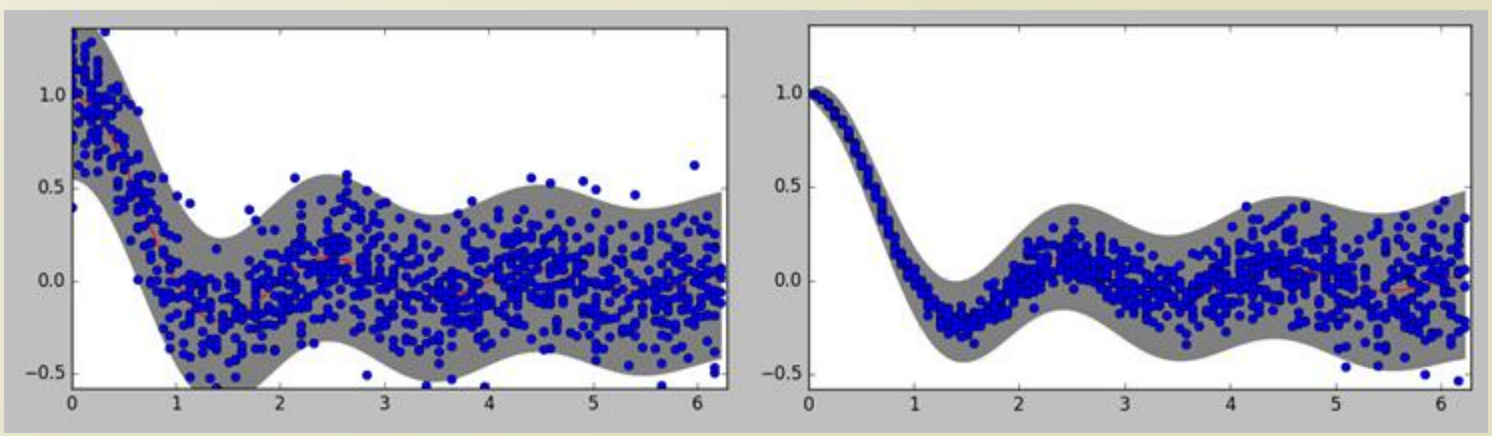




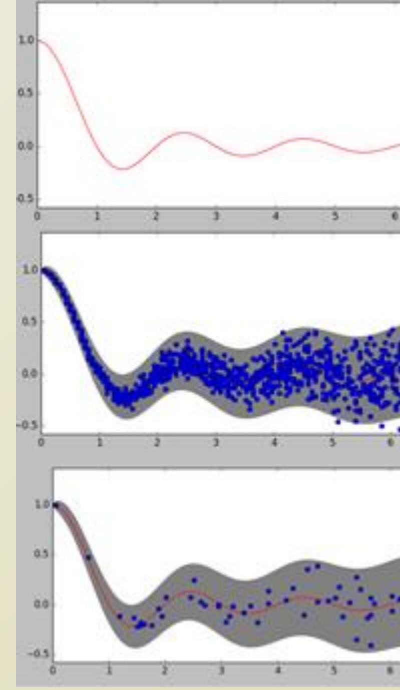
Underlying Function and Observations

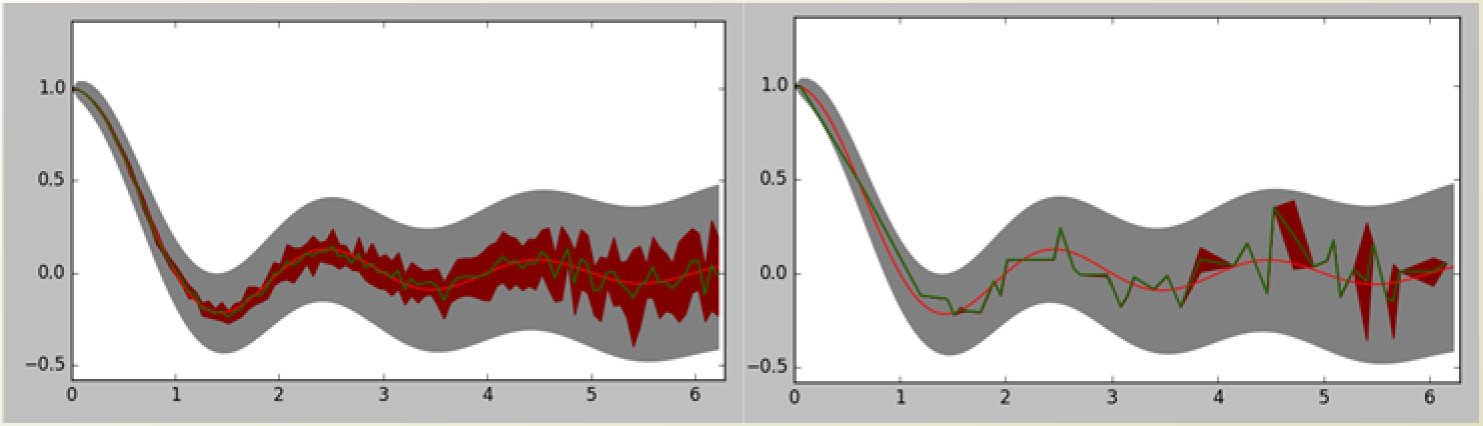


- Simple temporal line does not say much
  - Two cases of different observations from the same temporal line
- An observation dataset can be explained with two temporal functions
  - Function in two continuous domain
    - Under the assumption that the observation's noise is generated from a Gaussian 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 → 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

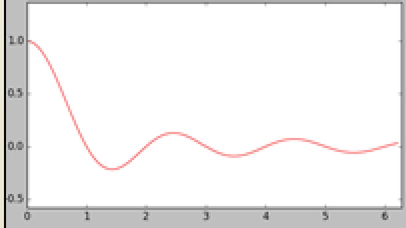
Simple Analyses with Domain Correlation

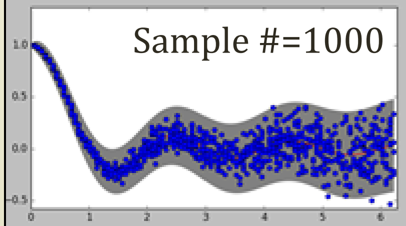
- Estimating the mean function with the domain correlation
  - Calculating the mean and the precision of Y with the correlated X
- Moving average with time-window $[w_{low}, w_{high}]$ and Dataset, D

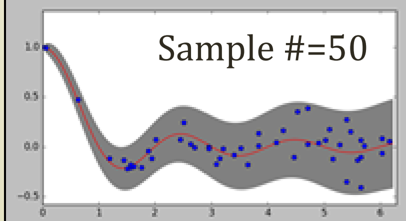
$$MA(x) = \frac{1}{N} \sum_{x_i \in W,D} y_i$$

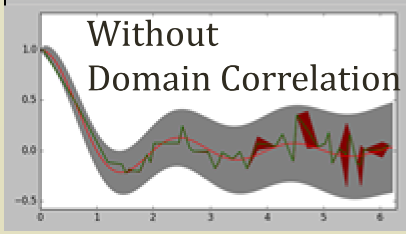
$$W = \left[x - w_{low}, x + w_{high}\right],\ N = |\{x_i | x_i \in W, D\}|$$

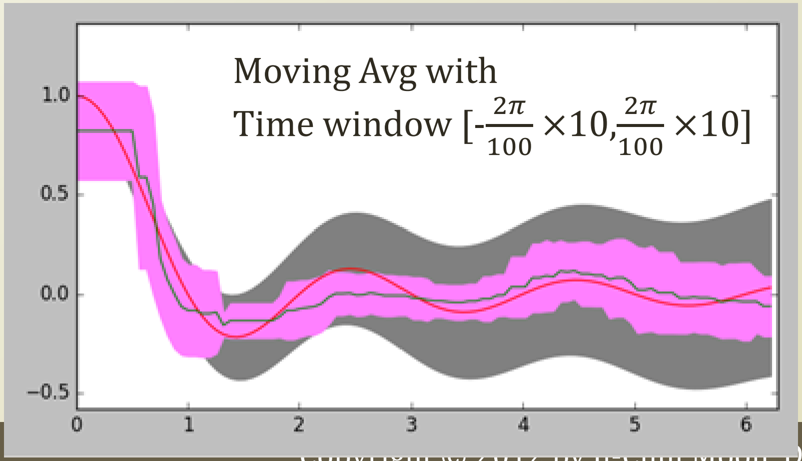
- Simple moving average because it does not differentiate yesterday and 10 days ago

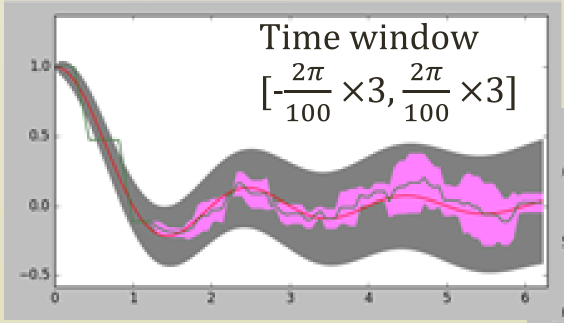


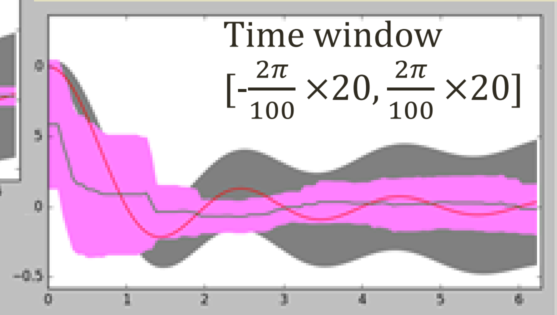




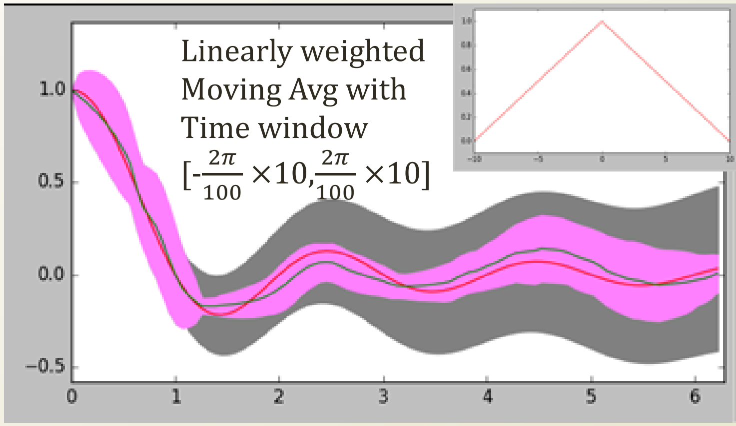


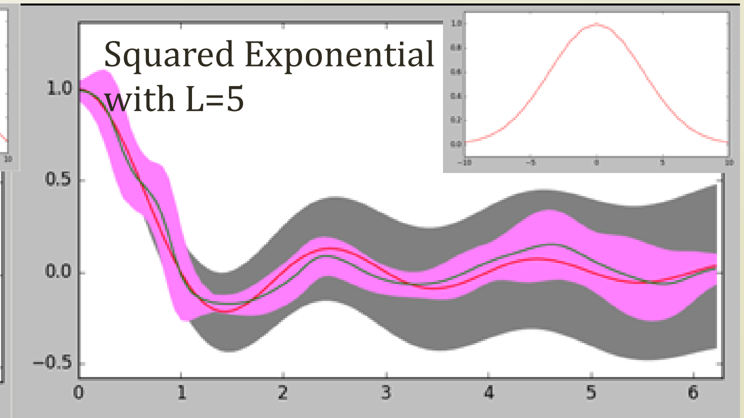


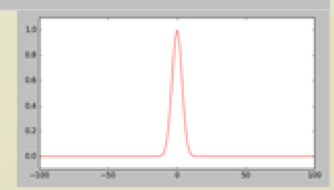


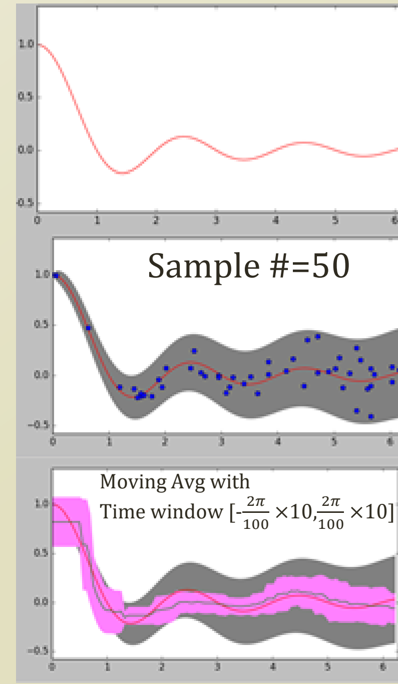


Simple Analyses with Differentiated Domain Correlation









- 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

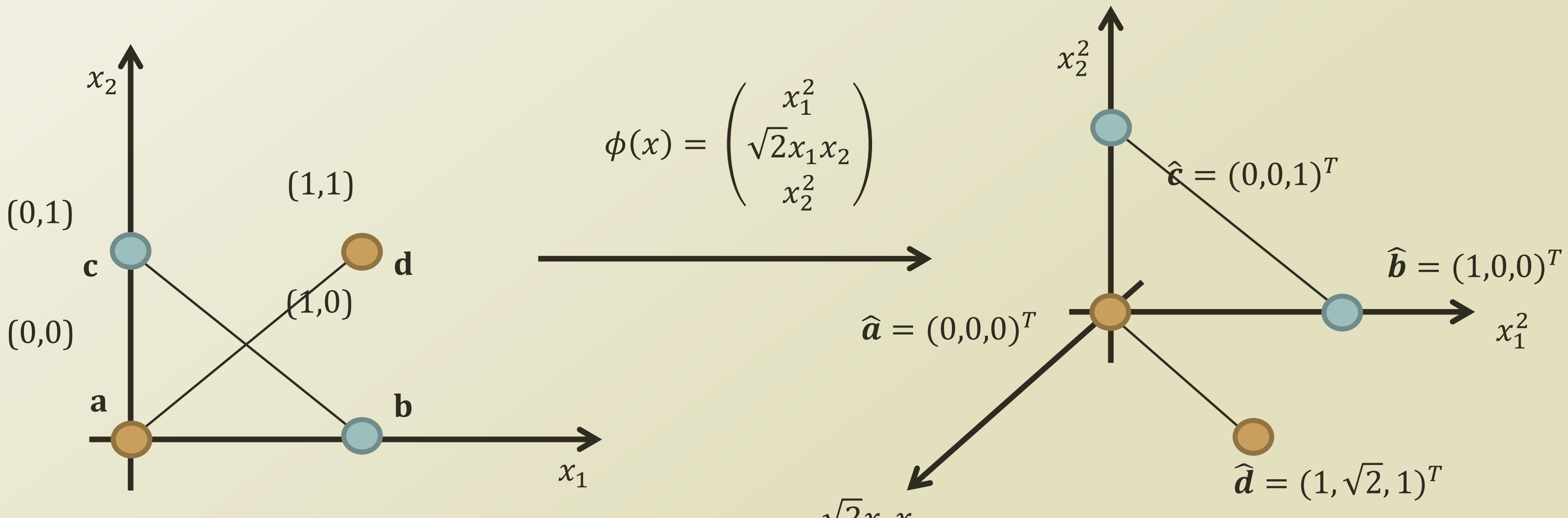
$$MA(x) = \frac{1}{\sum_{x_i \in W,D} k(x, x_i)} \sum_{x_i \in W,D} k(x, x_i) y_i \,, W = [x - w_{low}, x + w_{high}]$$

- 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: $\boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{e_3}, \ldots, \boldsymbol{e_n}$ → Linearly independent and generate $\boldsymbol{R^n}$
- Expanding the Basis through Space mapping function $\phi : L \rightarrow H$
  - Or, transformation function, etc…
- Any problem????
  - Feature space becomes bigger and bigger….



(a) Original space, L (b) Mapping space, H

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 → Y probability distribution
  - $E[Y] = E[\Phi w] = \Phi E[w] = 0$
  - $cov[Y] = E[(Y-0)(Y-0)^T) = E[YY^T]$

$$= E[\Phi w w^T \Phi^T] = \Phi E[ww^T]\Phi^T = \frac{1}{\alpha}\Phi\Phi^T$$

- $K_{nm} = k(x_n, x_m) = \frac{1}{\alpha}\phi(x_n)^T\phi(x_m)$
  - 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)
  - $K(\boldsymbol{x_i}, \boldsymbol{x_j}) = \varphi(\boldsymbol{x_i}) \cdot \varphi(\boldsymbol{x_j})$
- Some common kernels are following :
  - Polynomial(homogeneous)
    - $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\boldsymbol{x_i} \cdot \boldsymbol{x_j})^d$
  - Polynomial(inhomogeneous)
    - $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = (\boldsymbol{x_i} \cdot \boldsymbol{x_j} + 1)^d$
  - Gaussian kernel function, a.k.a. Radial Basis Function
    - $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \exp\left(-\gamma \left\|\boldsymbol{x_i} - \boldsymbol{x_j}\right\|^2\right)$
      - For $\gamma > 0$. Sometimes parameterized using $\gamma = \frac{1}{2\sigma^2}$
  - Hyperbolic tangent, a.k.a. Sigmoid Function
    - $k(\boldsymbol{x_i}, \boldsymbol{x_j}) = \tanh(\kappa \boldsymbol{x_i} \cdot \boldsymbol{x_j} + c)$
    - For some(not every) $\kappa > 0$ and $c < 0$

Detour: Polynomial Kernel Function

- Imagine we have
  - $\mathbf{x}$=<x_1, x_2> and $\mathbf{z}$=<z_1, z_2>
  - Polynomial Kernel Function of degree 1
    - $K(<x_1, x_2>, <z_1, z_2>) = <x_1, x_2> \cdot <z_1, z_2> = x_1 z_1 + x_2 z_2 = \mathbf{x} \cdot \mathbf{z}$
  - Polynomial Kernel Function of degree 2
    - $K(<x_1, x_2>, <z_1, z_2>) = <x_1^2, \sqrt{2} x_1 x_2, x_2^2> \cdot <z_1^2, \sqrt{2} z_1 z_2, z_2^2>$
    - $= x_1^2 z_1^2 + 2 x_1 x_2 z_1 z_2 + x_2^2 z_2^2 = (x_1 z_1 + x_2 z_2)^2 = (\mathbf{x} \cdot \mathbf{z})^2$
  - Polynomial Kernel Function of degree 3
    - $K(<x_1, x_2>, <z_1, z_2>) = (\mathbf{x} \cdot \mathbf{z})^3$
  - Polynomial Kernel Function of degree n
    - $K(<x_1, x_2>, <z_1, z_2>) = (\mathbf{x} \cdot \mathbf{z})^n$
- Do we need to express and calculate the transformed coordinate values for ***x*** and ***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

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

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

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

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

- Second order term of $lnP(Z)$
  - $-\frac{1}{2}Y^T K^{-1} Y - \frac{\beta}{2}T^T T + \frac{\beta}{2}TY + \frac{\beta}{2}YT - \frac{\beta}{2}Y^T Y$

$$= -\frac{1}{2}\begin{pmatrix} Y \\ T \end{pmatrix}^T \begin{pmatrix} K^{-1} + \beta I_N & -\beta I_N \\ -\beta I_N & \beta I_N \end{pmatrix} \begin{pmatrix} Y \\ T \end{pmatrix} = -\frac{1}{2}Z^T R Z$$

  - R becomes the precision matrix of Z
    - $M = (K^{-1} + \beta I_N - \beta I_N (\beta I_N)^{-1} \beta I_N)^{-1} = K$

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

  - $R^{-1} = \begin{pmatrix} K & K\beta I_N (\beta I_N)^{-1} \\ (\beta I_N)^{-1} \beta I_N K & (\beta I_N)^{-1} + (\beta I_N)^{-1} \beta I_N K \beta I_N (\beta I_N)^{-1} \end{pmatrix}$

$$= \begin{pmatrix} K & K \\ K & (\beta I_N)^{-1} + K \end{pmatrix}$$

- First order term of $lnP(Z)$ → None
- $P(Z) = N(Z|0, R^{-1})$

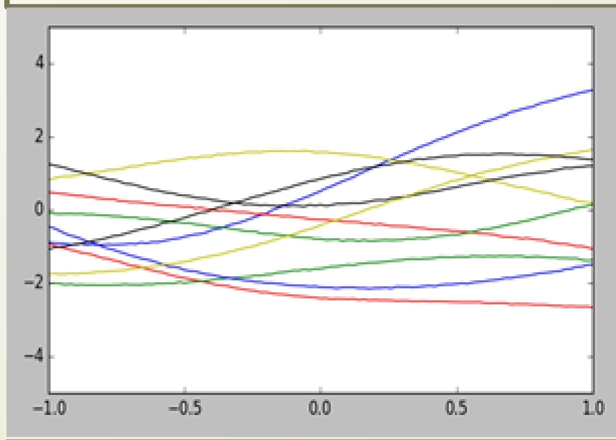
Marginal and Conditional Distribution of P(T)

- $P(T) = \int P(T|Y)P(Y)dY = \int N(T|Y, \beta^{-1}I_N)N(Y|0, K)dY$
  - $P(T|Y)P(Y) = P(Y, T) = P(Z)$
  - $P(Y, T) = N(Y, T|(0 \quad 0), \begin{pmatrix} K & K \\ K & (\beta I_N)^{-1} + K \end{pmatrix})$
    - 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 → $k(x_n, x_m) = \theta_0 \exp\left(-\frac{\theta_1}{2}||x_n - x_m||^2\right) + \theta_2 + \theta_3 x_n^T x_m$
- Our ultimate question as a regression problem is
  - $P(t_{N+1}|T_N)$=? → $P(T_{N+1})$=!

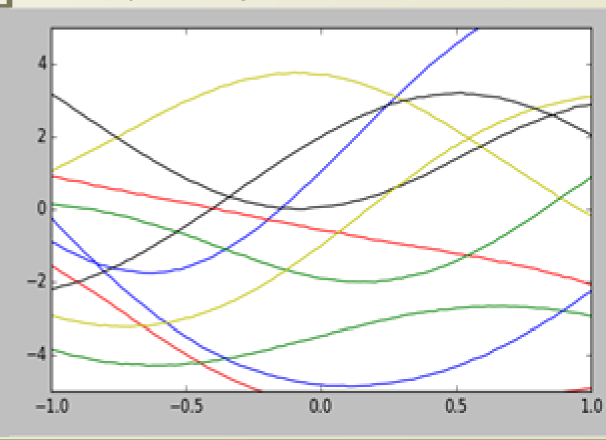
Sampling of P(T)

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

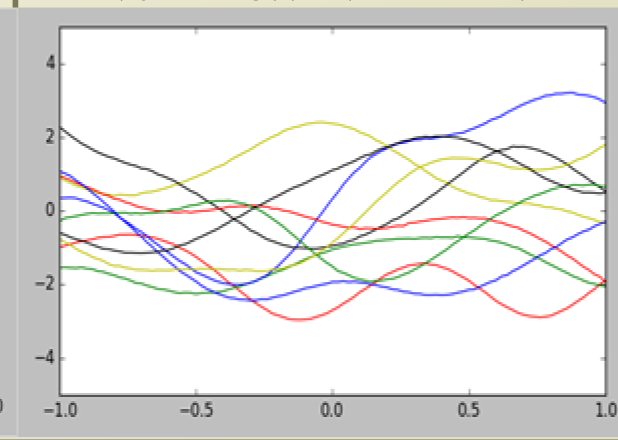
$(\theta_0, \theta_1, \theta_2, \theta_3, \beta) = (1,1,1,1,0.0001)$



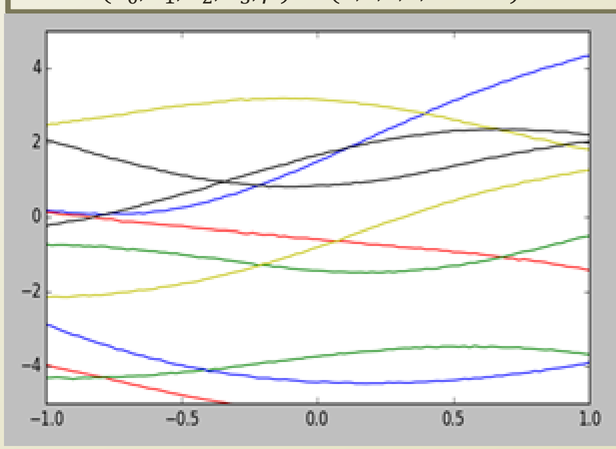
$(\theta_0, \theta_1, \theta_2, \theta_3, \beta) = (9,1,1,1,0.0001)$



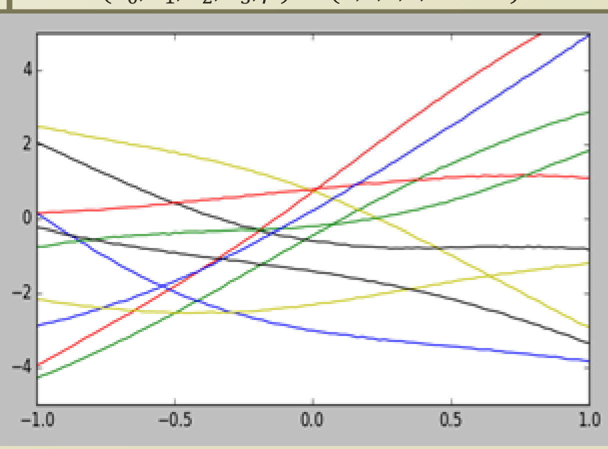
$(\theta_0, \theta_1, \theta_2, \theta_3, \beta) = (1,9,1,1,0.0001)$



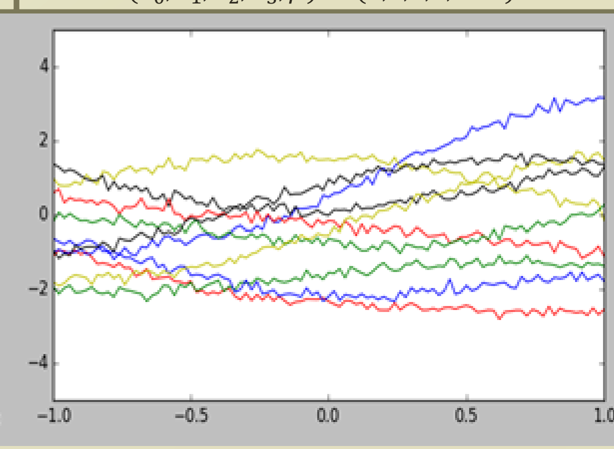
$(\theta_0, \theta_1, \theta_2, \theta_3, \beta) = (1,1,9,1,0.0001)$



$(\theta_0, \theta_1, \theta_2, \theta_3, \beta) = (1,1,1,9,0.0001)$



$(\theta_0, \theta_1, \theta_2, \theta_3, \beta) = (1,1,1,1,0.01)$



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

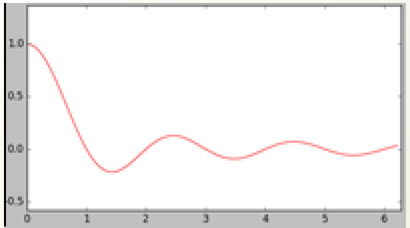
- $P(T) = N(T|0, (\beta I_N)^{-1} + K)$
  - $K_{nm} = k(x_n, x_m)$
- $P(T_{N+1}) = N(T|0, cov)$

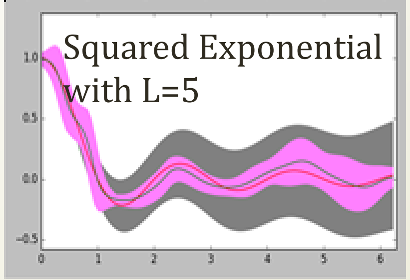
$$cov = \left[\begin{pmatrix} K_{11}+\beta & K_{12} & \cdots & K_{1N} & K_{1(N+1)} \\ K_{21} & K_{22}+\beta & & K_{2N} & K_{2(N+1)} \\ & \vdots & \ddots & & \vdots \\ K_{N1} & K_{N2} & & K_{NN}+\beta & K_{N(N+1)} \\ K_{(N+1)1} & K_{(N+1)2} & \cdots & K_{(N+1)N} & K_{(N+1)(N+1)}+\beta \end{pmatrix}\right]$$

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

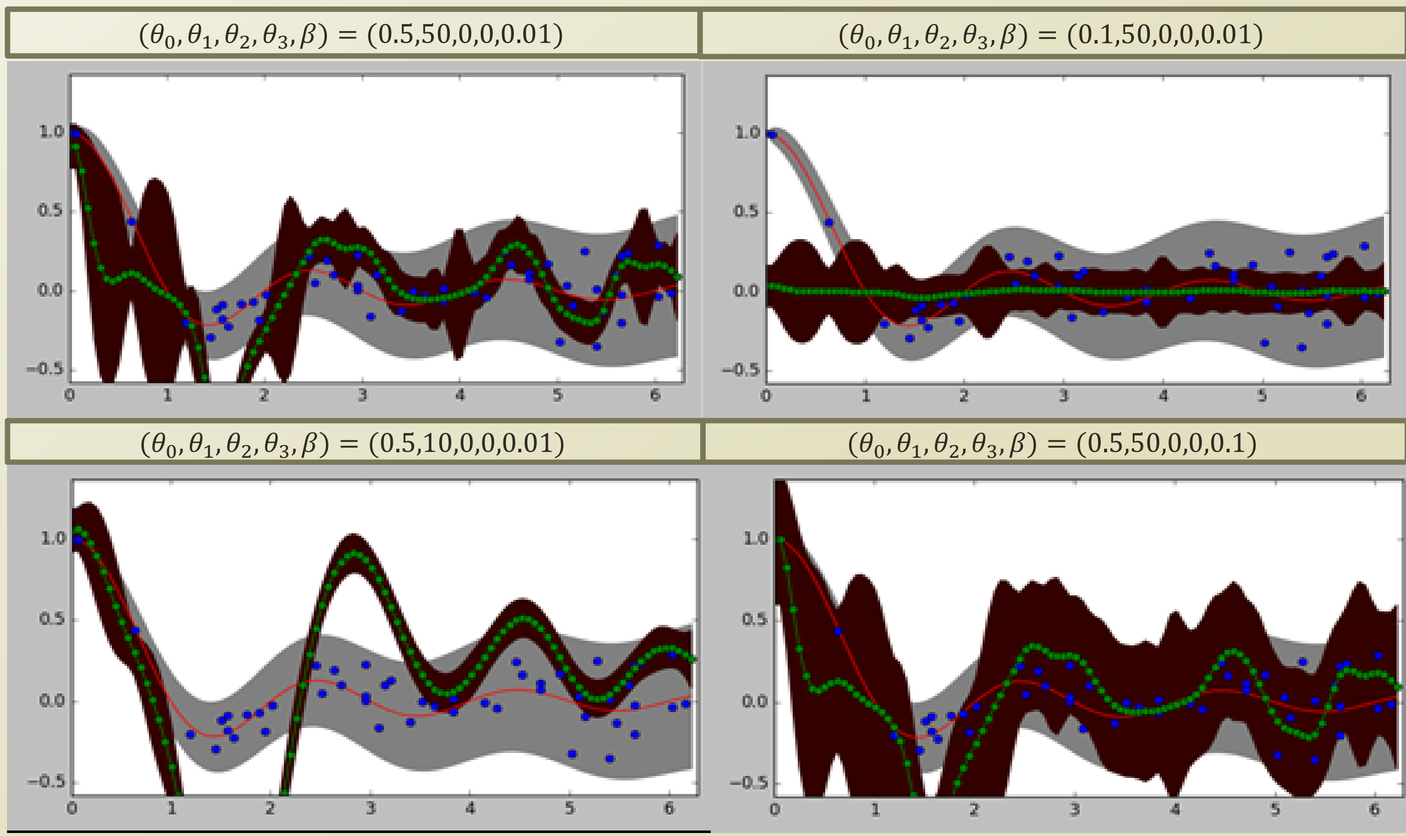
- Future distribution given the past data
  - Remember the theorem introduced earlier
    - $P(X_1|X_2) = N(X_1|\mu_1 + \Sigma_{12}\Sigma_{22}{}^{-1}(X_2 - \mu_2), \Sigma_{11} - \Sigma_{12}\Sigma_{22}{}^{-1}\Sigma_{21})$
  - $P(t_{N+1}|T_N) = \mathrm{N}(t_{N+1}|0 + k^T cov_N{}^{-1}(T_N - 0), \mathrm{c} - k^T cov_N{}^{-1}k)$
  - $\mu_{t_{N+1}} = k^T cov_N{}^{-1} T_N, \sigma^2{}_{t_{N+1}} = \mathrm{c} - k^T cov_N{}^{-1}k$

Gaussian Process Regression





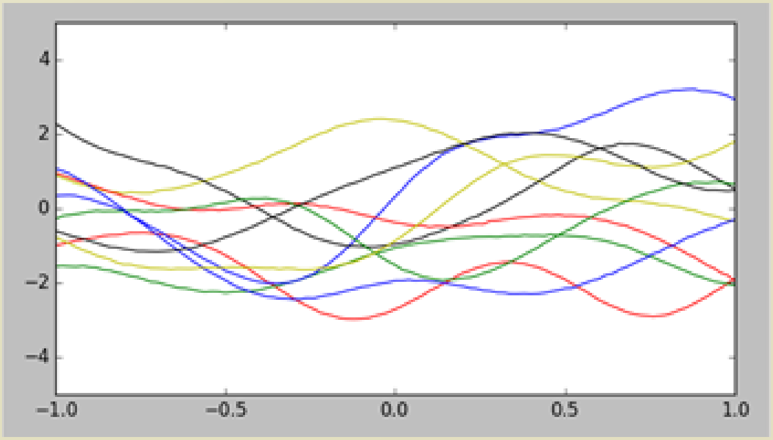
- $P(t_{N+1}|T_N) = \mathrm{N}(t_{N+1}|k^T cov_N{}^{-1}T_N, \mathrm{c} - k^T cov_N{}^{-1}k)$
- Gaussian process regression
  - Models the predictive distribution given the past records, $P(t_{N+1}|T_N)$
  - Mean of the predictive distribution could be the most likely point estimation of the prediction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



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 → $X(t,\omega)$ is a random variable over Ω
    - Fixed ω → $X(t,\omega)$ is a deterministic function of t , a sample function
- Example of random process
  - Gaussian process
  - $P(T) = N(T|0, (\beta I_N)^{-1} + K)$
    - $K_{nm} = k(x_n, x_m)$

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

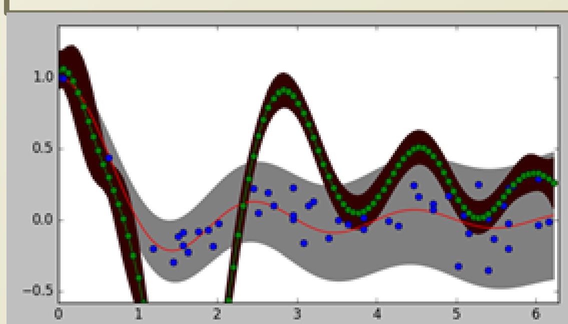


    - Fixed t, a random variable following a Gaussian distribution
    - Fixed ω, a deterministic curve of t

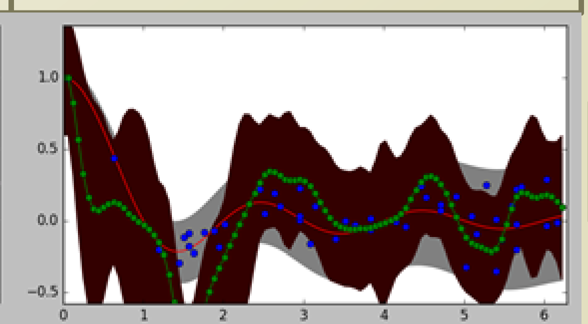
Hyper-parameters of Gaussian Process Regression

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

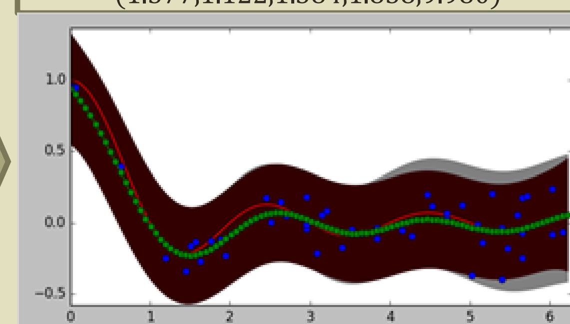
$(\theta_0, \theta_1, \theta_2, \theta_3, \beta) = (0.5, 10, 0, 0, 0.01)$



$(\theta_0, \theta_1, \theta_2, \theta_3, \beta) = (0.5, 50, 0, 0, 0.1)$



$(\theta_0, \theta_1, \theta_2, \theta_3, \beta) = (1.377, 1.122, 1.384, 1.858, 9.980)$



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(iteration, learningRate, trainingX, trainingY):
    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(10.0)

    # Kernel building
    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.pack(matCovarianceLinear)
    matCovariance = tf.reshape(matCovarianceCombined, [numDataPoints, numDataPoints])
    matCovarianceInv = tf.inv(matCovariance)
```

```
def KernelFunctionWithTensorFlow(theta0, theta1, theta2, theta3, X1, X2):
    insideexp1 = tf.mul(tf.div(theta1, 2.0), np.dot((X1 - X2), (X1 - X2)))
    insideexp2 = theta2
    insideexp3 = tf.mul(theta3, np.dot(np.transpose(X1), X2))
    insideexp = tf.add(tf.add(insideexp1, insideexp2), insideexp3)
    ret = tf.mul(theta0, tf.exp(insideexp))
    return ret
```

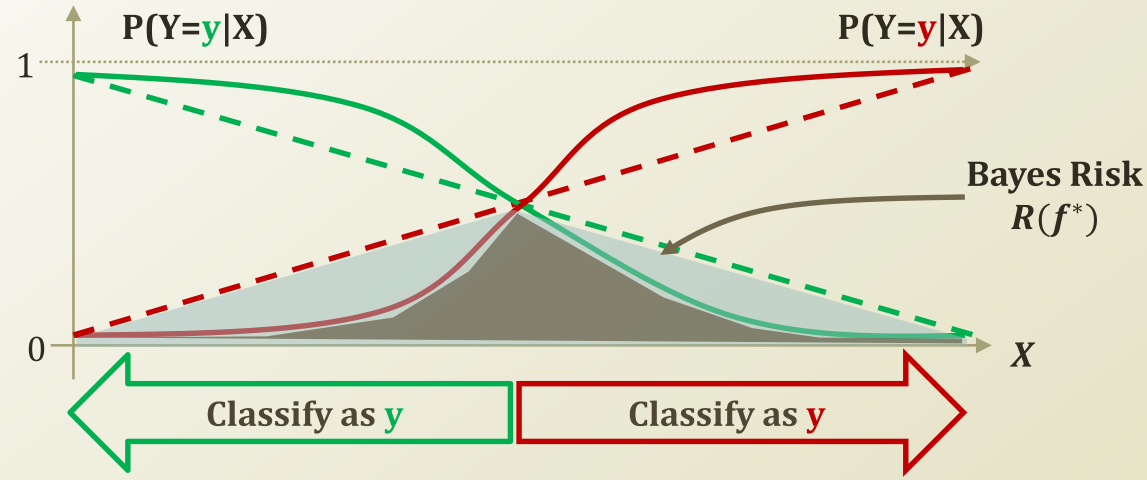
Probabilistic Programming for Hyperparameter Learning of GP (2)

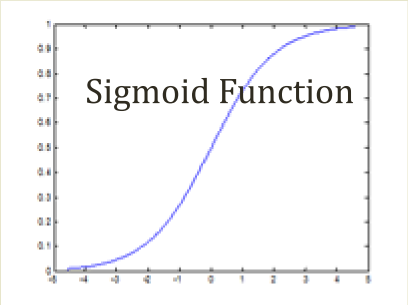
- $K_{nm} = k(x_n, x_m) = \theta_0 \exp\left(-\frac{\theta_1}{2} ||x_n - x_m||^2\right) + \theta_2 + \theta_3 x_n^T x_m$
- $P(T) = N(T|0, (\beta I_N)^{-1} + K) = N(T|0, C)$

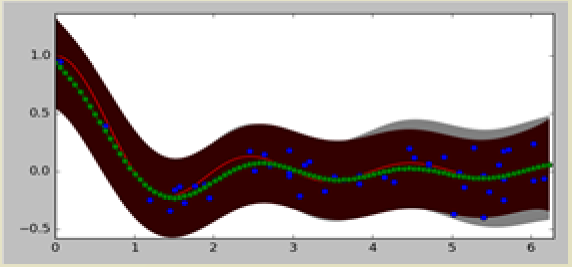
$$\mu_{t_{N+1}} = k^T {cov_N}^{-1} T_N, {\sigma^2}_{t_{N+1}} = \mathrm{c} - k^T {cov_N}^{-1} k$$

```python
# Prediction
negloglikelihood = 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.div(tf.add(tf.add(c, kernelEvaluationResult), 1), beta)

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

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

    negloglikelihood = tf.add(negloglikelihood, tf.div(tf.pow(tf.sub(predictionMu, tf.slice(obsY, [i, 0], [1, 1])), 2),tf.scalar_mul(tf.constant(2.0),predictionVar)))

# Training session declaration
training = tf.train.GradientDescentOptimizer(learningRate).minimize(negloglikelihood)
```

Gaussian Process Classifier







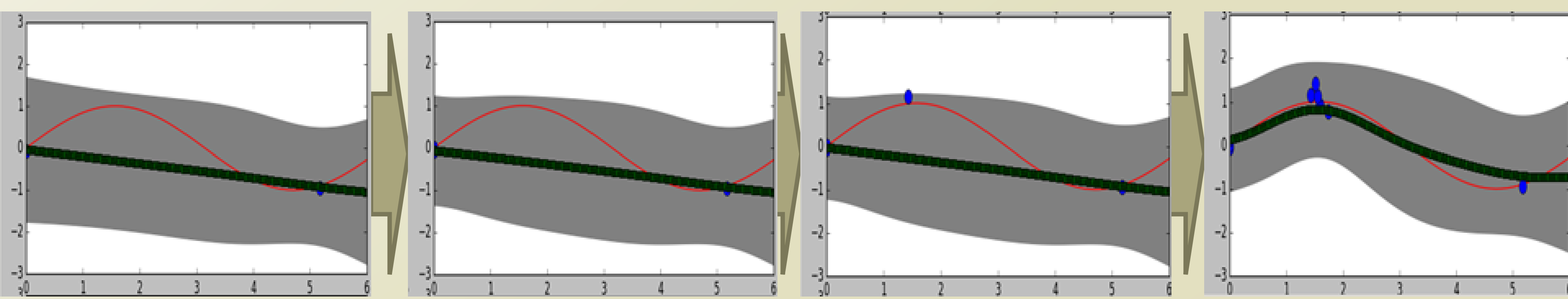
- Logistic regression
  - Sigmoid function(logistic function) + linear regression

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

- Gaussian process classifier
  - Sigmoid function(logistic function) + Gaussian process regression
  - Gaussian process : $f(x;\theta)$ → 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 → Next sampling
    - Predicted mean → potential optimized value
    - Predicted std. → 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
  - $MPI(x|D) = argmax_x P(y \geq (1+m)y_{max}|x, D), \quad y \sim N(\mu, \sigma^2)$

$$= argmax_x P\left(\frac{y-\mu}{\sigma} \geq \frac{(1+m)y_{max}-\mu}{\sigma}\right)$$

$$= argmax_x \left\{1 - \Phi\left(\frac{(1+m)y_{max}-\mu}{\sigma}\right)\right\}$$

$$= argmax_x \Phi\left(\frac{\mu-(1+m)y_{max}}{\sigma}\right)$$

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
  - $y = f(x), y_{max} = \max_{m=1,\dots,n} f(x_m)\,, u = \frac{y_{max}-\mu}{\sigma}, v = \frac{y-\mu}{\sigma}, \mu = f(x|\mathcal{D}), \sigma = K(x|\mathcal{D})$
  - $m = \max(0, y - y_{max}) = \max(0, (v-u)\sigma)$
  - $MEI(x|D) = argmax_x \int_0^\infty P(y \geq y_{max} + m)\, m\, dm$
  - $\int_0^\infty P(y \geq y_{max} + m)\, m\, dm = \int_0^\infty p\left(\frac{y-\mu}{\sigma} \geq \frac{y_{max}-\mu+m}{\sigma}\right) m\, dm = \int_0^\infty p\left(v \geq u + \frac{m}{\sigma}\right) m\, dm$
  - $= \int_u^\infty p(v)(v-u)\sigma\, dv = \sigma \int_u^\infty p(v)(v-u) dv = \sigma\left(\int_u^\infty vp(v)dv - \int_u^\infty up(v)dv\right)$
  - $= \sigma\left([-\phi(v)]_u^\infty - [u\Phi(v)]_u^\infty\right) = \sigma\left(\left(0-(-\phi(u))\right) - (u - u\Phi(u))\right) = \sigma(\phi(u) - u(1 -$

Bayesian Optimization Result

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

Sampling, Learned Function… | Prob. Of Improvement | Expected Improvement