# Predictive Distribution

*Abstract*—**This document contains theory involved in curve fitting.**

## 1 Objective

The objective is to implement the predictive distribution on a sinusoidal data set.

## 2 Generate Dataset

Create a sinusoidal function of the form

$$y = A\sin 2\pi x + n(t) \tag{2.0.1}$$

n(t) is the random noise that is included in the training set. This set consists of N samples of input data i.e. x expressed as shown below

$$x = \left(x_1, x_2, .., x_N\right)^T \tag{2.0.2}$$

which give the corresponding values of y denoted as

$$y = \left(y_1, y_2, .., y_N\right)^T \tag{2.0.3}$$

The corresponding values of y are generated from the Eq (4.0.3).The first term $A\sin 2\pi x$ is computed directly and then random noise samples having a normal(Gaussian) distribution are added inorder to get the corresponding values of y.

```
#Generate the sine curve
def g(X, noise_variance):
    '''Sinusoidal function plus noise'''
    return 0.5 + np.sin(2 * np.pi * X) + noise(X.
        shape, noise_variance)
```

The generated input matrix would look like

$$\mathbf{F} = \begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{N-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{N-1} \\ 1 & x_2 & x_2^2 & \dots & x_2^{N-1} \\ \vdots & & \vdots & & \vdots \\ 1 & \dots & \dots & \dots & x_N^{N-1} \end{pmatrix} \tag{2.0.4}$$

## 3 Polynomial Curve Fitting

The goal is to find the best line that fits into the pattern of the training data shown in the graph. We shall fit the data using a polynomial function of the form,

$$y\left(w, x\right) = \sum_{j=0}^{M} w_j x^j \tag{3.0.1}$$

$$\tag{3.0.2}$$

M is the order of the polynomial The polynomial coefficient are collectively denoted by the vector **w**.The proposed vector **w** of the model referring to Eq (2.0.4) is given by

$$\hat{\mathbf{w}} = \left(\mathbf{F}^T\mathbf{F}\right)^{-1}\mathbf{F}^T y \tag{3.0.3}$$

## 4 Predictive Distribution

For making a prediction t at a new location x we use the posterior predictive distribution which is defined as

$$p(t|x, t, \alpha, \beta) = \int p(t|x, w, \beta)p(w|t, \alpha, \beta)dw \tag{4.0.1}$$

The posterior predictive distribution includes uncertainty about parameters w into predictions by weighting the conditional distribution $p(t|x, w, \beta)$ with the posterior probability of weights $p(w|t, \alpha, \beta)$ over the entire weight parameter space. By using the predictive distribution we're not only getting the expected value of t at a new location x but also the uncertainty for that prediction. In our special case, the posterior predictive distribution is a Gaussian distribution as input data set is sinusoidal.

$$p(t|x, t, \alpha, \beta) = N(t|mTN\phi(x), \sigma 2N(x)) \tag{4.0.2}$$

where mean $mTN\phi(x)$ is the regression function after N observations and $\sigma 2N(x)$ is the corresponding predictive variance.

$$\sigma 2N(x) = 1\beta + \phi(x)TSN\phi(x) \tag{4.0.3}$$

The first term in 4.0.3 represents the inherent noise in the data and the second term covers the uncertainty about parameters w.

## 5 Implementation

Function *posterior* computes the mean and co variance matrix of the posterior distribution and function *posterior_predictive* computes the mean and the variances of the posterior predictive distribution.

```
import numpy as np
def posterior(Phi, t, alpha, beta, return_inverse=
    False):
    """Computes mean and covariance matrix of
        the posterior distribution."""
    S_N_inv = alpha * np.eye(Phi.shape[1]) +
        beta * Phi.T.dot(Phi)
    S_N = np.linalg.inv(S_N_inv)
    m_N = beta * S_N.dot(Phi.T).dot(t)
    if return_inverse:
        return m_N, S_N, S_N_inv
    else:
        return m_N, S_N
def posterior_predictive(Phi_test, m_N, S_N,
    beta):
    """Computes mean and variances of the
        posterior predictive distribution."""
    y = Phi_test.dot(m_N)
    # Only compute variances (diagonal elements
        of covariance matrix)
    y_var = 1 / beta + np.sum(Phi_test.dot(S_N
        ) * Phi_test, axis=1)
    return y, y_var
```

For fitting a linear model to a sinusoidal dataset we transform input x with *gaussian_basis_function* and later with *polynomial_basis_function*. These non-linear basis functions are necessary to model the non-linear relationship between input x and target t.

```
def gaussian_basis_function(x, mu, sigma=0.1):
    return np.exp(-0.5 * (x - mu) ** 2 / sigma
        ** 2)

def polynomial_basis_function(x, power):
    return x ** power
```

The following code shows how to fit a Gaussian basis function model to a noisy sinusoidal dataset.

```
N_list = [3, 8, 20]
beta = 25.0
alpha = 2.0
# Training observations in [-1, 1)
X = np.random.rand(N_list[-1], 1)
# Training target values
t = g(X, noise_variance=1/beta)
# Test observations
X_test = np.linspace(0, 1, 100).reshape(-1, 1)
# Function values without noise
y_true = g(X_test, noise_variance=0)

# Design matrix of test observations
Phi_test = expand(X_test, bf=
    gaussian_basis_function, bf_args=np.
    linspace(0, 1, 9))
```

This is below we are implementing the posterior predictive distribution function on our sinusoidal data set and the figure shows the predictive distribution.

```
for i, N in enumerate(N_list):
    X_N = X[:N]
    t_N = t[:N]

    # Design matrix of training observations
    Phi_N = expand(X_N, bf=
        gaussian_basis_function, bf_args=np.
        linspace(0, 1, 9))

    # Mean and covariance matrix of posterior
    m_N, S_N = posterior(Phi_N, t_N, alpha,
        beta)

    # Mean and variances of posterior predictive
    y, y_var = posterior_predictive(Phi_test,
        m_N, S_N, beta)

    # Draw 5 random weight samples from
        posterior and compute y values
    w_samples = np.random.multivariate_normal
        (m_N.ravel(), S_N, 5).T
    y_samples = Phi_test.dot(w_samples)

    plt.subplot(len(N_list), 2, i * 2 + 1)
    plot_data(X_N, t_N)
    plot_truth(X_test, y_true)
    plot_posterior_samples(X_test, y_samples)
    plt.ylim(-1.0, 2.0)
    plt.legend()

    plt.subplot(len(N_list), 2, i * 2 + 2)
    plot_data(X_N, t_N)
    plot_truth(X_test, y_true, label=None)
```

```
plot_predictive(X_test, y, np.sqrt(y_var))
plt.ylim(−1.0, 2.0)
plt.legend()
```

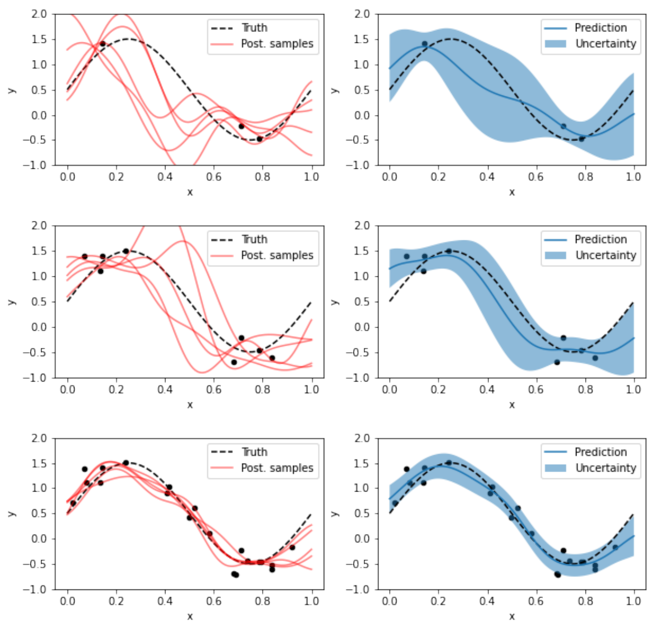


Fig. 0

Python code:

```
https://github.com/sahilsin/EE_IDP/blob/main/
    Assignment_4/pd.ipynb
```