# ENM 531: Data-driven modeling and probabilistic scientific computing

## *Lecture #5: Bayesian linear regression*

Paris Perdikaris
January 31, 2019

Penn
University of Pennsylvania

$$f : \mathcal{X} \to \mathcal{Y}$$

# Example #1: Atmospheric science

| | $\delta_K$ | | | |
|---|---|---|---|---|
| Latitude | $K = 0.67$ | $K = 1.5$ | $K = 2.0$ | $K = 3.0$ |
| 65 | -3.1 | 3.52 | 6.05 | 9.3 |
| 55 | -3.22 | 3.62 | 6.02 | 9.3 |
| 45 | -3.3 | 3.65 | 5.92 | 9.17 |
| 35 | -3.32 | 3.52 | 5.7 | 8.82 |
| 25 | -3.17 | 3.47 | 5.3 | 8.1 |
| 15 | -3.07 | 3.25 | 5.02 | 7.52 |
| 5 | -3.02 | 3.15 | 4.95 | 7.3 |
| -5 | -3.02 | 3.15 | 4.97 | 7.35 |
| -15 | -3.12 | 3.2 | 5.07 | 7.62 |
| -25 | -3.2 | 3.27 | 5.35 | 8.22 |
| -35 | -3.35 | 3.52 | 5.62 | 8.8 |
| -45 | -3.37 | 3.7 | 5.95 | 9.25 |
| -55 | -3.25 | 3.7 | 6.1 | 9.5 |

**Table 3.1.** Variation of the average yearly temperature on the Earth for four different values of the concentration $K$ of carbon acid at different latitudes

# Example #2: Finance

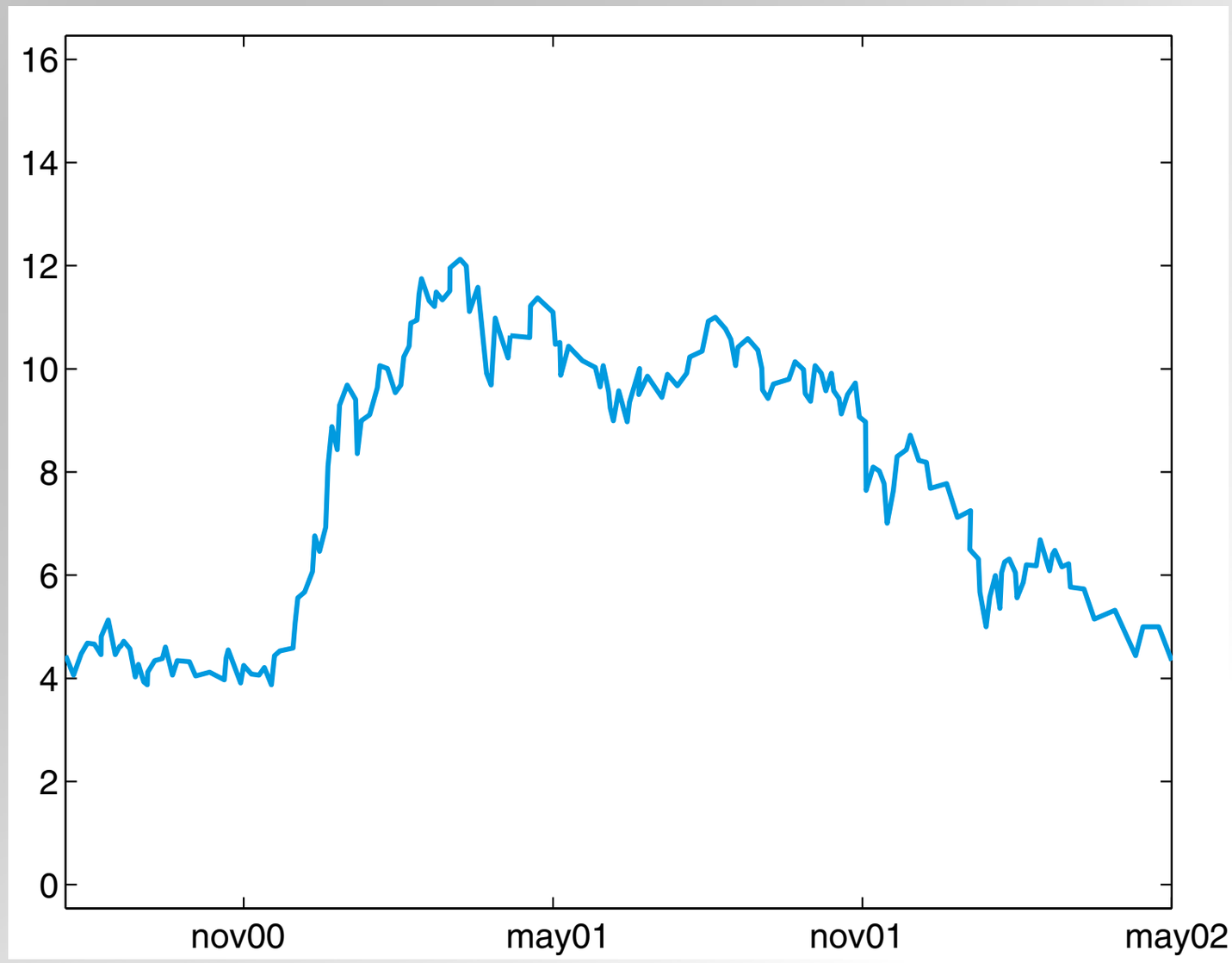
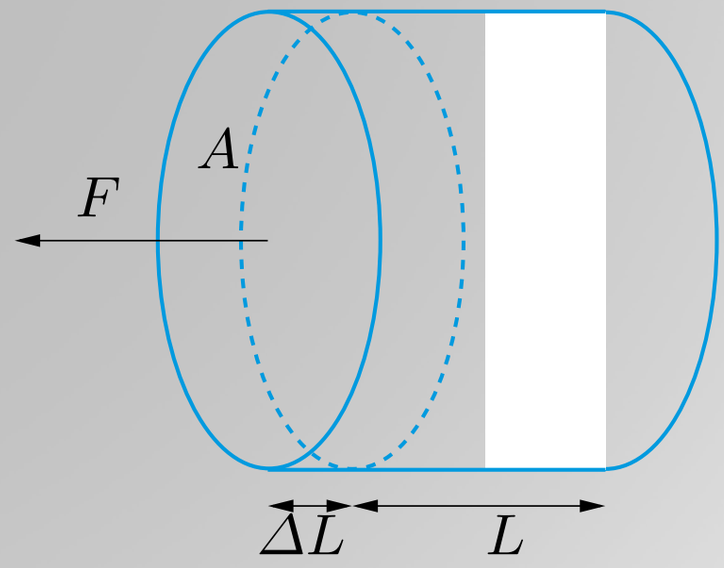



**Fig. 3.1.** Price variation of a stock over two years

# Example #3: Biomechanics




$$\sigma = F/A$$
$$\epsilon = \Delta L/L$$

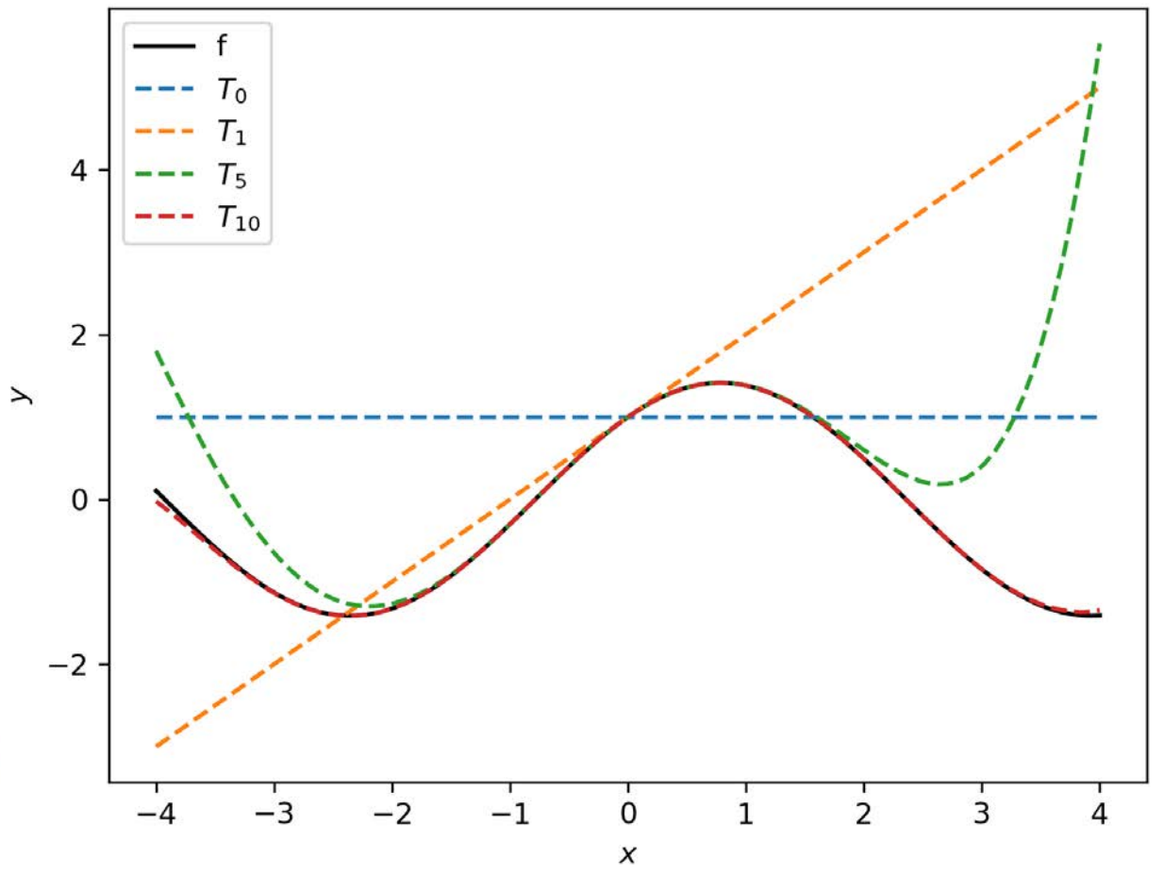
**Fig. 3.2.** A schematic representation of an intervertebral disc

| test | stress $\sigma$ | stress $\epsilon$ | test | stress $\sigma$ | stress $\epsilon$ |
|---|---|---|---|---|---|
| 1 | 0.00 | 0.00 | 5 | 0.31 | 0.23 |
| 2 | 0.06 | 0.08 | 6 | 0.47 | 0.25 |
| 3 | 0.14 | 0.14 | 7 | 0.60 | 0.28 |
| 4 | 0.25 | 0.20 | 8 | 0.70 | 0.29 |

**Table 3.2.** Values of the deformation for different values of a stress applied on an intervertebral disc

# Local approximation with Taylor series

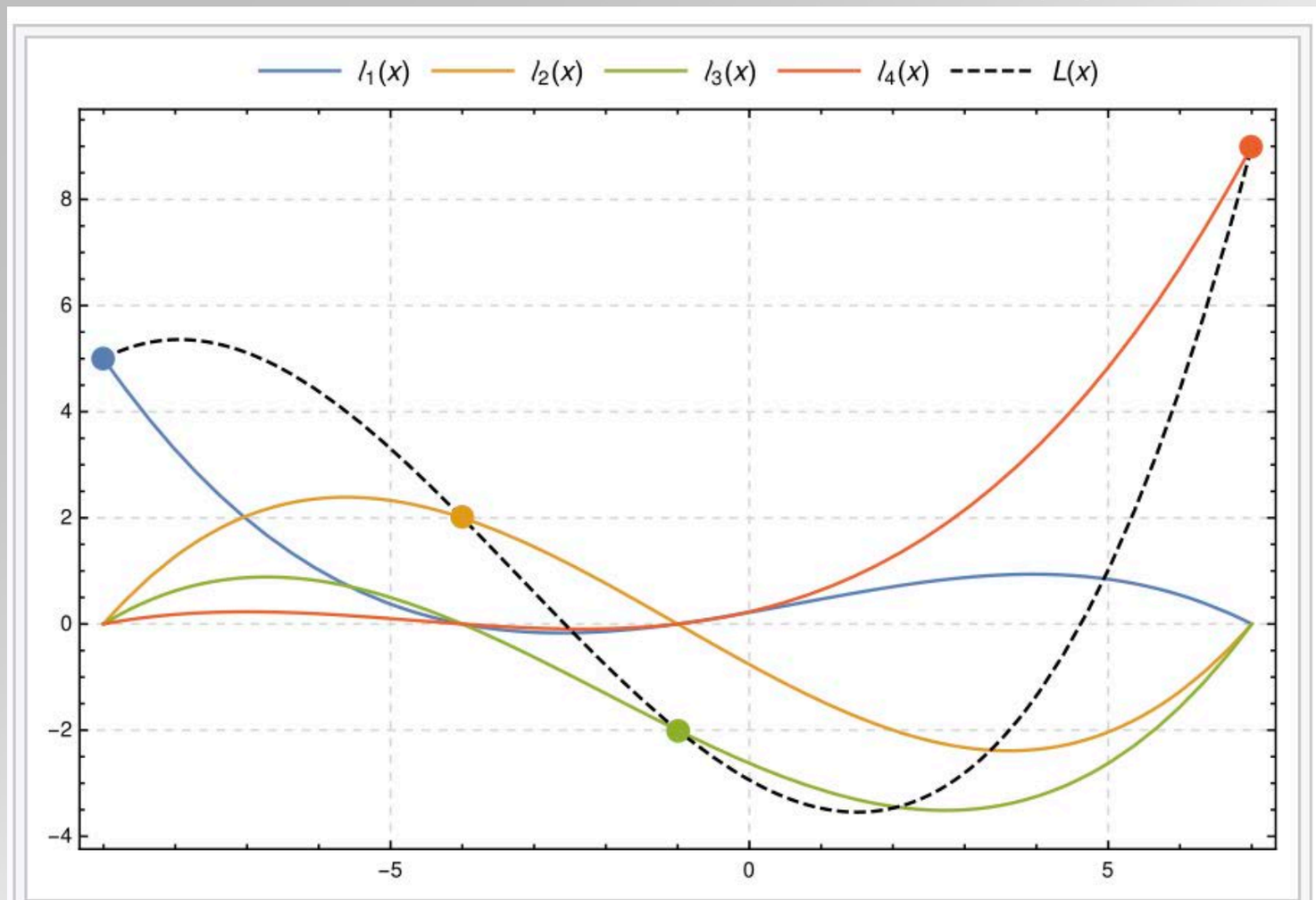
```python
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
Created on Tue Aug 28 12:27:37 2018

@author: paris
"""

import autograd.numpy as np
from autograd import grad
from scipy.special import factorial
import matplotlib.pyplot as plt

if __name__ == '__main__':

    def f(x):
        return np.sin(x) + np.cos(x)

    def TaylorSeries(f, x, x0, n = 2):
        T = f(x0)*np.ones_like(x)
        grad_f = grad(f)
        for i in range(0, n):
            T += grad_f(x0)*(x-x0)**(i+1) / factorial(i+1)
            grad_f = grad(grad_f)
        return T


    N = 100
    x = np.linspace(-4.0,4.0,N)
    y = f(x)

    x0 = 0.0

    n = [0, 1, 5, 10]
    plt.figure(1)
    plt.plot(x, y, 'k-', label = 'f')
    for i in range(0, len(n)):
        T = TaylorSeries(f, x, x0, n[i])
        plt.plot(x, T, '--', label = '$T_{%d}$' % (n[i]))
    plt.xlabel('$x$')
    plt.ylabel('$y$')
    plt.legend()
```




$$T_n(x) := \sum_{k=0}^{n} \frac{f^{(k)}(x_0)}{k!}(x - x_0)^k$$

# Interpolation with Lagrange polynomials

$$f(x) = \sum_{k=1}^{n} y_k \phi_k(x), \quad \phi_k(x) = \prod_{\substack{0 \le k \le n \\ k \neq j}} \frac{x - x_j}{x_k - x_j}$$

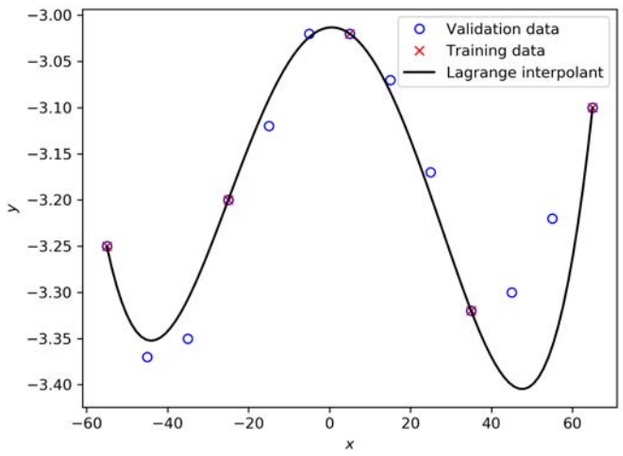



This image shows, for four points ((–9, 5), (–4, 2), (–1, –2), (7, 9)), the (cubic) interpolation polynomial $L(x)$ (dashed, black), which is the sum of the *scaled* basis polynomials $y_0\ell_0(x)$, $y_1\ell_1(x)$, $y_2\ell_2(x)$ and $y_3\ell_3(x)$. The interpolation polynomial passes through all four control points, and each *scaled* basis polynomial passes through its respective control point and is 0 where $x$ corresponds to the other three control points.

# Deterministic vs probabilistic modeling

$\boldsymbol{x}$

Inputs
(deterministic)


$\boldsymbol{y} = f_\theta(\boldsymbol{x})$

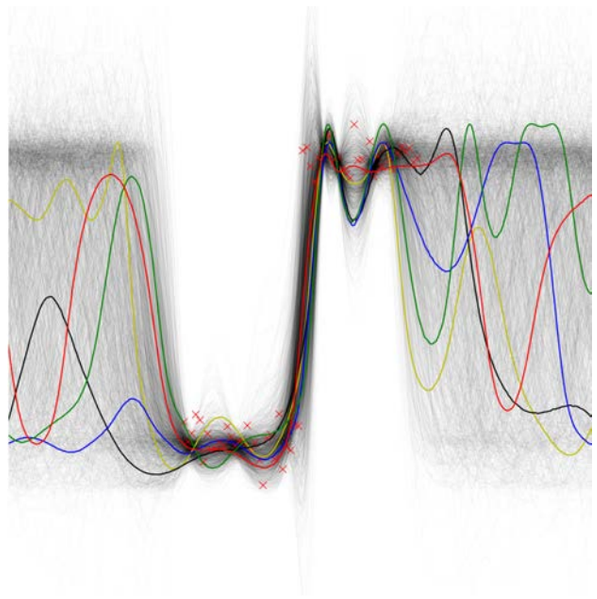
Outputs

Inputs
(random)

$\boldsymbol{z} \sim p(\boldsymbol{z})$

$\boldsymbol{x}$

Inputs
(deterministic)



$p_\theta(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$

Outputs

# Supervised learning

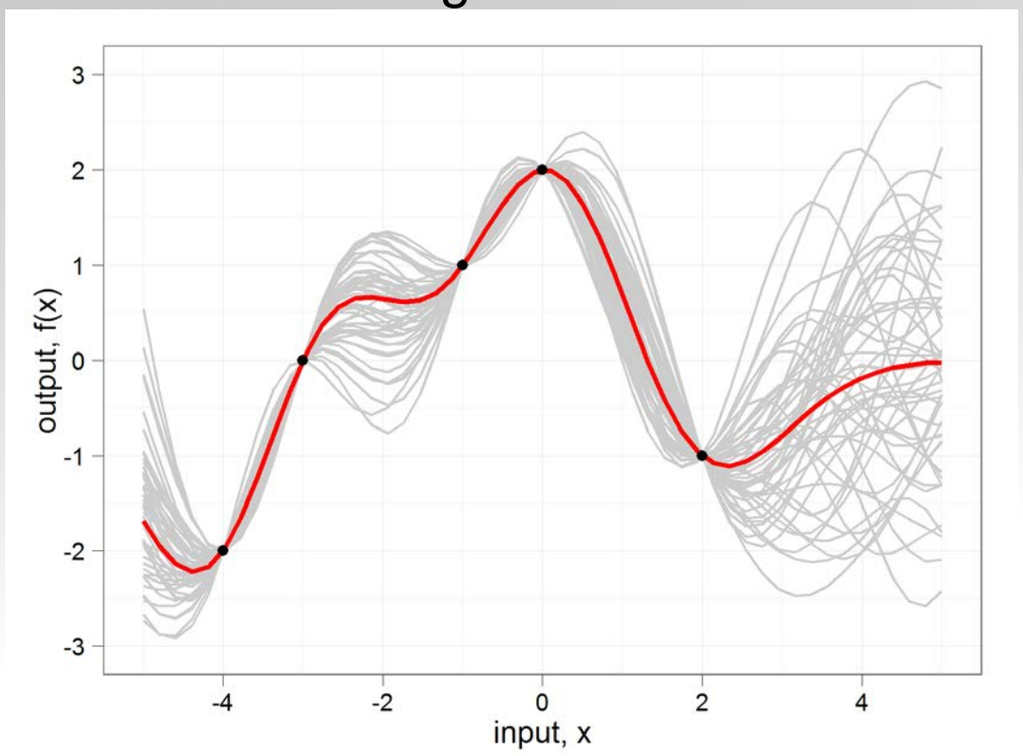
$$f : \mathcal{X} \to \mathcal{Y}$$

$$\mathcal{D} = \{\boldsymbol{x}, \boldsymbol{y}\},\ \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} \in \mathcal{Y}$$

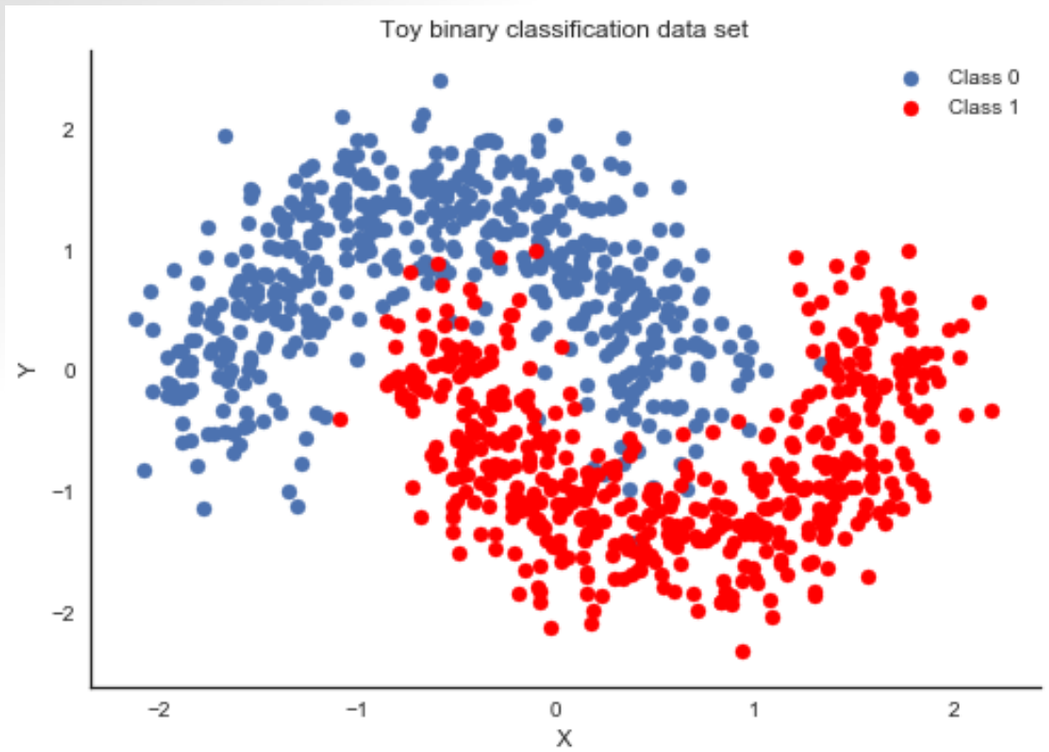
$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon$$

$$p(f(\boldsymbol{x}^*)|\boldsymbol{x}^*, \mathcal{D})$$

## Regression




## Classification



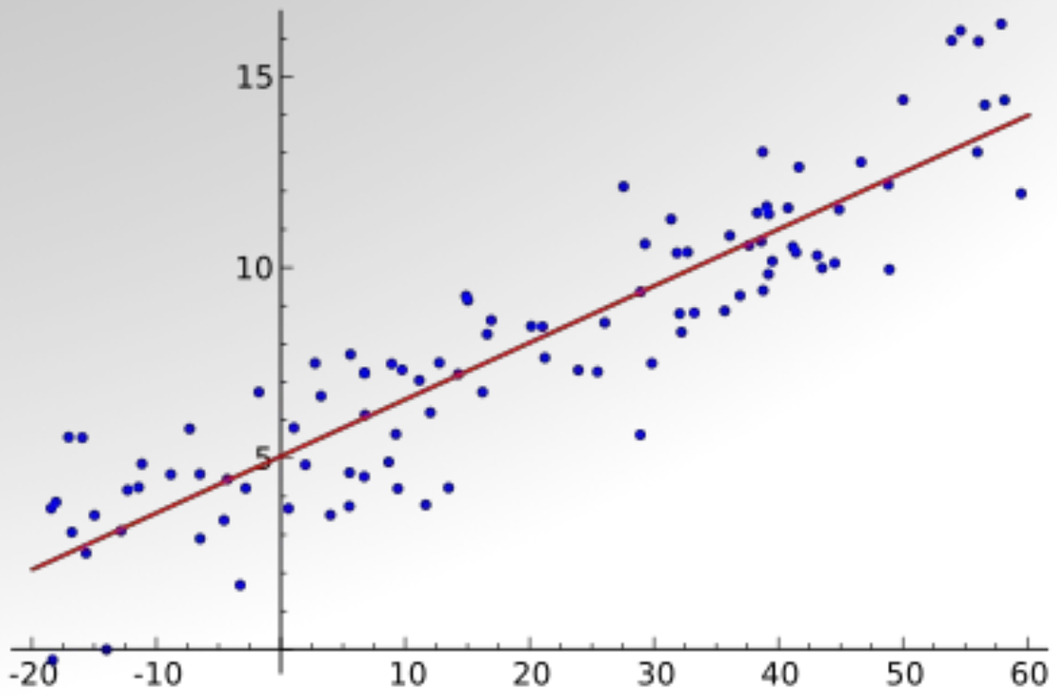
# Linear regression

$$f : \mathcal{X} \to \mathcal{Y}$$

$$\mathcal{D} = \{\boldsymbol{x}, \boldsymbol{y}\},\ \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} \in \mathcal{Y}$$

$$\boldsymbol{y} = f(\boldsymbol{x}) + \epsilon$$

$$f(\boldsymbol{x}) = w^T \boldsymbol{x}$$




*"It's <u>not</u> just about lines and planes!"*

# Linear regression

Nonlinear functions can be approximating using basis functions (or features)

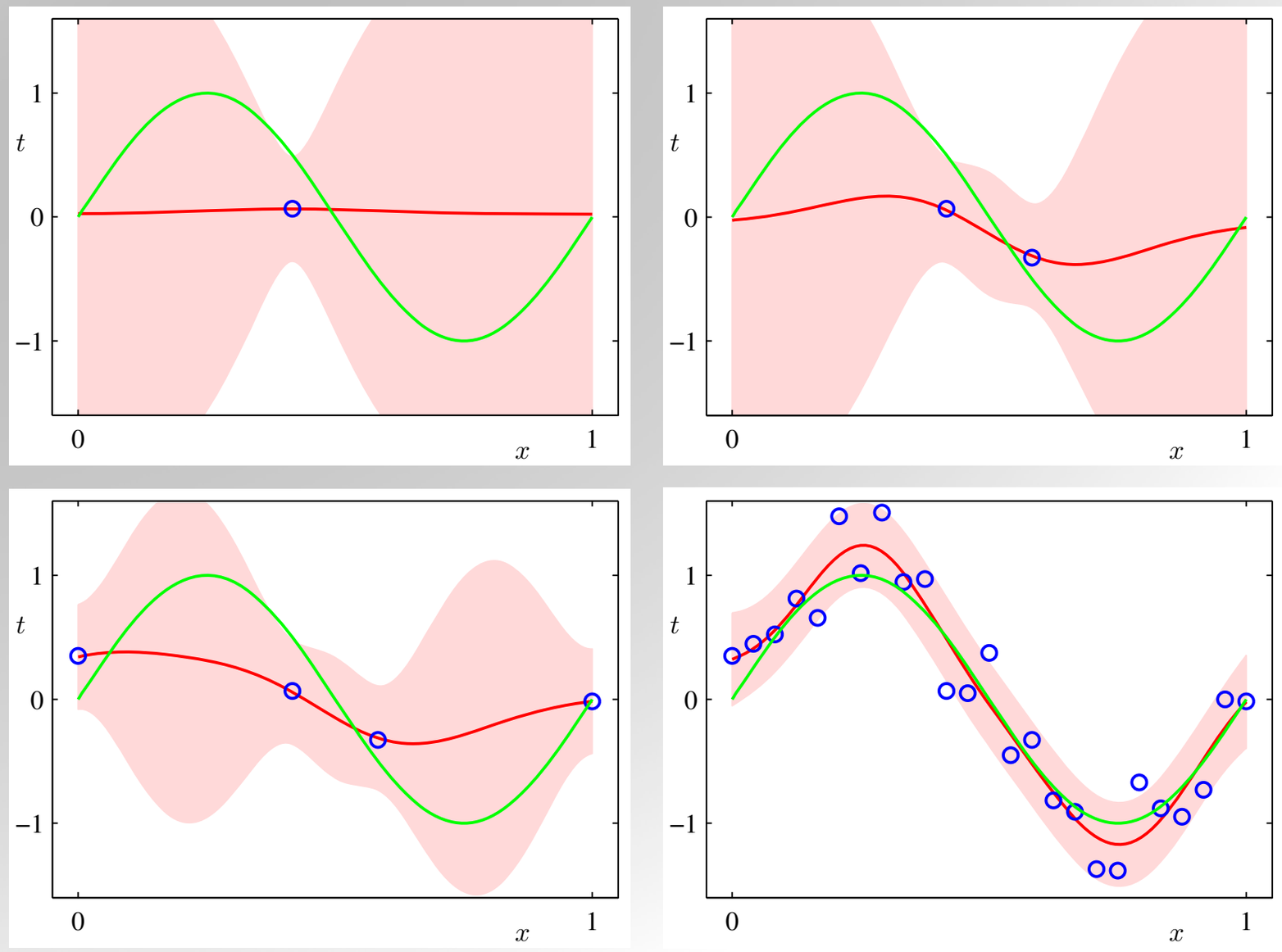



**Figure 3.8** Examples of the predictive distribution (3.58) for a model consisting of $9$ Gaussian basis functions of the form (3.4) using the synthetic sinusoidal data set of Section 1.1. See the text for a detailed discussion.

$$\boldsymbol{y} = w^T \phi(\boldsymbol{x}) + \epsilon$$

# Linear regression with basis functions

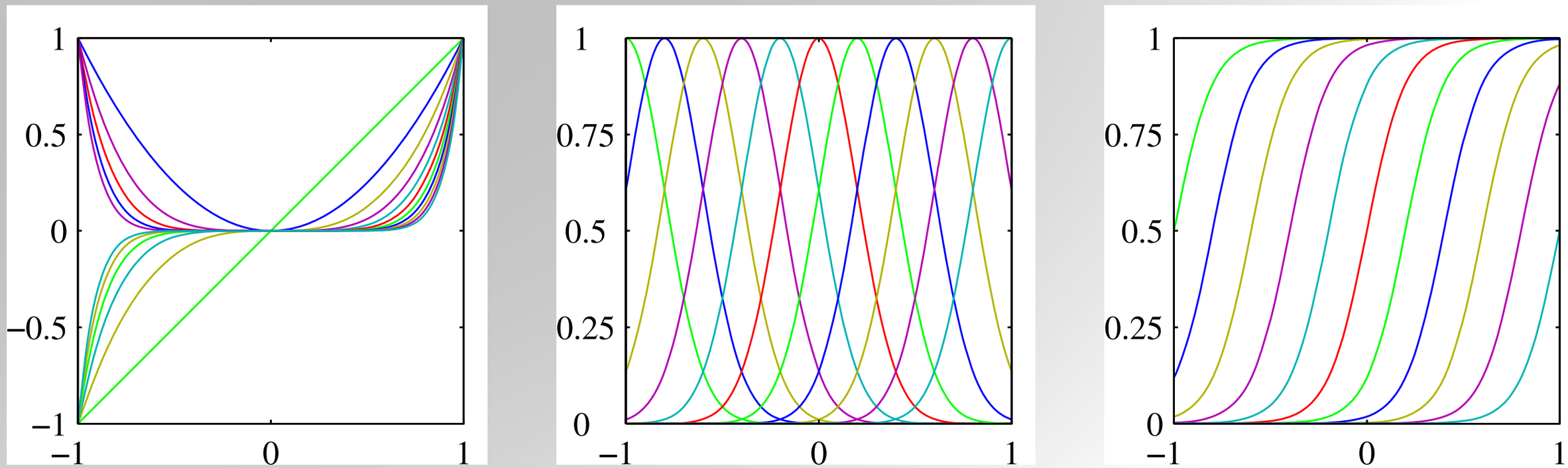
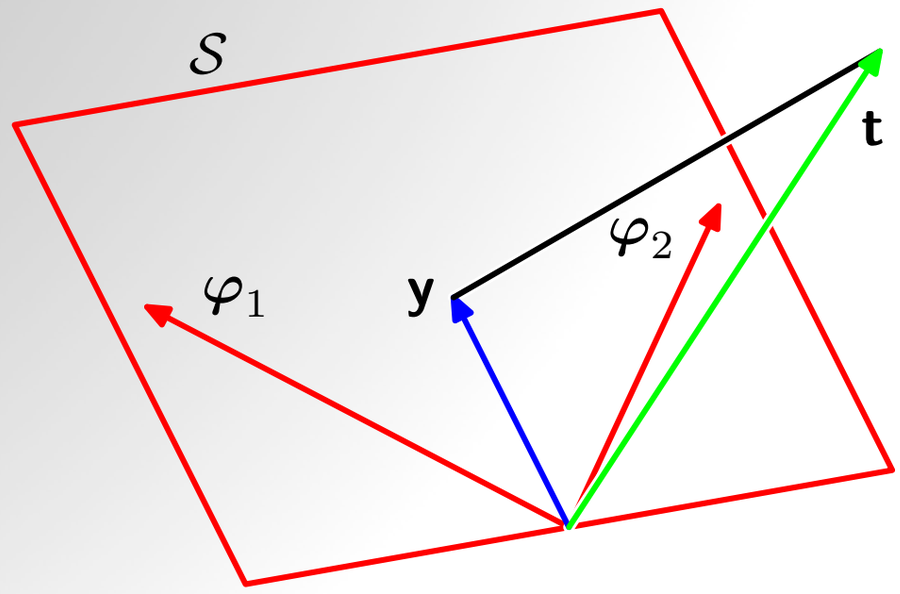


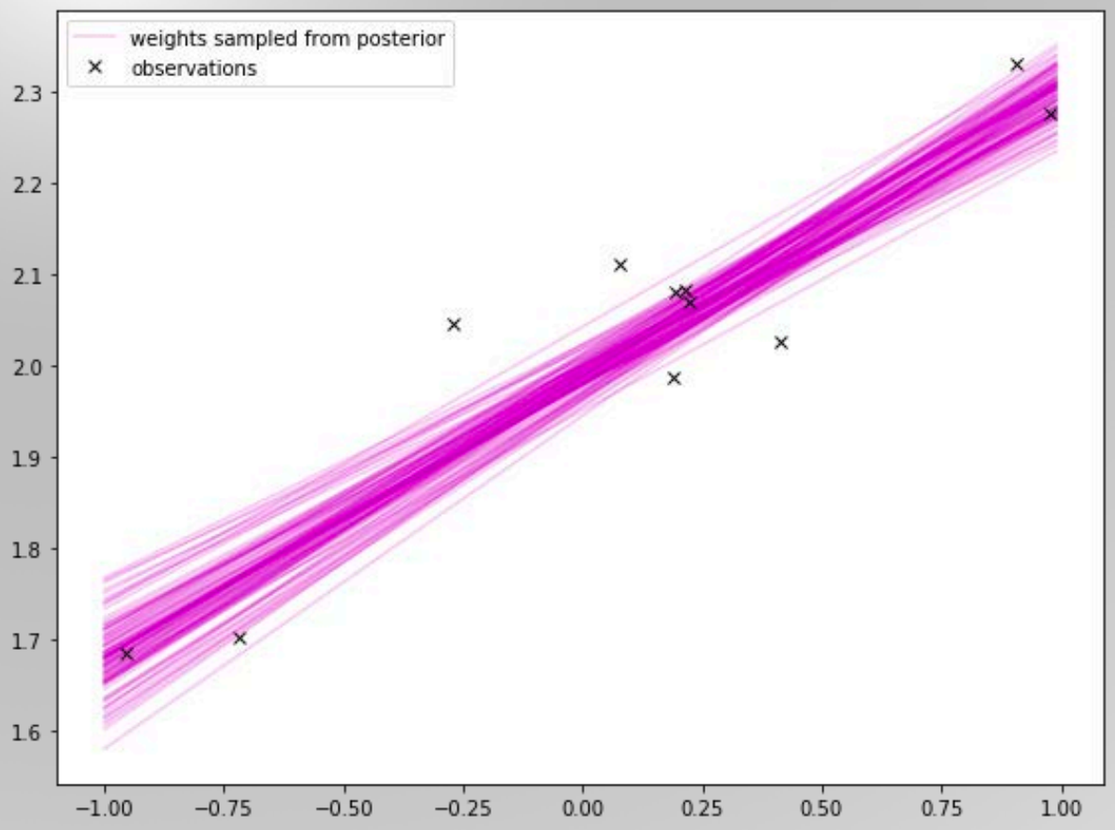

**Figure 3.1** Examples of basis functions, showing polynomials on the left, Gaussians of the form (3.4) in the centre, and sigmoidal of the form (3.5) on the right.

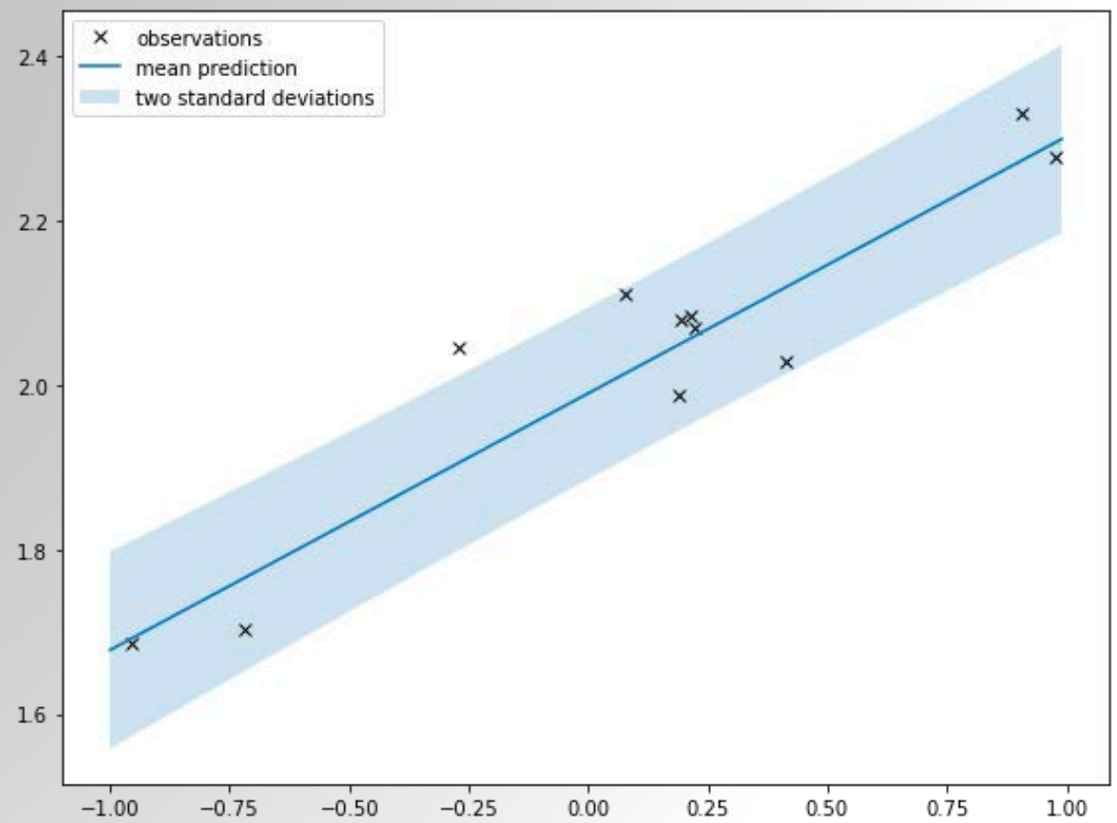
# Geometrical interpretation

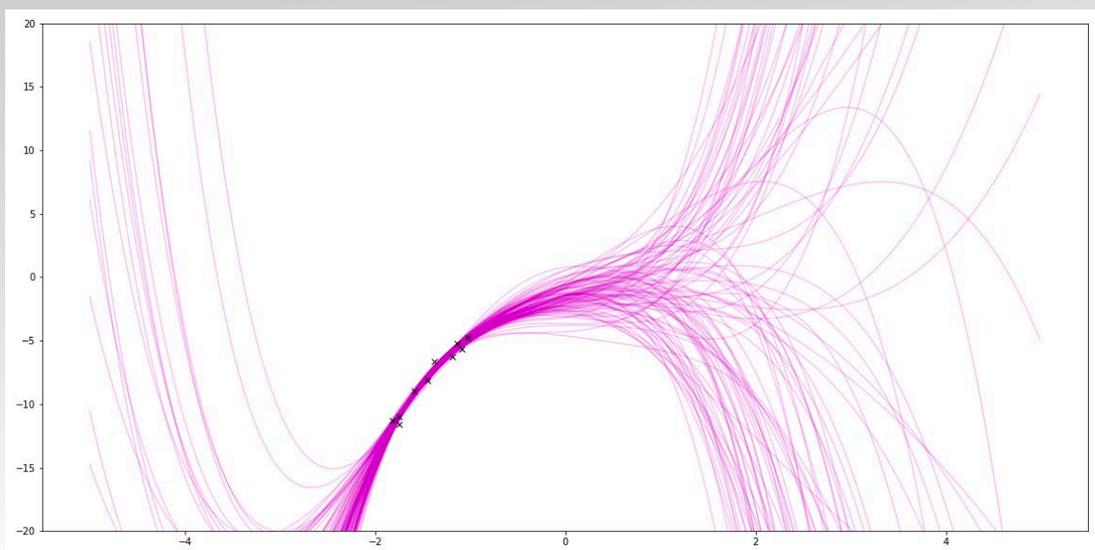
**Figure 3.2** Geometrical interpretation of the least-squares solution, in an $N$-dimensional space whose axes are the values of $t_1, \ldots, t_N$. The least-squares regression function is obtained by finding the orthogonal projection of the data vector **t** onto the subspace spanned by the basis functions $\phi_j(\mathbf{x})$ in which each basis function is viewed as a vector $\boldsymbol{\varphi}_j$ of length $N$ with elements $\phi_j(\mathbf{x}_n)$.

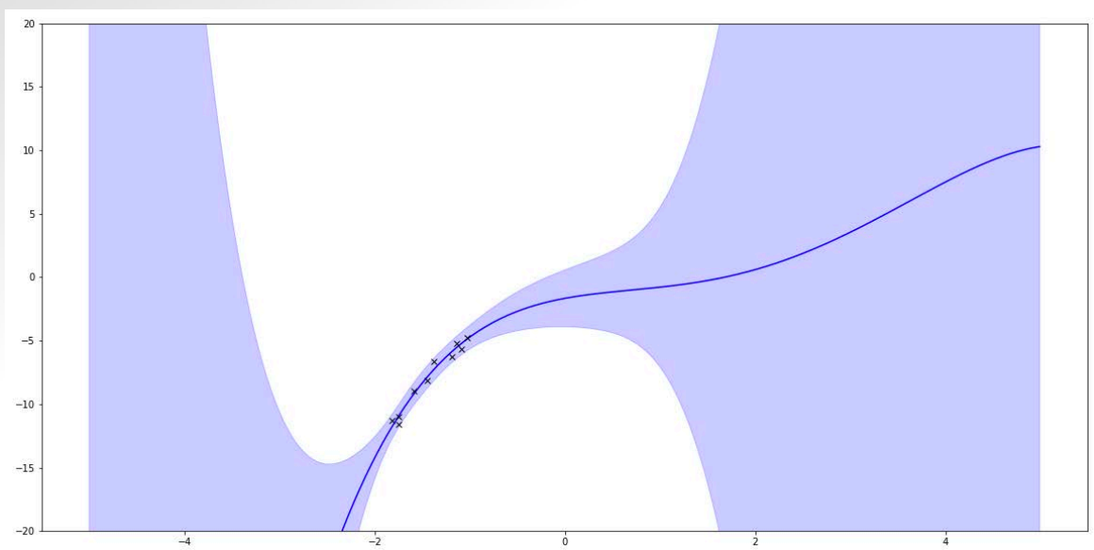


# Bayesian linear regression



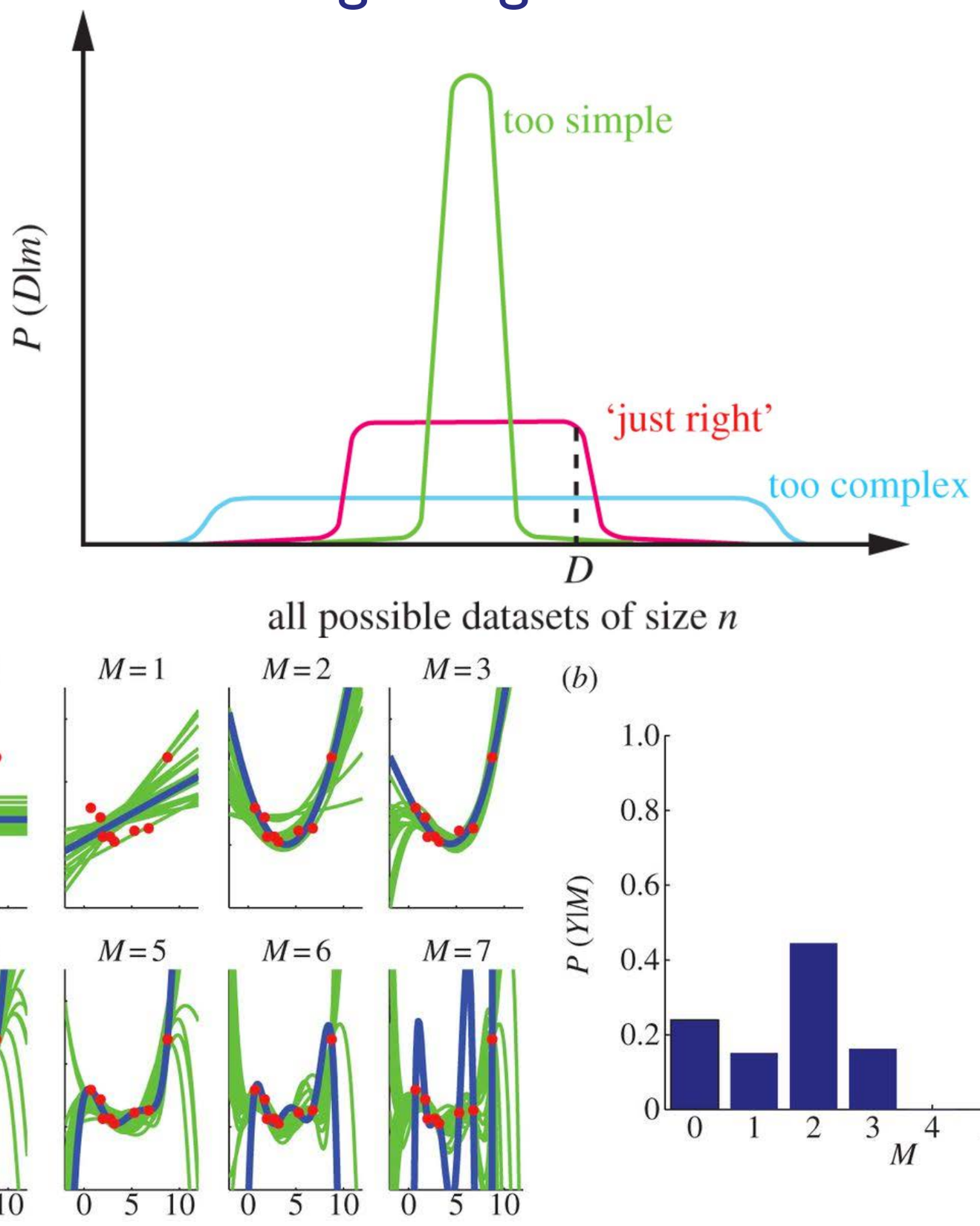



# Occam's razor - Overfitting - Regularization

William of Ockham (~1285-1347 A.D)

***"plurality should not be posited without necessity."***




*Ghahramani, Z. (2013). Bayesian non-parametrics and the probabilistic approach to modelling. Phil. Trans. R. Soc. A, 371(1984), 20110553.*

# Bayesian linear regression

```python
class BayesianLinearRegression:
    """
      Linear regression model: y = (w.T)*x + \epsilon
      w ~ N(0,beta^(-1)I)
      P(y|x,w) ~ N(y|(w.T)*x,alpha^(-1)I)
    """
    def __init__(self, X, y, alpha = 1.0, beta = 1.0):

        self.X = X
        self.y = y

        self.alpha = alpha
        self.beta = beta

        self.jitter = 1e-8


    def fit_MLE(self):
        xTx_inv = np.linalg.inv(np.matmul(self.X.T,self.X) + self.jitter)
        xTy = np.matmul(self.X.T, self.y)
        w_MLE = np.matmul(xTx_inv, xTy)

        self.w_MLE = w_MLE

        return w_MLE

    def fit_MAP(self):
        Lambda = np.matmul(self.X.T,self.X) + \
                 (self.beta/self.alpha)*np.eye(self.X.shape[1])
        Lambda_inv = np.linalg.inv(Lambda)
        xTy = np.matmul(self.X.T, self.y)
        mu = np.matmul(Lambda_inv, xTy)

        self.w_MAP = mu
        self.Lambda_inv = Lambda_inv

        return mu, Lambda_inv

    def predictive_distribution(self, X_star):
        mean_star = np.matmul(X_star, self.w_MAP)
        var_star = 1.0/self.alpha + \
                   np.matmul(X_star, np.matmul(self.Lambda_inv, X_star.T))
        return mean_star, var_star
```