Access Code: **413868**

CMP9794M
Advanced Artificial Intelligence

Heriberto Cuayahuitl

UNIVERSITY OF LINCOLN

School of Engineering and Physical Sciences

Last Week: Gaussian Bayes Nets (GBNs) via Linear Gaussians

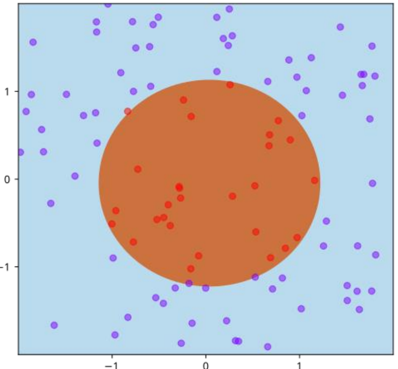
- Let Y be a continuous random variable with continuous parents $X_1, \dots, X_k$. A linear Gaussian can be defined as

$$P(Y|pa(Y)) = N(\underbrace{\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \beta_k X_k; \sigma^2})$$

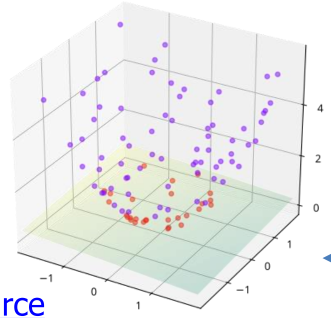
Beta coefficients estimated via linear (such as Ridge, Lasso) or non-linear regression (e.g., KernelRidge).

Note: The latter (KernelRigde) can be considered as non-linear in the original feature space and linear in the transformed feature space (via the Kernel trick).

Original feature space



Source



Transfored feature space

Univariate Gaussian Distribution

A Gaussian (a.k.a. normal distribution) is a **PDF** defined as

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Mean

Standard deviation

$e = 2.71828 \ldots$

$\pi = 3.14159 \ldots$

Equivalent to:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

Variance

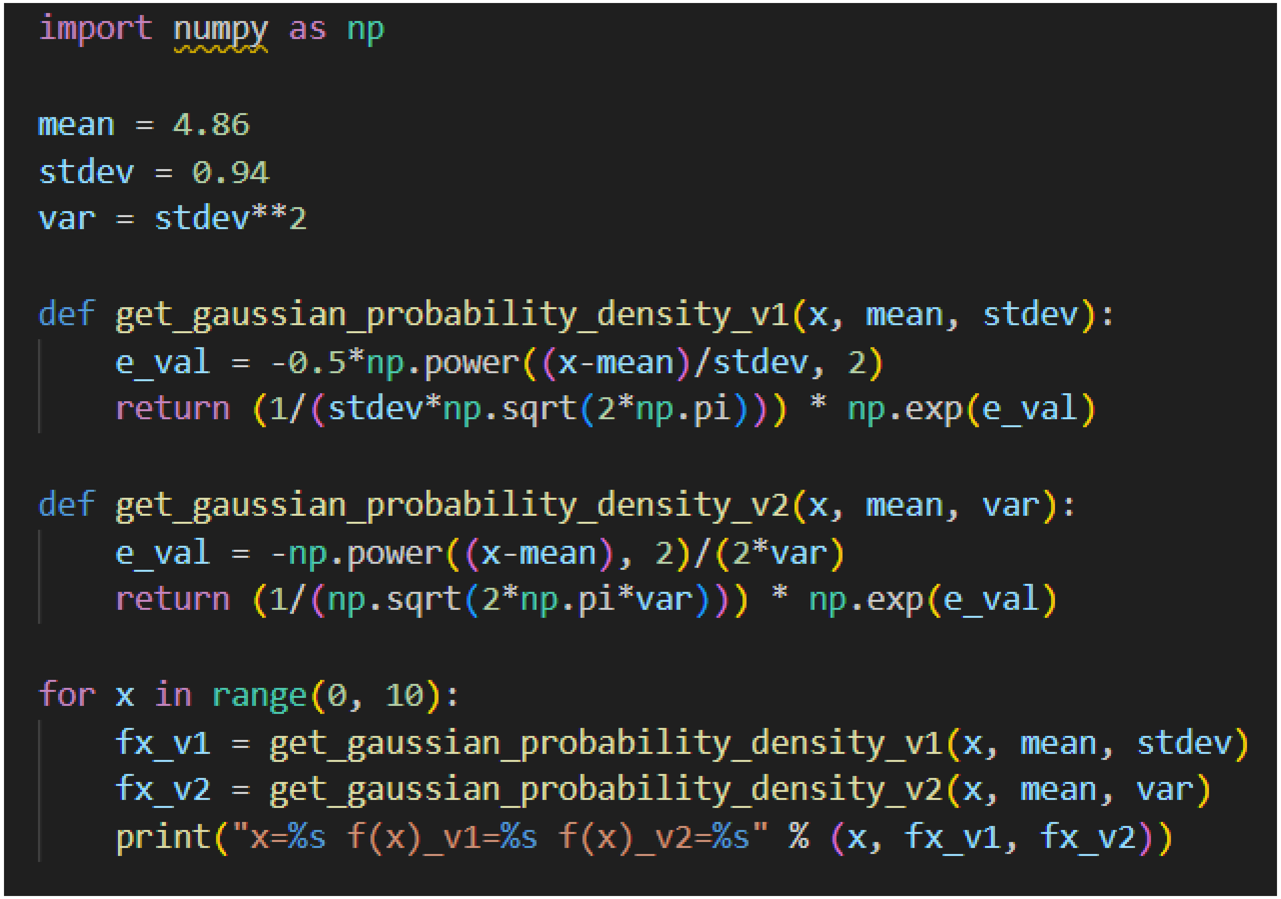
Also written as:

$$f(x) = \mathcal{N}(\mu, \sigma^2)$$

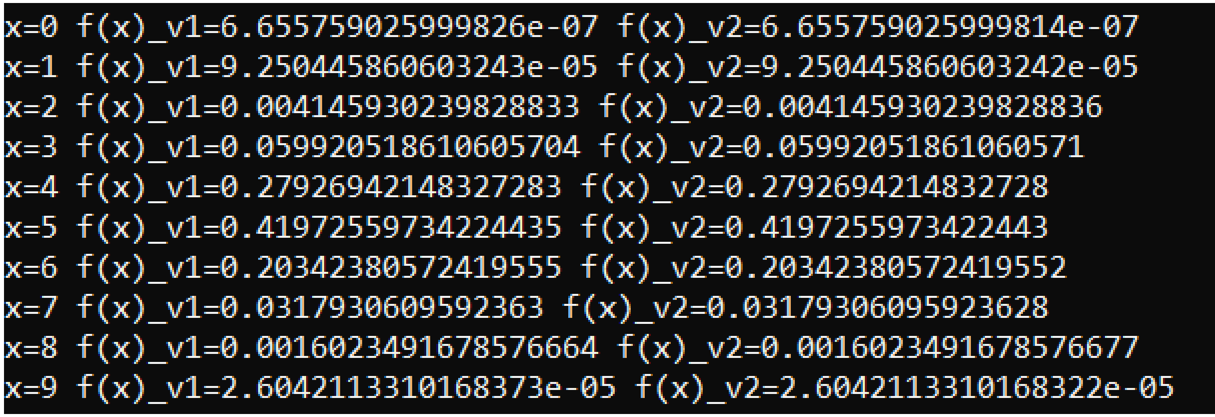
or

$$f(x) = \mathcal{N}(\mu, \sigma)$$

Univariate Gaussians

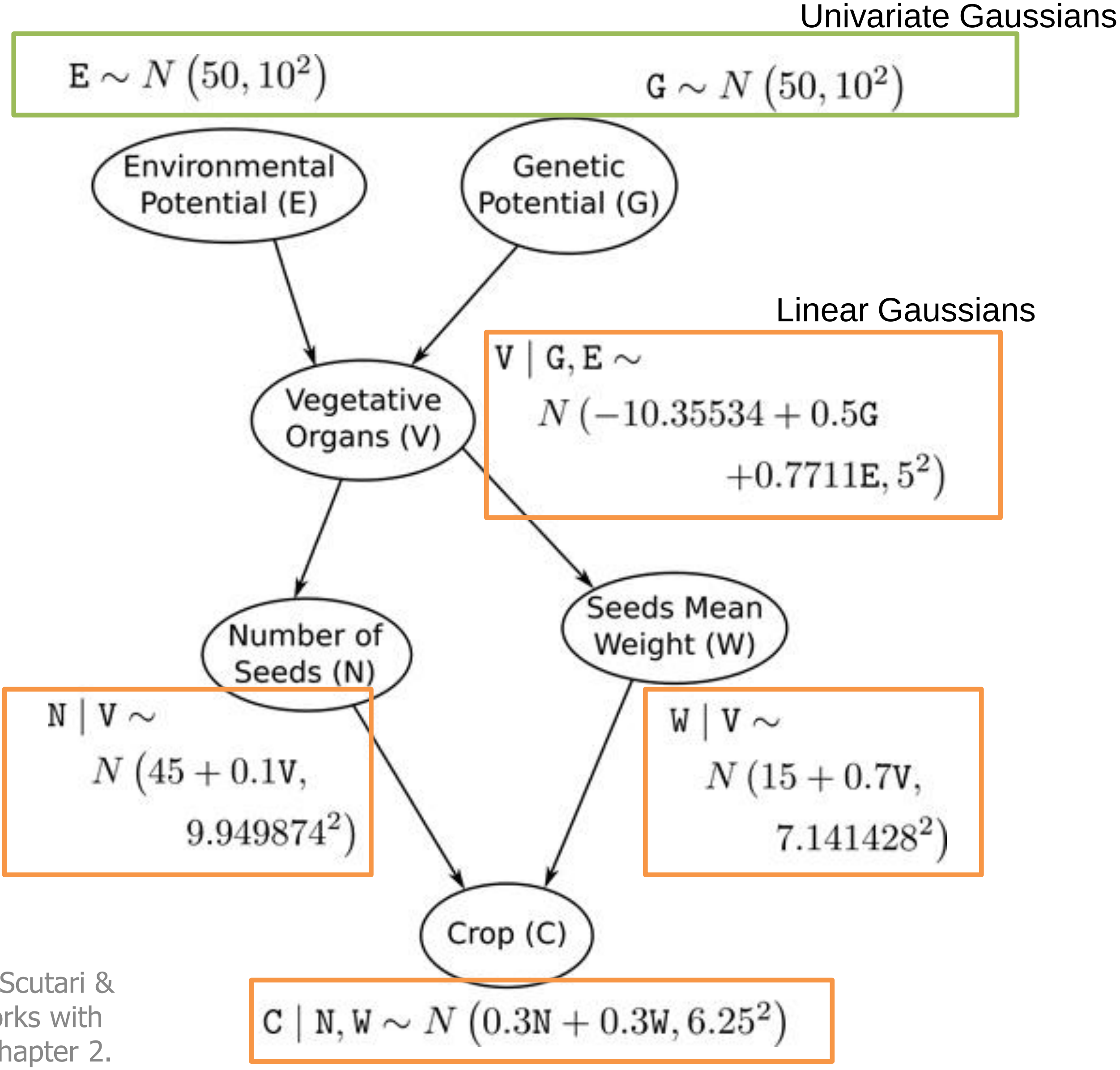


```python
import numpy as np

mean = 4.86
stdev = 0.94
var = stdev**2

def get_gaussian_probability_density_v1(x, mean, stdev):
    e_val = -0.5*np.power((x-mean)/stdev, 2)
    return (1/(stdev*np.sqrt(2*np.pi))) * np.exp(e_val)

def get_gaussian_probability_density_v2(x, mean, var):
    e_val = -np.power((x-mean), 2)/(2*var)
    return (1/(np.sqrt(2*np.pi*var))) * np.exp(e_val)

for x in range(0, 10):
    fx_v1 = get_gaussian_probability_density_v1(x, mean, stdev)
    fx_v2 = get_gaussian_probability_density_v2(x, mean, var)
    print("x=%s f(x)_v1=%s f(x)_v2=%s" % (x, fx_v1, fx_v2))
```



```
x=0 f(x)_v1=6.655759025999826e-07 f(x)_v2=6.655759025999814e-07
x=1 f(x)_v1=9.250445860603243e-05 f(x)_v2=9.250445860603242e-05
x=2 f(x)_v1=0.004145930239828833 f(x)_v2=0.004145930239828836
x=3 f(x)_v1=0.059920518610605704 f(x)_v2=0.05992051861060571
x=4 f(x)_v1=0.27926942148327283 f(x)_v2=0.2792694214832728
x=5 f(x)_v1=0.41972559734224435 f(x)_v2=0.4197255973422443
x=6 f(x)_v1=0.20342380572419555 f(x)_v2=0.20342380572419552
x=7 f(x)_v1=0.0317930609592363 f(x)_v2=0.03179306095923628
x=8 f(x)_v1=0.0016023491678576664 f(x)_v2=0.0016023491678576677
x=9 f(x)_v1=2.6042113310168373e-05 f(x)_v2=2.6042113310168322e-05
```

Example GBN: Revisited



Univariate Gaussians

$\mathtt{E} \sim N\left(50, 10^2\right)$

$\mathtt{G} \sim N\left(50, 10^2\right)$

Linear Gaussians

$$\mathtt{V} \mid \mathtt{G}, \mathtt{E} \sim N\left(-10.35534 + 0.5\mathtt{G} + 0.7711\mathtt{E}, 5^2\right)$$

$$\mathtt{N} \mid \mathtt{V} \sim N\left(45 + 0.1\mathtt{V}, 9.949874^2\right)$$

$$\mathtt{W} \mid \mathtt{V} \sim N\left(15 + 0.7\mathtt{V}, 7.141428^2\right)$$

$$\mathtt{C} \mid \mathtt{N}, \mathtt{W} \sim N\left(0.3\mathtt{N} + 0.3\mathtt{W}, 6.25^2\right)$$

Example network from Scutari & Denis, Bayesian Networks with Examples in R, 2022. Chapter 2.

Today

- Preliminaries
  - **Overview of linear algebra**
  - Multivariate Gaussian distributions
  - The kernel trick revisited

- Gaussian processes for regression (GPR)

- Gaussian processes for classification (GPC)

Notation of Linear Algebra

Symbol	Description
$\boldsymbol{y}^T$	The transpose of vector $\boldsymbol{y}$
$\boldsymbol{y}^{-1}$	The inverse of vector $\boldsymbol{y}$
$\boldsymbol{y}\boldsymbol{y}^{-1} = \boldsymbol{I}$	The identity matrix
$\backslash$	$\boldsymbol{a}\backslash\boldsymbol{b}$ is the vector $\boldsymbol{x}$ solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$
$\mathbf{0}$ or $\mathbf{0}_n$	The vector of all zeros of length n
$\lvert\boldsymbol{A}\rvert$	The determinant of matrix $\boldsymbol{A}$
$diag(\boldsymbol{y})$	Dialogonal matrix containing the elements of $\boldsymbol{y}$
$cholesky\ (\boldsymbol{A})$	L is a lower diagonal matrix such that $LL^T = \boldsymbol{A}$

Note: We will use libraries like numpy or scipy to carry out such operations.

Linear Algebra: Transpose

The transpose of vector $y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix}$ is $y^T = (y_1, y_2, y_3)$

Consider vector $A = \begin{pmatrix} 4 & 2 & -2 \\ 2 & 5 & -5 \\ -2 & -5 & 8 \end{pmatrix}$

In A^T the rows become columns, giving us

$$A^T = \begin{pmatrix} 4 & 2 & -2 \\ 2 & 5 & -5 \\ -2 & -5 & 8 \end{pmatrix}^T = \begin{pmatrix} 4 & 2 & -2 \\ 2 & 5 & -5 \\ -2 & -5 & 8 \end{pmatrix}$$

Linear Algebra: Identity Matrix

Consider vector $A = \begin{pmatrix} 4 & 2 & -2 \\ 2 & 5 & -5 \\ -2 & -5 & 8 \end{pmatrix}$

The identify of A is $AA^{-1} = I = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Linear Algebra: Determinant

Consider vector $A = \begin{pmatrix} 4 & 2 & -2 \\ 2 & 5 & -5 \\ -2 & -5 & 8 \end{pmatrix}$

$$\det(A) = 4 \cdot \begin{vmatrix} 5 & -5 \\ -5 & 8 \end{vmatrix} - 2 \cdot \begin{vmatrix} 2 & -5 \\ -2 & 8 \end{vmatrix} + (-2) \cdot \begin{vmatrix} 2 & 5 \\ -2 & -5 \end{vmatrix}$$

$$\begin{vmatrix} 5 & -5 \\ -5 & 8 \end{vmatrix} = (5 \cdot 8) - (-5 \cdot -5) = 40 - 25 = 15$$

$$\begin{vmatrix} 2 & -5 \\ -2 & 8 \end{vmatrix} = (2 \cdot 8) - (-5 \cdot -2) = 16 - 10 = 6$$

$$\begin{vmatrix} 2 & 5 \\ -2 & -5 \end{vmatrix} = (2 \cdot -5) - (5 \cdot -2) = 10 - 10 = 0$$

The determinant of A is $|A| = 4 \cdot 15 - 2 \cdot 6 + (-2) \cdot 0 = 48$

Linear Algebra: Inverse

Consider vector $A = \begin{pmatrix} 4 & 2 & -2 \\ 2 & 5 & -5 \\ -2 & -5 & 8 \end{pmatrix}$

Adjugate

$$A^{-1} = \frac{1}{\det(A)}\mathrm{Adj}(A) = \frac{1}{48}\mathrm{Adj}(A)$$

$$\mathrm{Adj}(A) = \mathrm{Cofactor}(A)^T = \begin{pmatrix} 15 & -6 & 0 \\ -6 & 28 & 16 \\ 0 & 16 & 16 \end{pmatrix}^T = \begin{pmatrix} 15 & -6 & 0 \\ -6 & 28 & 16 \\ 0 & 16 & 16 \end{pmatrix}$$

The inverse of A is $A^{-1} = \begin{pmatrix} 0.3125 & -0.125 & 0 \\ -0.125 & 0.5833 & 0.3333 \\ 0 & 0.3333 & 0.3333 \end{pmatrix}$

Linear Algebra: $a\backslash\mathrm{b}\ (Ax = b)$

Consider vector $b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$

$$x = A^{-1}b = \begin{pmatrix} 0.3125 & -0.125 & 0 \\ -0.125 & 0.5833 & 0.3333 \\ 0 & 0.3333 & 0.3333 \end{pmatrix} \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0.1875 \\ 0.7917 \\ 0.6667 \end{pmatrix}$$

Linear Algebra: Zero Vector

The zero vector of 3 elements is $O_3 = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}$

The zero matrix of 2 × 3 elements is $O_{2\times 3} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$

Linear Algebra: Diagonal Matrix

Consider vector $y = \begin{pmatrix} 1 \\ 3 \\ 2 \end{pmatrix}$

The diagonal of y is $\mathrm{diag}(y) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 2 \end{pmatrix}$

Linear Algebra: Cholesky Decomposition

Consider vector $A = \begin{pmatrix} 4 & 2 & -2 \\ 2 & 5 & -5 \\ -2 & -5 & 8 \end{pmatrix}$ such that $A = LL^T$

The Cholesky decomposition for A gives $L = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ -1 & -2 & \sqrt{3} \end{pmatrix}$

Thus,

$$A = \begin{pmatrix} 2 & 0 & 0 \\ 1 & 2 & 0 \\ -1 & -2 & \sqrt{3} \end{pmatrix} \begin{pmatrix} 2 & 1 & -1 \\ 0 & 2 & -2 \\ 0 & 0 & \sqrt{3} \end{pmatrix} = \begin{pmatrix} 4 & 2 & -2 \\ 2 & 5 & -5 \\ -2 & -5 & 8 \end{pmatrix}$$

Notation of Linear Algebra: Numpy

Symbol	Description	Numpy Function
$\boldsymbol{y}^T$	The transpose of vector $\boldsymbol{y}$	y.T
$\boldsymbol{y}^{-1}$	The inverse of vector $\boldsymbol{y}$	np.linalg.inv(y)
$\boldsymbol{y}\boldsymbol{y}^{-1} = \boldsymbol{I}$	The identity matrix	np.identity(y)
$\backslash$	$\boldsymbol{a}\backslash\boldsymbol{b}$ is the vector $\boldsymbol{x}$ solving $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$	np.linalg.solve(a, b)
$\mathbf{0}$ or $\mathbf{0}_n$	The vector of all zeros of length n	np.zeros()
$\lvert\boldsymbol{A}\rvert$	The determinant of matrix $\boldsymbol{A}$	np.linalg.det(A)
$diag(\boldsymbol{y})$	Dialogonal matrix of $\boldsymbol{y}$	np.diag(y)
$cholesky\ (\boldsymbol{A})$	For lower diagonal matrix L, $LL^T = \boldsymbol{A}$	np.linalg.cholesky(A)

Assumption: the above numpy functions receive numpy arrays as inputs.

Today

- Preliminaries
  - Overview of linear algebra
  - **Multivariate Gaussian distributions**
  - The kernel trick revisited
- Gaussian processes for regression (GPR)
- Gaussian processes for classification (GPC)

Multivariate Gaussian Distribution

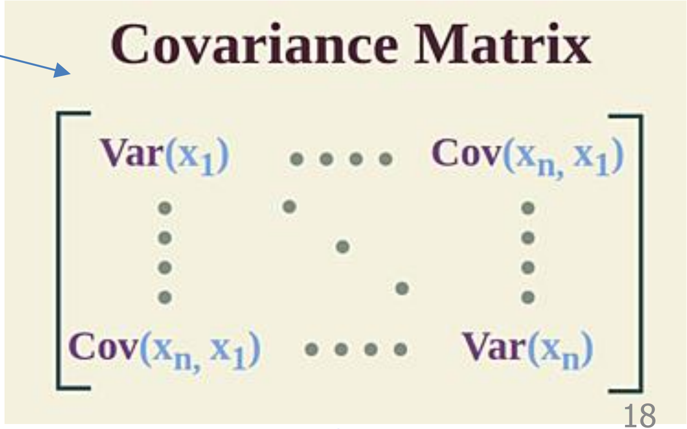
A multivariate Gaussian is a generalisation of the 1D Gaussian to multiple continuous random variables, defined as

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

Notation:

D = dimentionality (number of random variables)
μ =mean vector (D real numbers)
$\sum$ =covariance matrix ($D \times D$),
where $\Sigma_{ij} = \text{cov}(x_i, x_j)$

Covariance=how to random variables differ
+ cov: variables head in same direction
- cov: variables head in opposite direction
0 cov: no relationship between variables



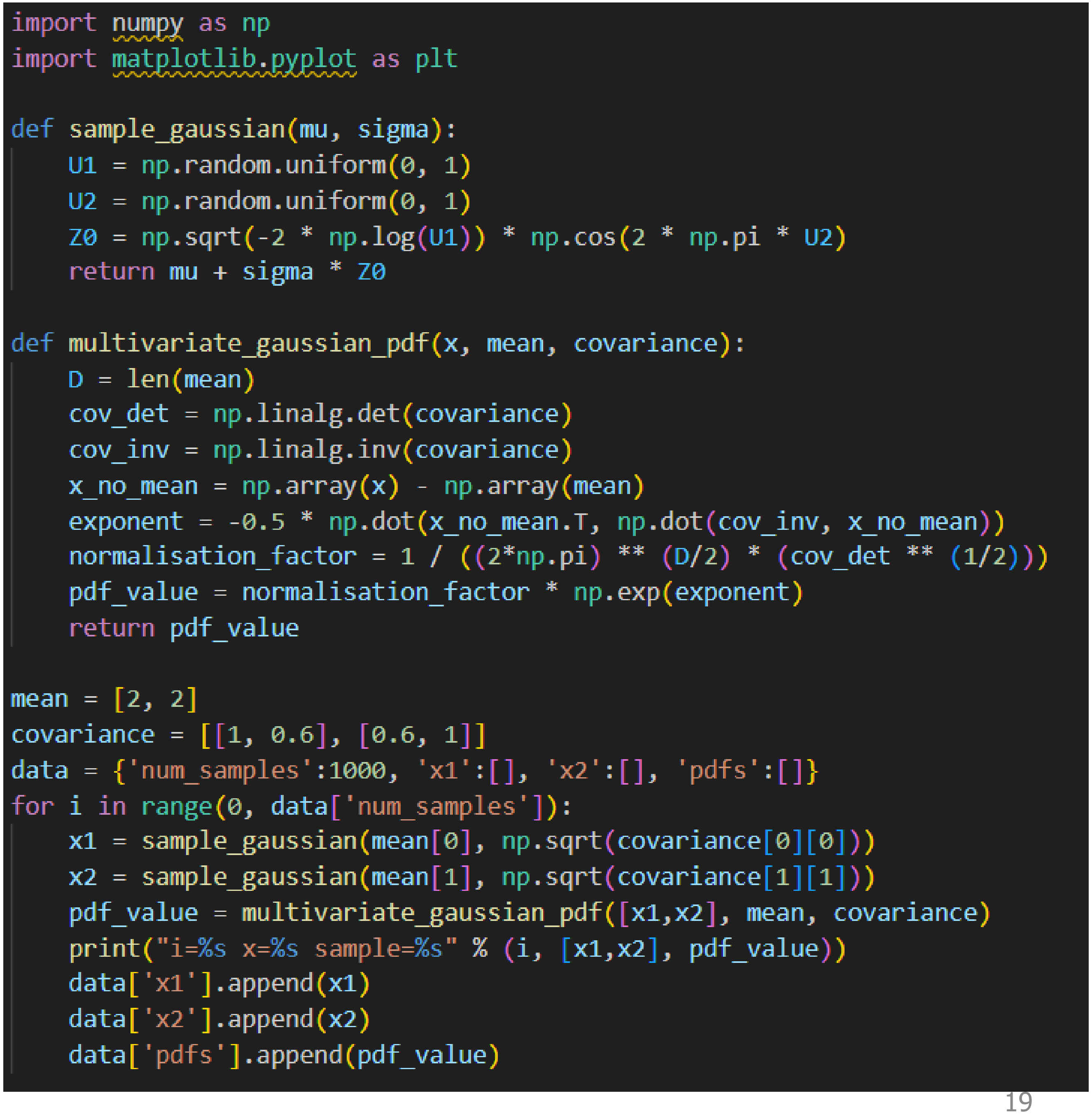
Covariance Matrix

$$\begin{bmatrix} \mathrm{Var}(x_1) & \cdots & \mathrm{Cov}(x_n, x_1) \\ \vdots & \ddots & \vdots \\ \mathrm{Cov}(x_n, x_1) & \cdots & \mathrm{Var}(x_n) \end{bmatrix}$$

Source

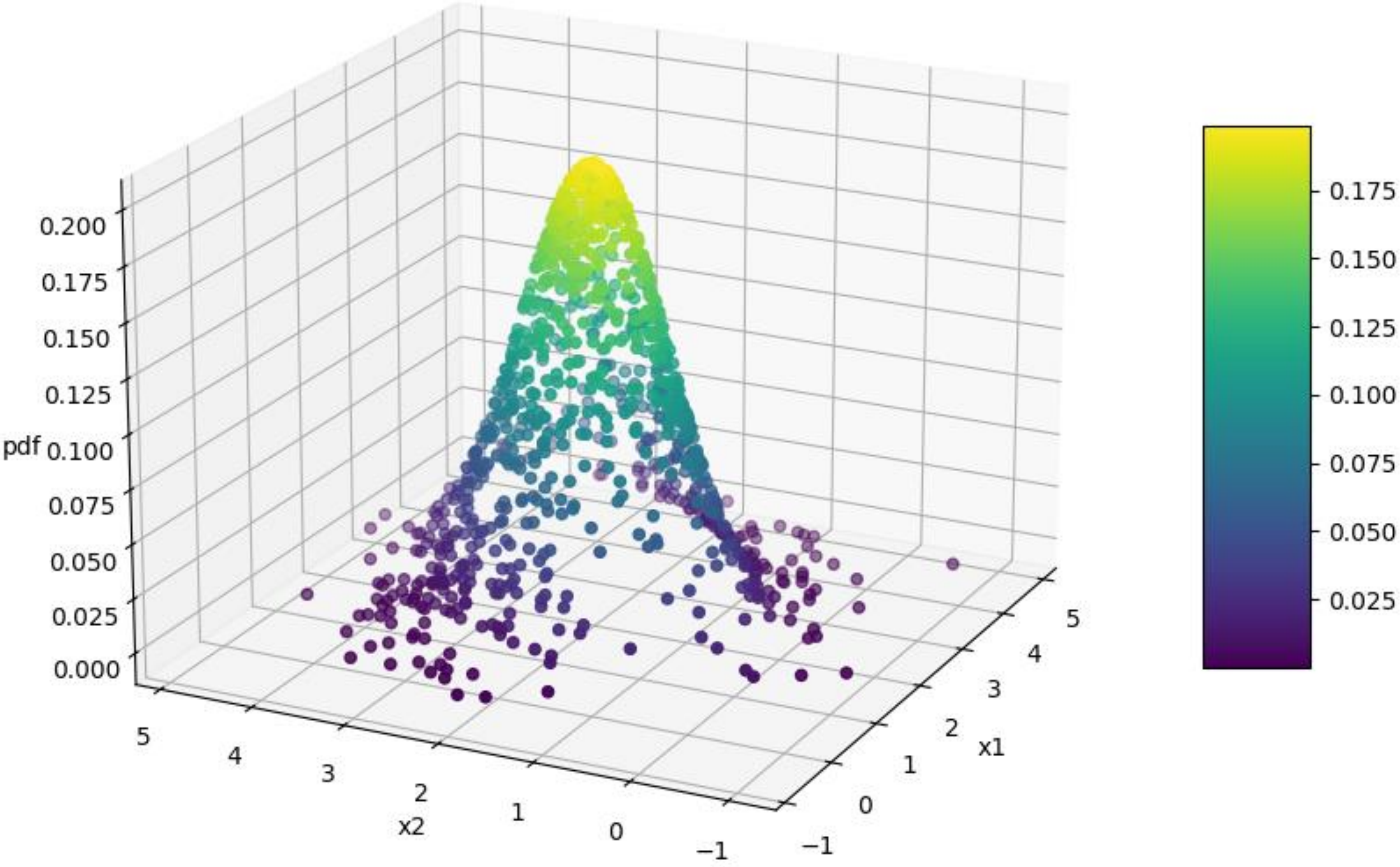
$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}}\exp\left[-\frac{1}{2}(x-\mu)^{\top}\Sigma^{-1}(x-\mu)\right]$$

➔



```python
import numpy as np
import matplotlib.pyplot as plt

def sample_gaussian(mu, sigma):
    U1 = np.random.uniform(0, 1)
    U2 = np.random.uniform(0, 1)
    Z0 = np.sqrt(-2 * np.log(U1)) * np.cos(2 * np.pi * U2)
    return mu + sigma * Z0

def multivariate_gaussian_pdf(x, mean, covariance):
    D = len(mean)
    cov_det = np.linalg.det(covariance)
    cov_inv = np.linalg.inv(covariance)
    x_no_mean = np.array(x) - np.array(mean)
    exponent = -0.5 * np.dot(x_no_mean.T, np.dot(cov_inv, x_no_mean))
    normalisation_factor = 1 / ((2*np.pi) ** (D/2) * (cov_det ** (1/2)))
    pdf_value = normalisation_factor * np.exp(exponent)
    return pdf_value

mean = [2, 2]
covariance = [[1, 0.6], [0.6, 1]]
data = {'num_samples':1000, 'x1':[], 'x2':[], 'pdfs':[]}
for i in range(0, data['num_samples']):
    x1 = sample_gaussian(mean[0], np.sqrt(covariance[0][0]))
    x2 = sample_gaussian(mean[1], np.sqrt(covariance[1][1]))
    pdf_value = multivariate_gaussian_pdf([x1,x2], mean, covariance)
    print("i=%s x=%s sample=%s" % (i, [x1,x2], pdf_value))
    data['x1'].append(x1)
    data['x2'].append(x2)
    data['pdfs'].append(pdf_value)
```

Output of Previous Implementation



Multivariate Gaussian Distribution: Marginals and Conditionals

Consider a Gaussian with the following mean and covariance

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$$

Marginals:

$$P(x_1) = \mathcal{N}(x_1|\mu_1, \Sigma_{11})$$
$$P(x_2) = \mathcal{N}(x_2|\mu_2, \Sigma_{22})$$

Conditional:

$$P(x_1|x_2) = \mathcal{N}(x_1|\mu_{1|2}, \Sigma_{1|2})$$
$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2)$$
$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21}$$

For further information on marginals and conditionals of multivariate Gaussians, see Bishop. C. Pattern Recognition and Machine Learning. 2006 – sections 2.3.1 and 2.3.2.

Multivariate Gaussian Distribution: Example of Marginals and Conditionals

Consider $x_1 = 2.8$, $x_2 = 3.5$, and mean and covariance

$$\mu = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} 4 & 2 \\ 2 & 5 \end{pmatrix}$$

Marginals:

$$P(x_1) = \mathcal{N}(x_1|\mu_1, \Sigma_{11}) = \mathcal{N}(2.8;\ 2, 4) = 0.18413$$

$$P(x_2) = \mathcal{N}(x_2|\mu_2, \Sigma_{22}) = \mathcal{N}(3.5;\ 3, 5) = 0.17400$$

Conditional:

$$P(x_1|x_2) = \mathcal{N}\left(x_1\middle|\mu_{1|2}, \Sigma_{1|2}\right) = \mathcal{N}(2.8; 2.2, 3.2) = 0.21081$$

$$\mu_{1|2} = \mu_1 + \Sigma_{12}\Sigma_{22}^{-1}(x_2 - \mu_2) = 2 + 2 \times \frac{1}{5} \times (3.5 - 3) = 2.2$$

$$\Sigma_{1|2} = \Sigma_{11} - \Sigma_{12}\Sigma_{22}^{-1}\Sigma_{21} = 4 - 2 \times \frac{1}{5} \times 2 = 3.2$$

Multivariate Gaussian Distribution: Marginals and Conditionals

Consider a Gaussian with the following mean and covariance

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}$$

Marginals:

$$P(x_1) = \mathcal{N}(x_1 | \mu_1, \Sigma_{11})$$
$$P(x_2) = \mathcal{N}(x_2 | \mu_2, \Sigma_{22})$$

Conditional:

$$P(x_2 | x_1) = \mathcal{N}(x_2 | \mu_{2|1}, \Sigma_{2|1})$$
$$\mu_{2|1} = \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(x_1 - \mu_1)$$
$$\Sigma_{2|1} = \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}$$

Note: Vectors and matrices can use bold or non-bold fonts in the literature.

Multivariate Gaussian Distribution: Example of Marginals and Conditionals

Consider $x_1 = 2.8$, $x_2 = 3.5$, and mean and covariance

$$\mu = \begin{pmatrix} 2 \\ 3 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} 4 & 2 \\ 2 & 5 \end{pmatrix}$$

Marginals:

$$P(x_1) = \mathcal{N}(x_1 | \mu_1, \Sigma_{11}) = \mathcal{N}(2.8;\ 2, 4) = 0.18413$$

$$P(x_2) = \mathcal{N}(x_2 | \mu_2, \Sigma_{22}) = \mathcal{N}(3.5;\ 3, 5) = 0.17400$$

Conditional:

$$P(x_2 | x_1) = \mathcal{N}(x_2 | \mu_{2|1}, \Sigma_{2|1}) = \mathcal{N}(3.5; 3.4, 4.0) = 0.19922$$

$$\mu_{2|1} = \mu_2 + \Sigma_{21}\Sigma_{11}^{-1}(x_1 - \mu_1) = 3 + 2 \times \frac{1}{4} \times (2.8 - 2) = 3.4$$

$$\Sigma_{2|1} = \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12} = 5 - 2 \times \frac{1}{4} \times 2 = 4.0$$

Multivariate Gaussians: Conditionals

Consider a Gaussian with the following mean and covariance

$$\mu = \begin{pmatrix} \mu_1 \\ \mu_2 \\ \mu_3 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} & \Sigma_{13} \\ \Sigma_{21} & \Sigma_{22} & \Sigma_{23} \\ \Sigma_{31} & \Sigma_{32} & \Sigma_{33} \end{pmatrix}$$

Partitioning μ and Σ, by separating x_3, we get

$$\mu = \begin{pmatrix} \begin{pmatrix} \mu_1 \\ \mu_2 \end{pmatrix} \\ \mu_3 \end{pmatrix} = \begin{pmatrix} \mu_{1,2} \\ \mu_3 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \Sigma_{1,2} & \Sigma_{1,2,3} \\ \Sigma_{3,1,2} & \Sigma_{33} \end{pmatrix}$$

Conditional:

$$P(x_3|x_1, x_2) = \mathcal{N}(x_3|\mu_{3|1,2}, \Sigma_{3|1,2})$$

$$\mu_{3|1,2} = \mu_3 + \Sigma_{3,1,2}\Sigma_{1,2}^{-1}(x_{1,2} - \mu_{1,2})$$

$$\Sigma_{3|1,2} = \Sigma_{33} - \Sigma_{3,1,2}\Sigma_{1,2}^{-1}\Sigma_{1,2,3}$$

Multivariate Gaussian Distributions: Another Example of Conditionals

Consider the following:

$$x = \begin{pmatrix} x_1 = 1 \\ x_2 = 2 \\ x_3 = 3 \end{pmatrix} \qquad \mu = \begin{pmatrix} 1 \\ -3 \\ 4 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} 4 & 2 & -2 \\ 2 & 5 & -5 \\ -2 & -5 & 8 \end{pmatrix}$$

Partitioning μ and Σ, by separating x_3, we get

$$\mu = \begin{pmatrix} \begin{pmatrix} 1 \\ -3 \end{pmatrix} \\ 4 \end{pmatrix} = \begin{pmatrix} \mu_{1,2} \\ \mu_3 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \Sigma_{1,2} & \Sigma_{1,2,3} \\ \Sigma_{3,1,2} & \Sigma_{33} \end{pmatrix}$$

Conditional:

$$P(x_3|x_1, x_2) = \mathcal{N}\left(x_3 \middle| \mu_{3|1,2}, \Sigma_{3|1,2}\right) = ?$$

$$\mu_{3|1,2} = \mu_3 + \Sigma_{3,1,2}\Sigma_{1,2}^{-1}\left(x_{1,2} - \mu_{1,2}\right) = ?$$

$$\Sigma_{3|1,2} = \Sigma_{33} - \Sigma_{3,1,2}\Sigma_{1,2}^{-1}\Sigma_{1,2,3} = ?$$

Multivariate Gaussians: Conditionals

Consider a Gaussian with the following mean and covariance

$$\mu = \begin{pmatrix} \mu_1 \\ \cdots \\ \mu_5 \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} & \dots & \Sigma_{15} \\ \Sigma_{21} & \Sigma_{22} & \dots & \Sigma_{25} \\ \dots & \dots & & \dots \\ \Sigma_{51} & \Sigma_{52} & \dots & \Sigma_{55} \end{pmatrix}$$

Partitioning μ and Σ, by separating x_4 and x_5, we get

$$\mu = \begin{pmatrix} \mu_{1:3} \\ \mu_{4:5} \end{pmatrix} \qquad \Sigma = \begin{pmatrix} \Sigma_{1:3,1:3} & \Sigma_{1:3,4:5} \\ \Sigma_{4:5,1:3} & \Sigma_{4:5.4:5} \end{pmatrix}$$

Conditional:

$$P(x_4, x_5 | x_1, x_2, x_3) = \mathcal{N}(x_4, x_5 | \mu_{4:5|1:3}, \Sigma_{4:5|1:3})$$

$$\mu_{4:5|1:3} = \mu_{4:5} + \Sigma_{4:5,1:3}\Sigma_{1:3,1:3}^{-1}(x_{1:3} - \mu_{1:3})$$

$$\Sigma_{4:5|1:3} = \Sigma_{4:5,4:5} - \Sigma_{4:5,1:3}\Sigma_{1:3,1:3}^{-1}\Sigma_{1:3,4:5}$$

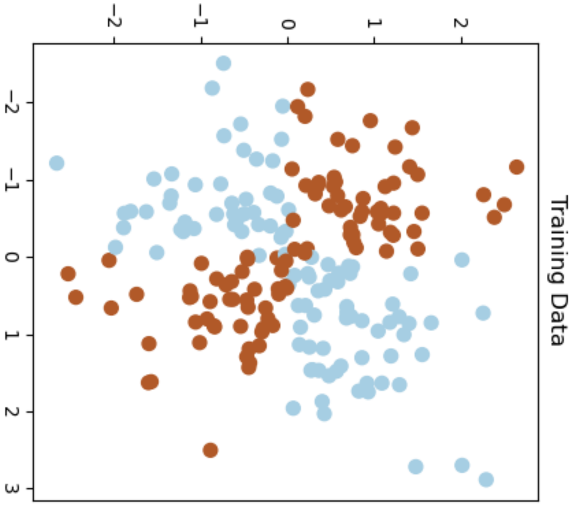
Today

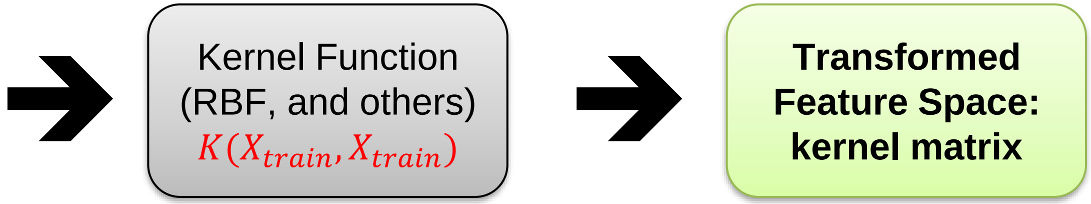
- Preliminaries
  - Overview of linear algebra
  - Multivariate Gaussian distributions
  - **The kernel trick revisited**

- Gaussian processes for regression (GPR)

- Gaussian processes for classification (GPC)

Revisiting the Kernel Trick

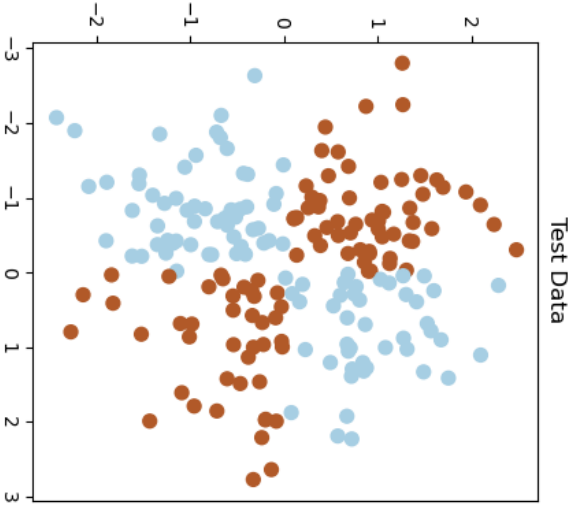


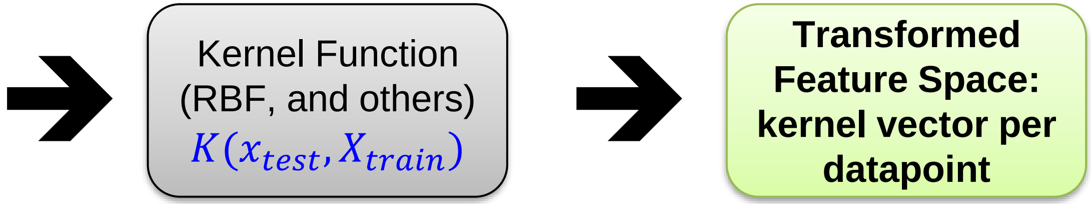

Kernel Function (RBF, and others) $K(X_{train}, X_{train})$

Transformed Feature Space: kernel matrix

RBF Kernel: $K(x_i, x_j) = \exp(-\gamma \left|\left|x_i - x_j\right|\right|^2$

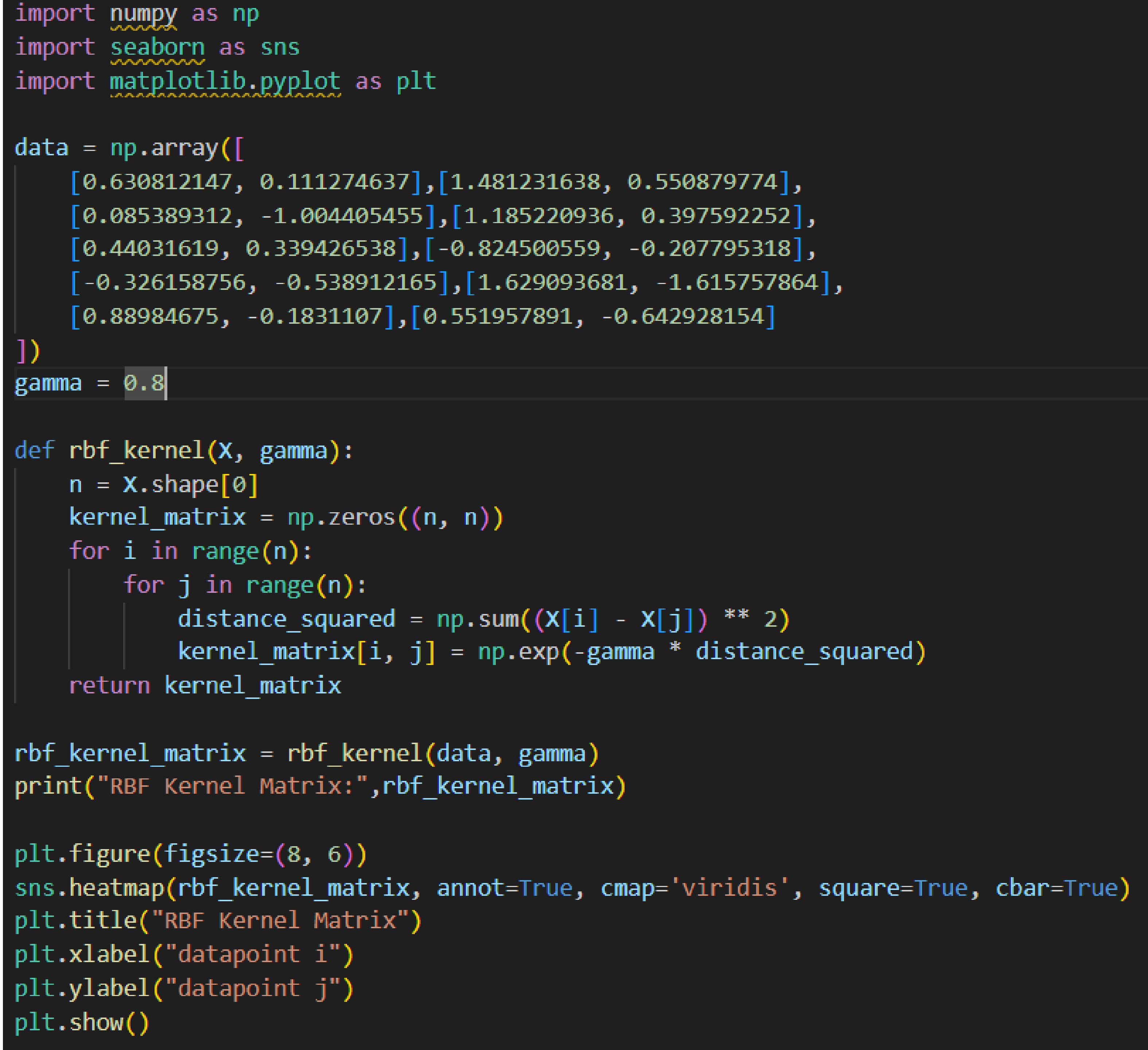
Adjusts the similarity between datapoints




Kernel Function (RBF, and others) $K(x_{test}, X_{train})$

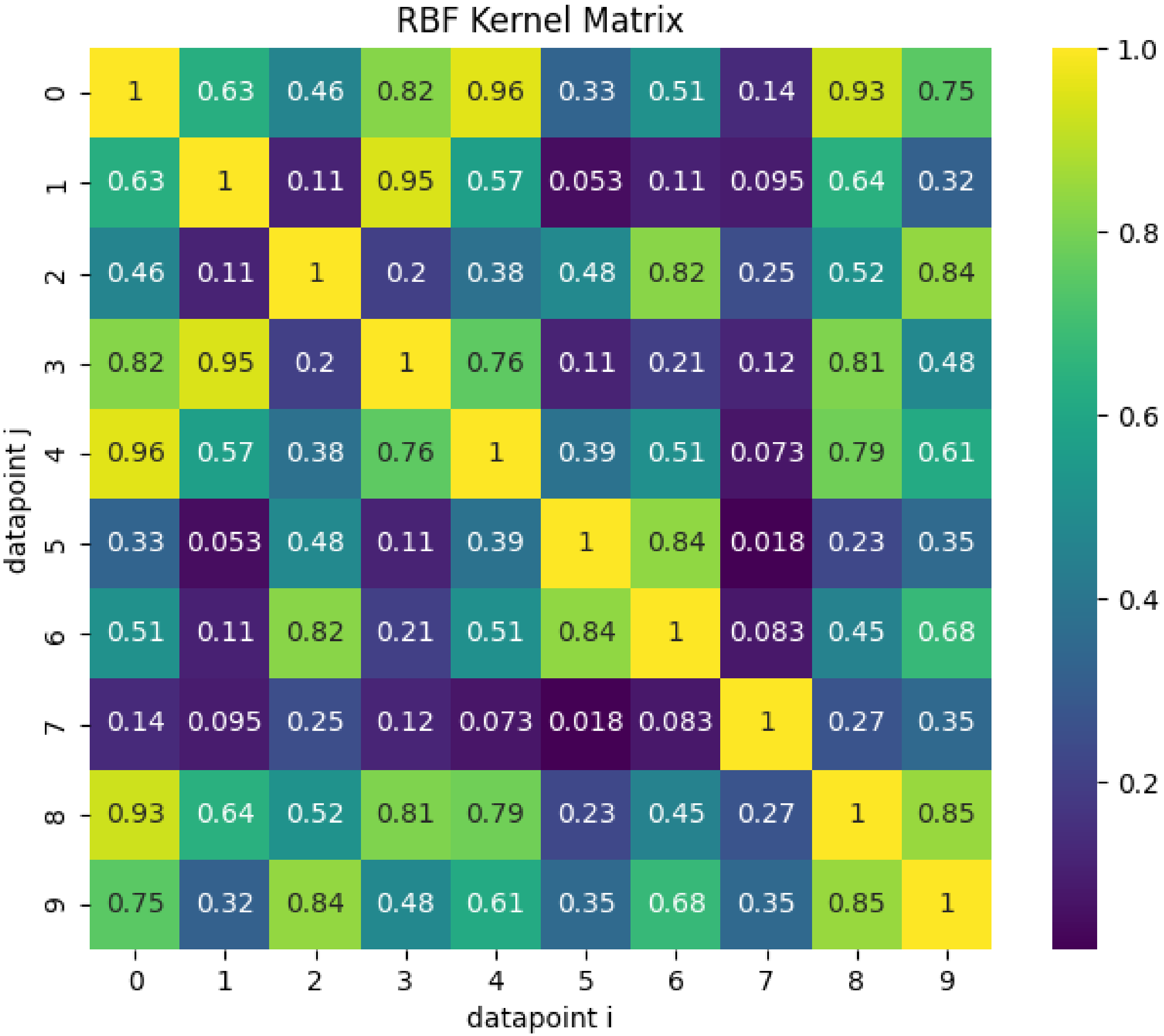
Transformed Feature Space: kernel vector per datapoint

The Kernel Trick: Example Implementation on Toy Data



```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

data = np.array([
    [0.630812147, 0.111274637],[1.481231638, 0.550879774],
    [0.085389312, -1.004405455],[1.185220936, 0.397592252],
    [0.44031619, 0.339426538],[-0.824500559, -0.207795318],
    [-0.326158756, -0.538912165],[1.629093681, -1.615757864],
    [0.88984675, -0.1831107],[0.551957891, -0.642928154]
])
gamma = 0.8

def rbf_kernel(X, gamma):
    n = X.shape[0]
    kernel_matrix = np.zeros((n, n))
    for i in range(n):
        for j in range(n):
            distance_squared = np.sum((X[i] - X[j]) ** 2)
            kernel_matrix[i, j] = np.exp(-gamma * distance_squared)
    return kernel_matrix

rbf_kernel_matrix = rbf_kernel(data, gamma)
print("RBF Kernel Matrix:",rbf_kernel_matrix)

plt.figure(figsize=(8, 6))
sns.heatmap(rbf_kernel_matrix, annot=True, cmap='viridis', square=True, cbar=True)
plt.title("RBF Kernel Matrix")
plt.xlabel("datapoint i")
plt.ylabel("datapoint j")
plt.show()
```

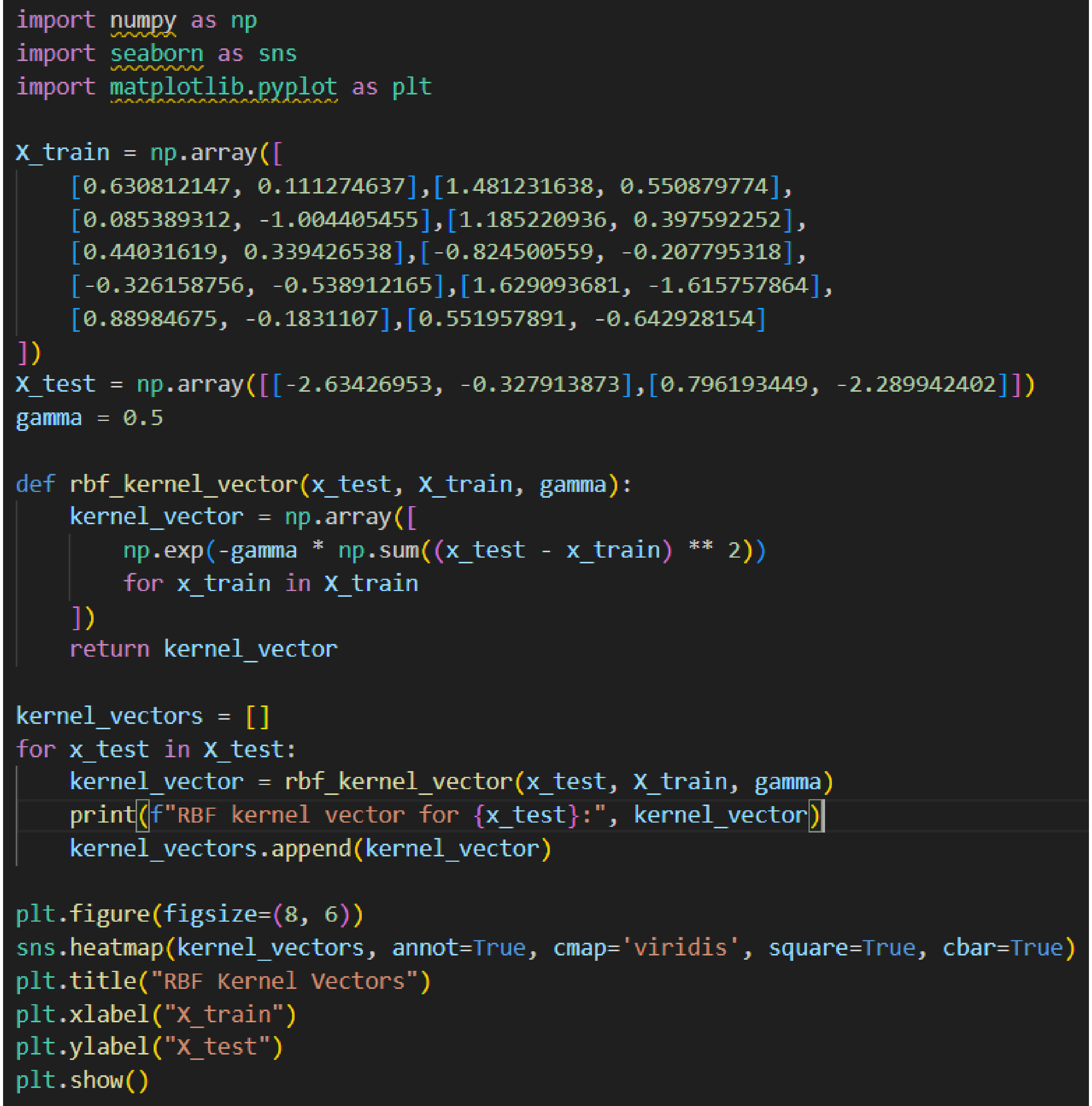
The Kernel Trick: Output of Implementation on Toy Data



RBF Kernel Matrix

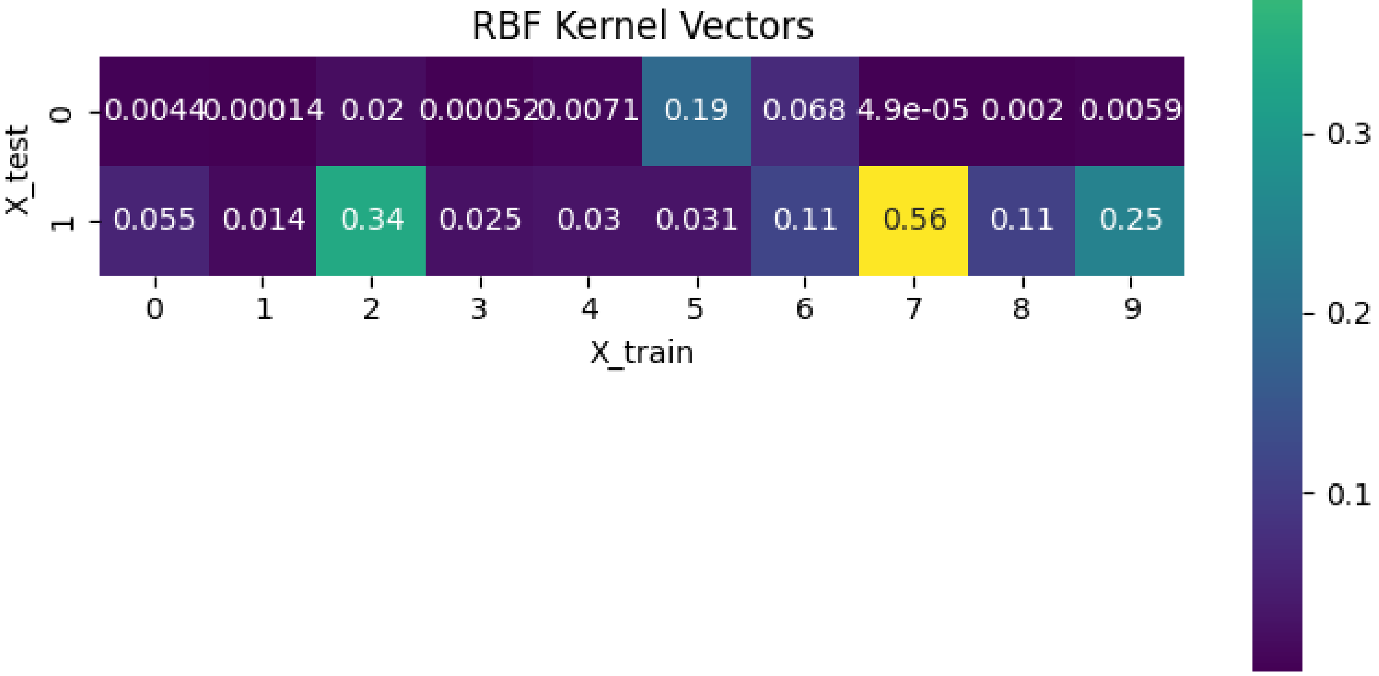
datapoint j \ datapoint i	0	1	2	3	4	5	6	7	8	9
0	1	0.63	0.46	0.82	0.96	0.33	0.51	0.14	0.93	0.75
1	0.63	1	0.11	0.95	0.57	0.053	0.11	0.095	0.64	0.32
2	0.46	0.11	1	0.2	0.38	0.48	0.82	0.25	0.52	0.84
3	0.82	0.95	0.2	1	0.76	0.11	0.21	0.12	0.81	0.48
4	0.96	0.57	0.38	0.76	1	0.39	0.51	0.073	0.79	0.61
5	0.33	0.053	0.48	0.11	0.39	1	0.84	0.018	0.23	0.35
6	0.51	0.11	0.82	0.21	0.51	0.84	1	0.083	0.45	0.68
7	0.14	0.095	0.25	0.12	0.073	0.018	0.083	1	0.27	0.35
8	0.93	0.64	0.52	0.81	0.79	0.23	0.45	0.27	1	0.85
9	0.75	0.32	0.84	0.48	0.61	0.35	0.68	0.35	0.85	1

The Kernel Trick: Example Implementation on Test Data



```python
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

X_train = np.array([
    [0.630812147, 0.111274637],[1.481231638, 0.550879774],
    [0.085389312, -1.004405455],[1.185220936, 0.397592252],
    [0.44031619, 0.339426538],[-0.824500559, -0.207795318],
    [-0.326158756, -0.538912165],[1.629093681, -1.615757864],
    [0.88984675, -0.1831107],[0.551957891, -0.642928154]
])
X_test = np.array([[-2.63426953, -0.327913873],[0.796193449, -2.289942402]])
gamma = 0.5

def rbf_kernel_vector(x_test, X_train, gamma):
    kernel_vector = np.array([
        np.exp(-gamma * np.sum((x_test - x_train) ** 2))
        for x_train in X_train
    ])
    return kernel_vector

kernel_vectors = []
for x_test in X_test:
    kernel_vector = rbf_kernel_vector(x_test, X_train, gamma)
    print(f"RBF kernel vector for {x_test}:", kernel_vector)
    kernel_vectors.append(kernel_vector)

plt.figure(figsize=(8, 6))
sns.heatmap(kernel_vectors, annot=True, cmap='viridis', square=True, cbar=True)
plt.title("RBF Kernel Vectors")
plt.xlabel("X_train")
plt.ylabel("X_test")
plt.show()
```

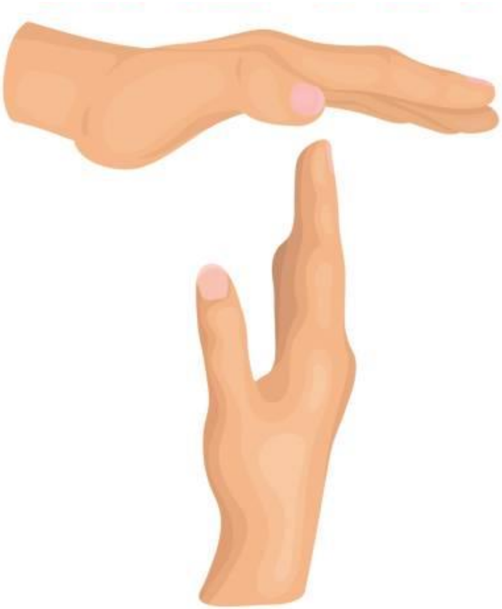
The Kernel Trick: Output of Implementation on Test Data



Linear Regression with Kernelised Data

- A kernel function is needed (such as RBF)
- Training consists of computing the kernel matrix $K(X_{train}, X_{train})$ and applying a loss function to learn coefficients (weights) β as discussed last week.
- Testing consists of computing the kernel vector $K(x_{test}, X_{train})$ and calculating an output value according to $\hat{y} = K(x_{test}, X_{train})^T \beta$

Break



Today

- Preliminaries
  - Overview of linear algebra
  - Multivariate Gaussian distributions
  - The kernel trick revisited

- **Gaussian processes for regression (GPR)**

- Gaussian processes for classification (GPC)

Gaussian Processes for Regression

Consider dataset $\{(\boldsymbol{x}_i, y_i) | i = 1, \ldots, n\}$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ denotes data point i and $y_i \in \mathbb{R}$ denotes its corresponding target value.

A Gaussian Process (GP) is a collection of random variables, any finite number of which have a joint Gaussian $p(f(\boldsymbol{x}1), \ldots, f(\boldsymbol{x}_n))$

$$p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\boldsymbol{\mu}, \mathbf{K}) \quad \text{or} \quad \boldsymbol{f} = \mathcal{GP}(\boldsymbol{\mu}, \boldsymbol{K})$$

with

- mean vector $\boldsymbol{\mu} = (\mu(\boldsymbol{x}_1), \ldots, \mu(\boldsymbol{x}_n))$
- covariance matrix (derived from a kernel function) $\mathbf{K}_{\boldsymbol{ij}} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$
- function values $\mathbf{f}$ (define the underlying function being modelled)
- data inputs $\mathbf{X}$
- $\boldsymbol{y}$ can be seen as $\mathbf{y} = \boldsymbol{f} + \epsilon$, where ϵ is Gaussian noise.

Gaussian Processes for Regression

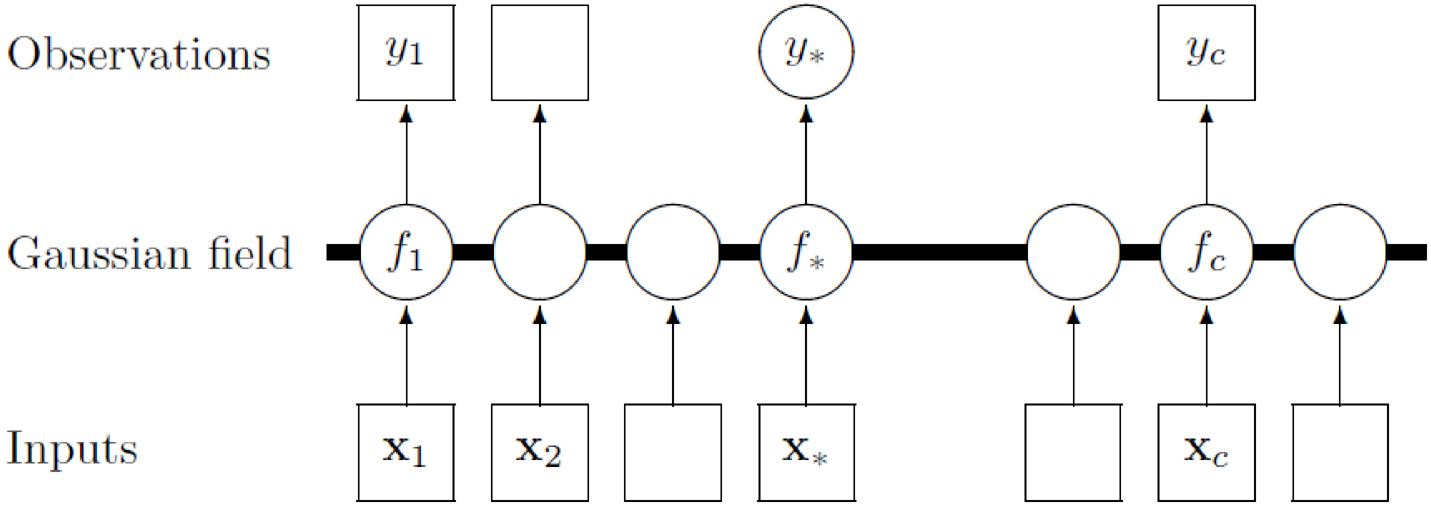


Figure 2.3: Graphical model (chain graph) for a GP for regression. Squares represent observed variables and circles represent unknowns. The thick horizontal bar represents a set of fully connected nodes. Note that an observation y_i is conditionally independent of all other nodes given the corresponding latent variable, f_i. Because of the marginalization property of GPs addition of further inputs, $\mathbf{x}$, latent variables, f, and *unobserved* targets, y_*, does not change the distribution of any other variables.

Image above from Rasmussen & Williams. Gaussian Processes for Machine Learning. MIT Press, 2006. Chapter 2

Gaussian Processes for Regression

GP prior $p(\mathbf{f}|\mathbf{X})$ can be converted into GP posterior $p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f})$, where $\mathbf{f}_*$ are the latent predictions of test data $\mathbf{X}_*$.

The joint distribution of observed values $\mathbf{f}$ and predictions $\mathbf{f}_*$ is a Gaussian represented as

$$\begin{pmatrix} \mathbf{f} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} \mathbf{K} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right)$$

with N training data points and M test data points

$\mathbf{0}$ mean vector (e.g., by removing the means from the data)

$\mathbf{K} = \kappa(\mathbf{X}, \mathbf{X})$ is an $N \times \ N$ matrix

$\mathbf{K}_* = \kappa(\mathbf{X}, \mathbf{X}_*)$ is an $N \times M$ matrix

$\mathbf{K}_*^T = \kappa(\mathbf{X}_* \mathbf{X})$ is an $M \times N$ matrix

$\mathbf{K}_{**} = \kappa(\mathbf{X}_*, \mathbf{X}_*)$ is an $M \times M$ matrix

Gaussian Processes for Regression

$$\begin{pmatrix}\mathbf{f}\\ \mathbf{f}_*\end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix}\boldsymbol{\mu}\\ \boldsymbol{\mu}_*\end{pmatrix}, \begin{pmatrix}\mathbf{K} & \mathbf{K}_*\\ \mathbf{K}_*^{\mathrm{T}} & \mathbf{K}_{**}\end{pmatrix}\right)$$

Can be seen as:

$$\begin{pmatrix}\mathrm{f}_1\\ f_2\\ \dots\\ f_N\\ f_{*1}\\ f_{*2}\\ \dots\\ f_{*M}\end{pmatrix} \sim \mathcal{N}\left(\begin{pmatrix}\mu_1\\ \mu_2\\ \dots\\ \mu_N\\ \mu_{*1}\\ \mu_{*2}\\ \dots\\ \mu_{*M}\end{pmatrix}, \begin{matrix}\begin{pmatrix}\kappa(x,x)_{11} \dots \kappa(x,x)_{1N}\\ \kappa(x,x)_{21} \dots \kappa(x,x)_{2N}\\ \dots \quad \dots\\ \kappa(x,x)_{N1} \dots \kappa(x,x)_{NN}\end{pmatrix} & \begin{pmatrix}\kappa(x,x_*)_{11} \dots \kappa(x,x_*)_{1M}\\ \kappa(x,x_*)_{21} \dots \kappa(x,x_*)_{2M}\\ \dots \quad \dots\\ \kappa(x,x_*)_{N1} \dots \kappa(x,x_*)_{NM}\end{pmatrix}\\ \begin{pmatrix}\kappa(x_*,x)_{11} \dots \kappa(x_*,x)_{1N}\\ \kappa(x_*,x)_{21} \dots \kappa(x_*,x)_{2N}\\ \dots \quad \dots\\ \kappa(x_*,x)_{M1} \dots \kappa(x_*,x)_{MN}\end{pmatrix} & \begin{pmatrix}\kappa(x_*,x_*)_{11} \dots \kappa(x_*,x_*)_{1M}\\ \kappa(x_*,x_*)_{21} \dots \kappa(x_*,x_*)_{2M}\\ \dots \quad \dots\\ \kappa(x_*,x_*)_{M1} \dots \kappa(x_*,x_*)_{MM}\end{pmatrix}\end{matrix}\right)$$

Gaussian Processes for Regression

The predicted distribution (noise-free) is defined as

$$p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f}) = \mathcal{N}(\mathbf{f}_*|\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$$

$$\boldsymbol{\mu}_* = \mathbf{K}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{f} = \mathbf{K}^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{y}$$

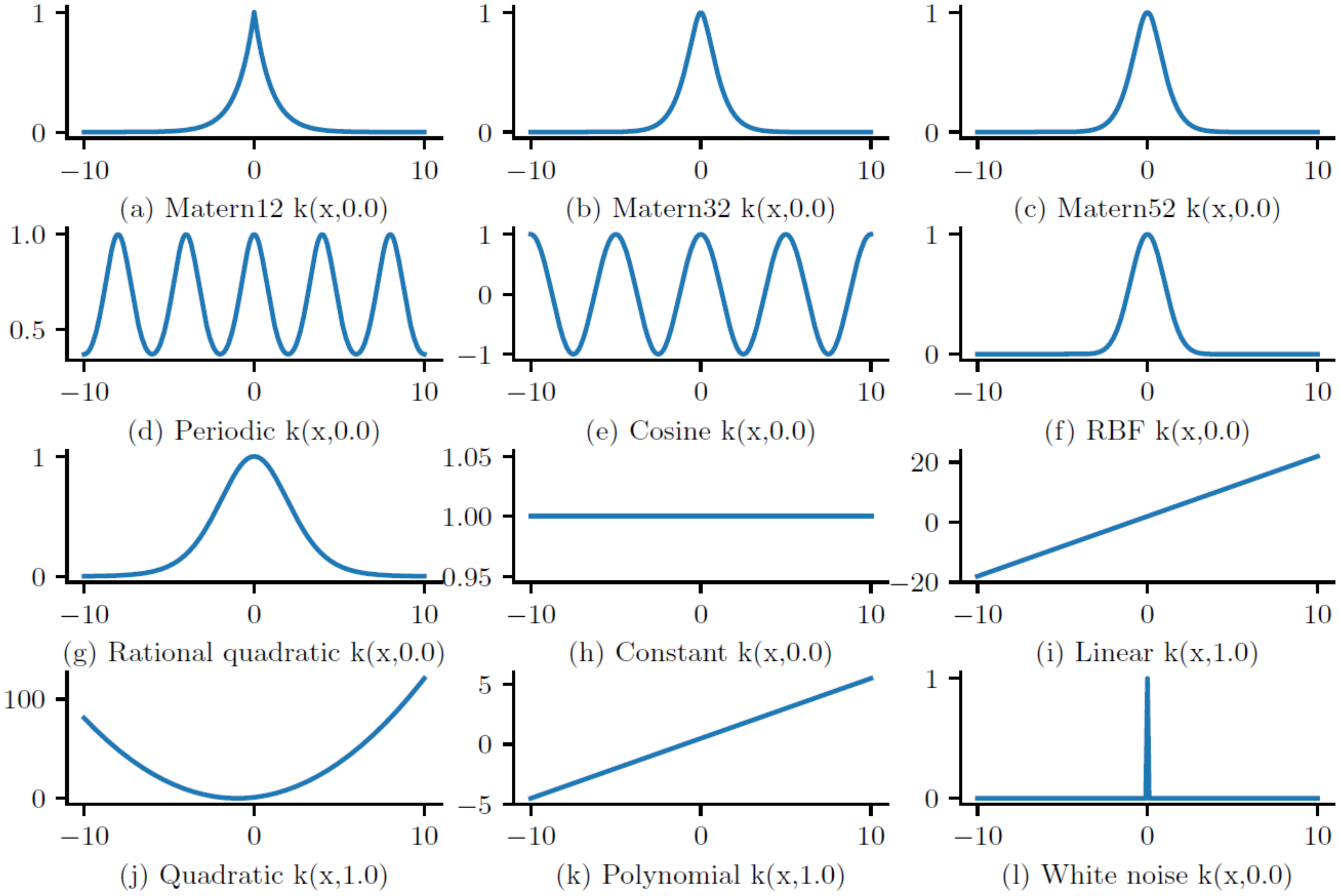
$$\boldsymbol{\Sigma}_* = \mathbf{K}_{**} - \mathbf{K}_*^{\mathrm{T}}\mathbf{K}^{-1}\mathbf{K}_*$$

By adding noise to the function values $\mathbf{y} = \mathbf{f} + \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, the term $\mathbf{K}^{-1}$ above is rewritten as $(\mathbf{K} + \sigma^2\mathbf{I})^{-1}$ and illustrated as

$$\begin{pmatrix} \mathbf{f} \\ \mathbf{f}_* \end{pmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{pmatrix} \mathbf{K} + \sigma^2\mathbf{I} & \mathbf{K}_* \\ \mathbf{K}_*^T & \mathbf{K}_{**} \end{pmatrix}\right)$$

where $\mathbf{I}$ is the identity matrix.

Gaussian Process Kernels



(a) Matern12 k(x,0.0)

(b) Matern32 k(x,0.0)

(c) Matern52 k(x,0.0)

(d) Periodic k(x,0.0)

(e) Cosine k(x,0.0)

(f) RBF k(x,0.0)

(g) Rational quadratic k(x,0.0)

(h) Constant k(x,0.0)

(i) Linear k(x,1.0)

(j) Quadratic k(x,1.0)

(k) Polynomial k(x,1.0)

(l) White noise k(x,0.0)

Pricture from K. Murphy, Probabilistic Machine Learning: Advanced Topics, Chapter 18, 2023.

Radial Basis Function (RBF) Kernel

$$\kappa(\mathbf{x}_\mathrm{i},\mathbf{x}_\mathrm{j}) = \boldsymbol{\sigma}_f^{\mathbf{2}} \exp\left(\frac{1}{2l^2}\left(\mathbf{x}_\mathrm{i}-\mathbf{x}_\mathrm{j}\right)^T\left(\mathbf{x}_\mathrm{i}-\mathbf{x}_\mathrm{j}\right)\right)$$

a.k.a. Gaussian kernel,
Squared Exponential kernel

Parameters:

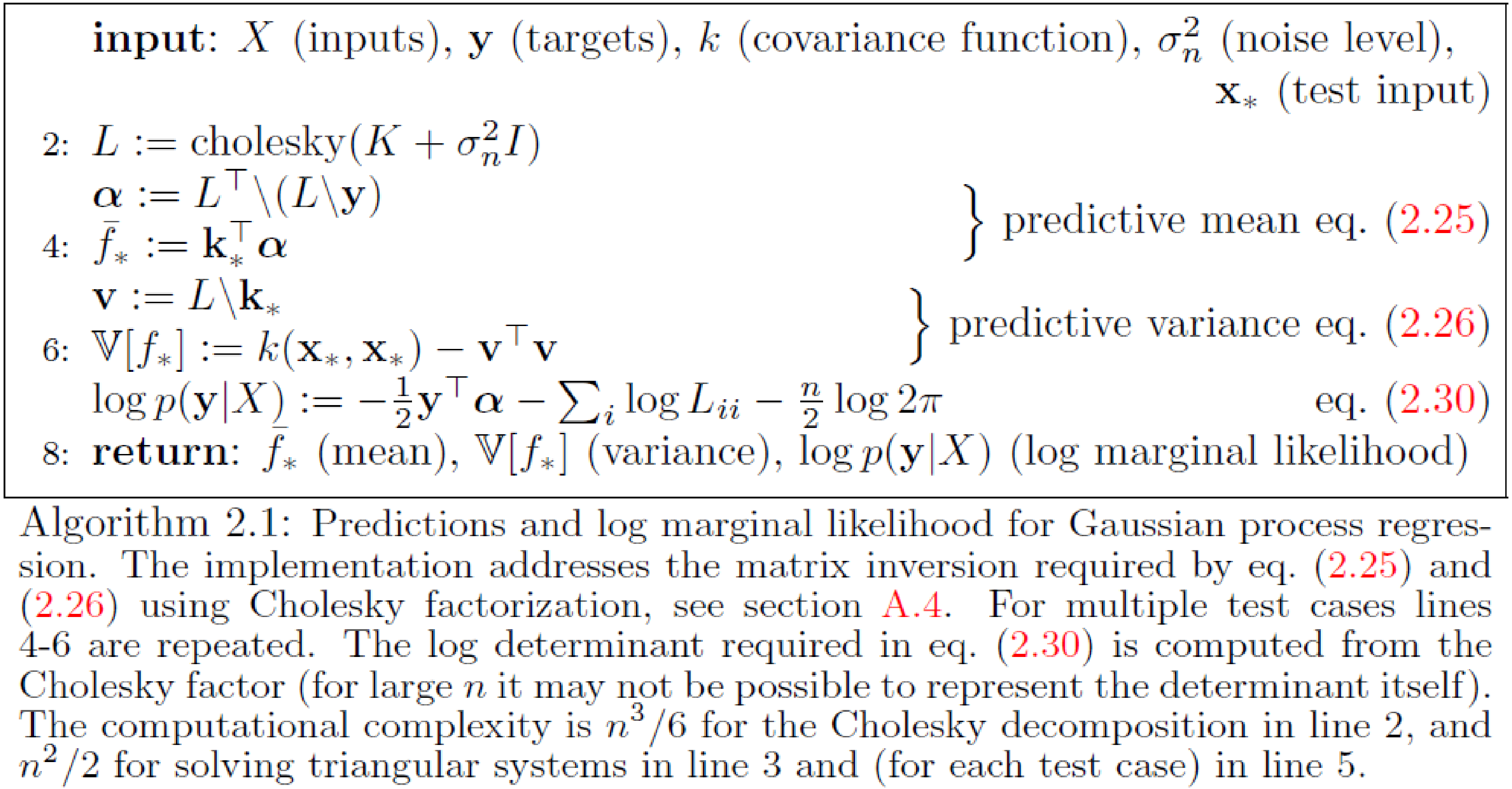
- σ_f controls the vertical variation
- l controls the smoothness of the function

Other kernels not only include Exponential, Matern32, Matern52, Periodic, Cosine, Brownian, Linear, but also their combinations.

A kernel is a positive definite matrix ($\mathbf{x}^\mathrm{T}\boldsymbol{Kx} > 0$):

```
numpy.all(np.linalg.eigvals(K)>0)
```

GPR: Algorithm



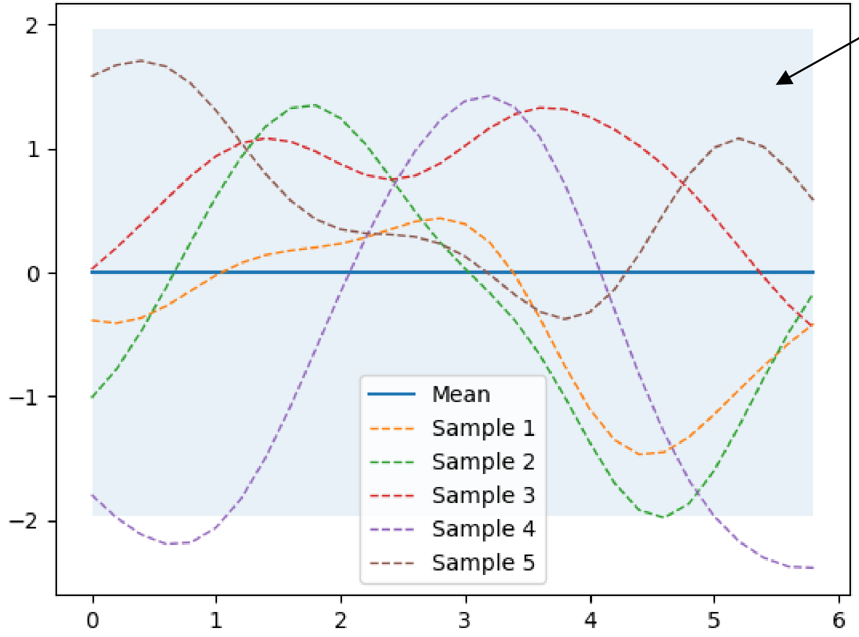
```
   input: $X$ (inputs), $\mathbf{y}$ (targets), $k$ (covariance function), $\sigma_n^2$ (noise level),
                                                                        $\mathbf{x}_*$ (test input)
2: $L := \text{cholesky}(K + \sigma_n^2 I)$
   $\boldsymbol{\alpha} := L^\top \backslash (L \backslash \mathbf{y})$                    $\}$ predictive mean eq. (2.25)
4: $\bar{f}_* := \mathbf{k}_*^\top \boldsymbol{\alpha}$
   $\mathbf{v} := L \backslash \mathbf{k}_*$                              $\}$ predictive variance eq. (2.26)
6: $\mathbb{V}[f_*] := k(\mathbf{x}_*, \mathbf{x}_*) - \mathbf{v}^\top \mathbf{v}$
   $\log p(\mathbf{y}|X) := -\frac{1}{2}\mathbf{y}^\top \boldsymbol{\alpha} - \sum_i \log L_{ii} - \frac{n}{2} \log 2\pi$        eq. (2.30)
8: return: $\bar{f}_*$ (mean), $\mathbb{V}[f_*]$ (variance), $\log p(\mathbf{y}|X)$ (log marginal likelihood)
```

Algorithm 2.1: Predictions and log marginal likelihood for Gaussian process regression. The implementation addresses the matrix inversion required by eq. (2.25) and (2.26) using Cholesky factorization, see section A.4. For multiple test cases lines 4-6 are repeated. The log determinant required in eq. (2.30) is computed from the Cholesky factor (for large n it may not be possible to represent the determinant itself). The computational complexity is $n^3/6$ for the Cholesky decomposition in line 2, and $n^2/2$ for solving triangular systems in line 3 and (for each test case) in line 5.

Image above from Rasmussen & Williams. Gaussian Processes for Machine Learning. MIT Press, 2006. Chapter 2

Sampling from a Prior

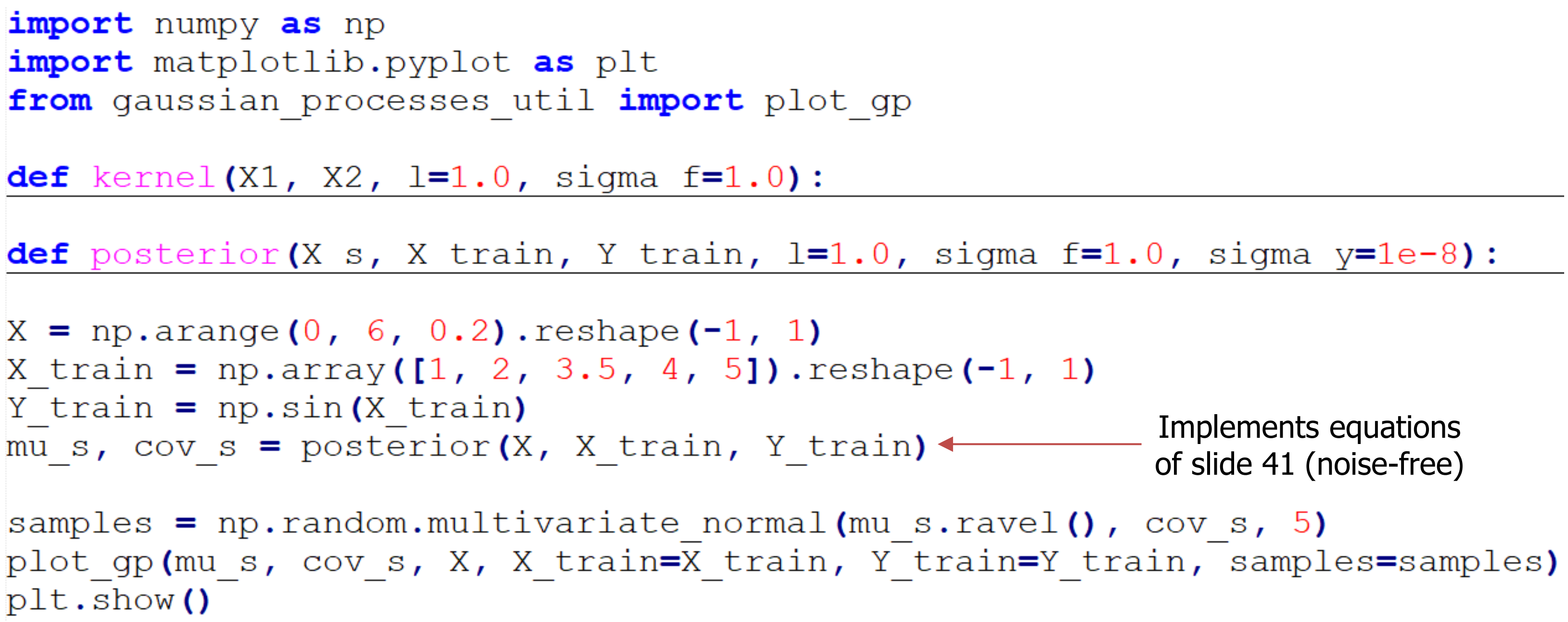
```
import numpy as np
import matplotlib.pyplot as plt
from gaussian_processes_util import plot_gp

def kernel(X1, X2, l=1.0, sigma_f=1.0):
    sqdist = np.sum(X1**2, 1).reshape(-1, 1) + np.sum(X2**2, 1) - 2 * np.dot(X1, X2.T)
    return sigma_f**2 * np.exp(-0.5 / l**2 * sqdist)

X = np.arange(0, 6, 0.2).reshape(-1, 1)
mu = np.zeros(X.shape)
cov = kernel(X, X)

samples = np.random.multivariate_normal(mu.ravel(), cov, 5)
plot_gp(mu, cov, X, samples=samples)
plt.show()
```



Uncertainty in blue (95% confidence interval)

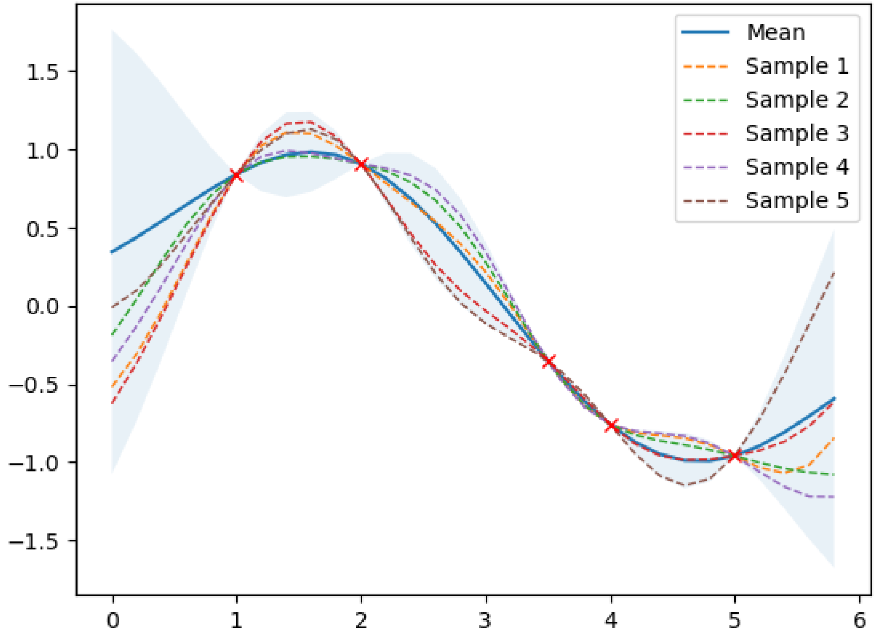
Generated from this source

Sampling from a Posterior



```
import numpy as np
import matplotlib.pyplot as plt
from gaussian_processes_util import plot_gp

def kernel(X1, X2, l=1.0, sigma f=1.0):

def posterior(X s, X train, Y train, l=1.0, sigma f=1.0, sigma y=1e-8):

X = np.arange(0, 6, 0.2).reshape(-1, 1)
X_train = np.array([1, 2, 3.5, 4, 5]).reshape(-1, 1)
Y_train = np.sin(X_train)
mu_s, cov_s = posterior(X, X_train, Y_train)

samples = np.random.multivariate_normal(mu_s.ravel(), cov_s, 5)
plot_gp(mu_s, cov_s, X, X_train=X_train, Y_train=Y_train, samples=samples)
plt.show()
```

Implements equations of slide 41 (noise-free)

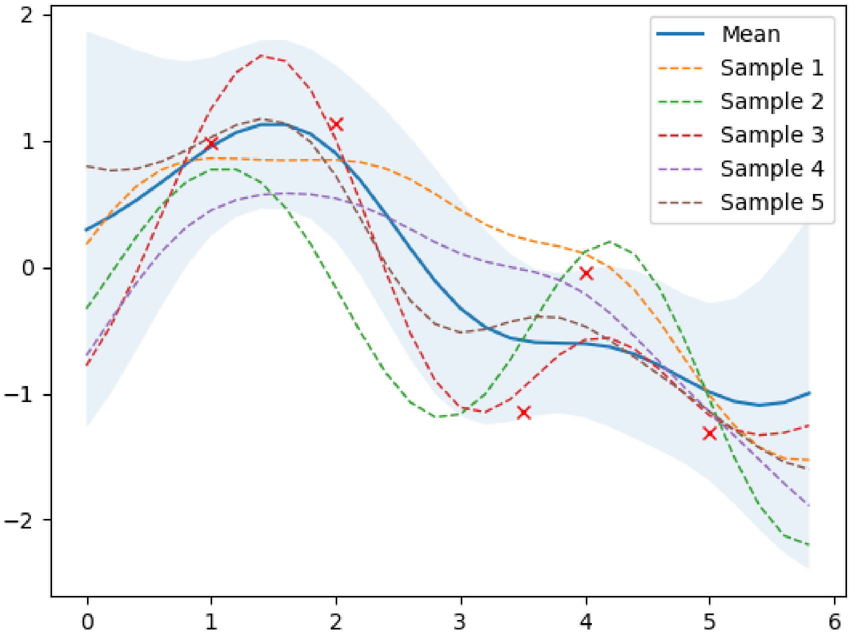


Generated from this source

Sampling from a Posterior

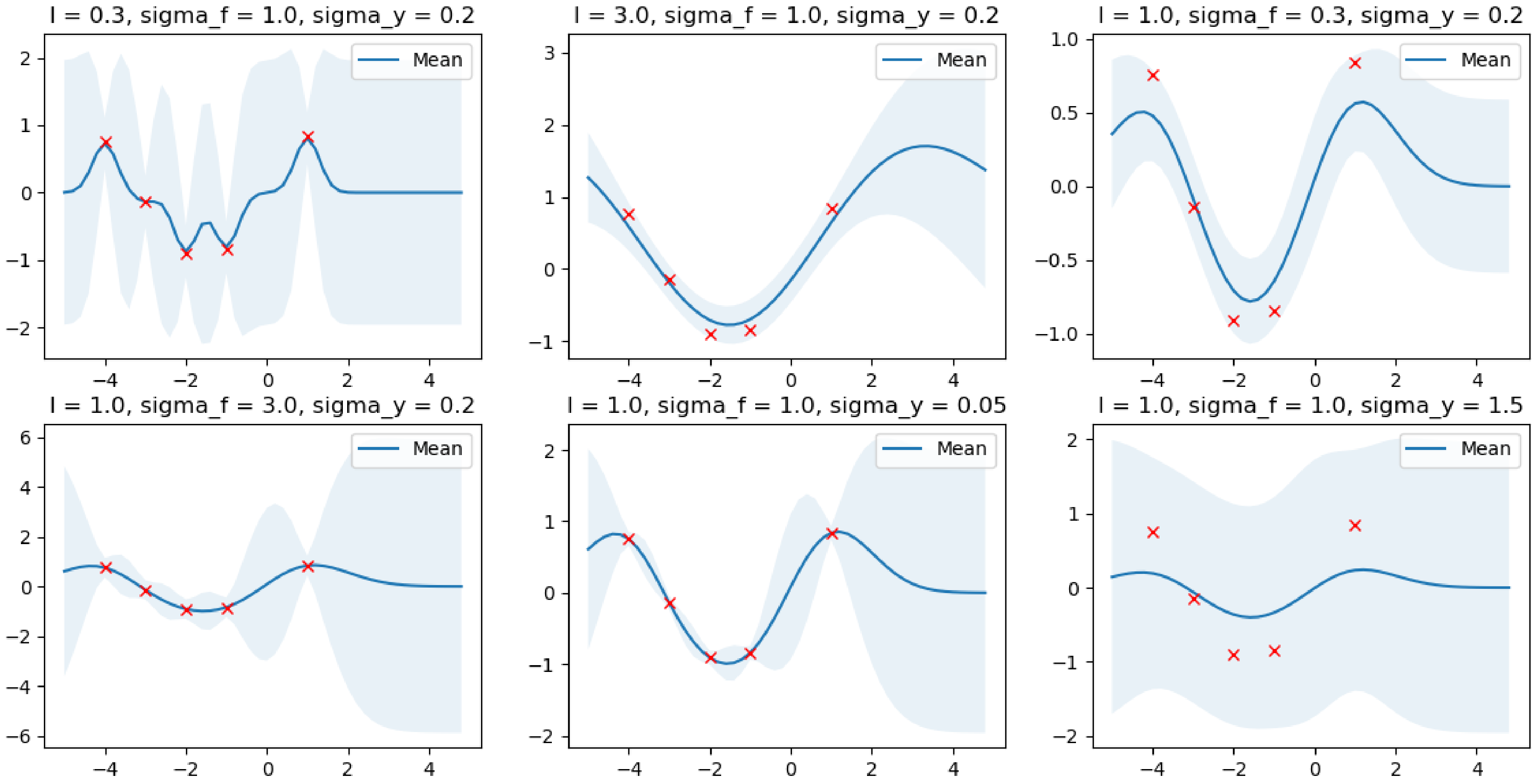
```
import numpy as np
import matplotlib.pyplot as plt
from gaussian_processes_util import plot_gp

def kernel(X1, X2, l=1.0, sigma_f=1.0):

def posterior(X_s, X_train, Y_train, l=1.0, sigma_f=1.0, sigma_y=1e-8):

noise = 0.4
X = np.arange(0, 6, 0.2).reshape(-1, 1)
X_train = np.array([1, 2, 3.5, 4, 5]).reshape(-1, 1)
Y_train = np.sin(X_train) + noise * np.random.randn(*X_train.shape)
mu_s, cov_s = posterior(X, X_train, Y_train, sigma_y=noise)

samples = np.random.multivariate_normal(mu_s.ravel(), cov_s, 5)
plot_gp(mu_s, cov_s, X, X_train=X_train, Y_train=Y_train, samples=samples)
plt.show()
```

Noise level to distort the targets
… and increase uncertainty



Generated
from this
source

Effects of Kernel and Noise Parameters



Generated from this source

Kernel parameters σ_f and l. Noise parameter σ_y.

Optimisation of Kernel Parameters

Optimal values for the kernel parameters can be estimated by maximising the log marginal likelihood

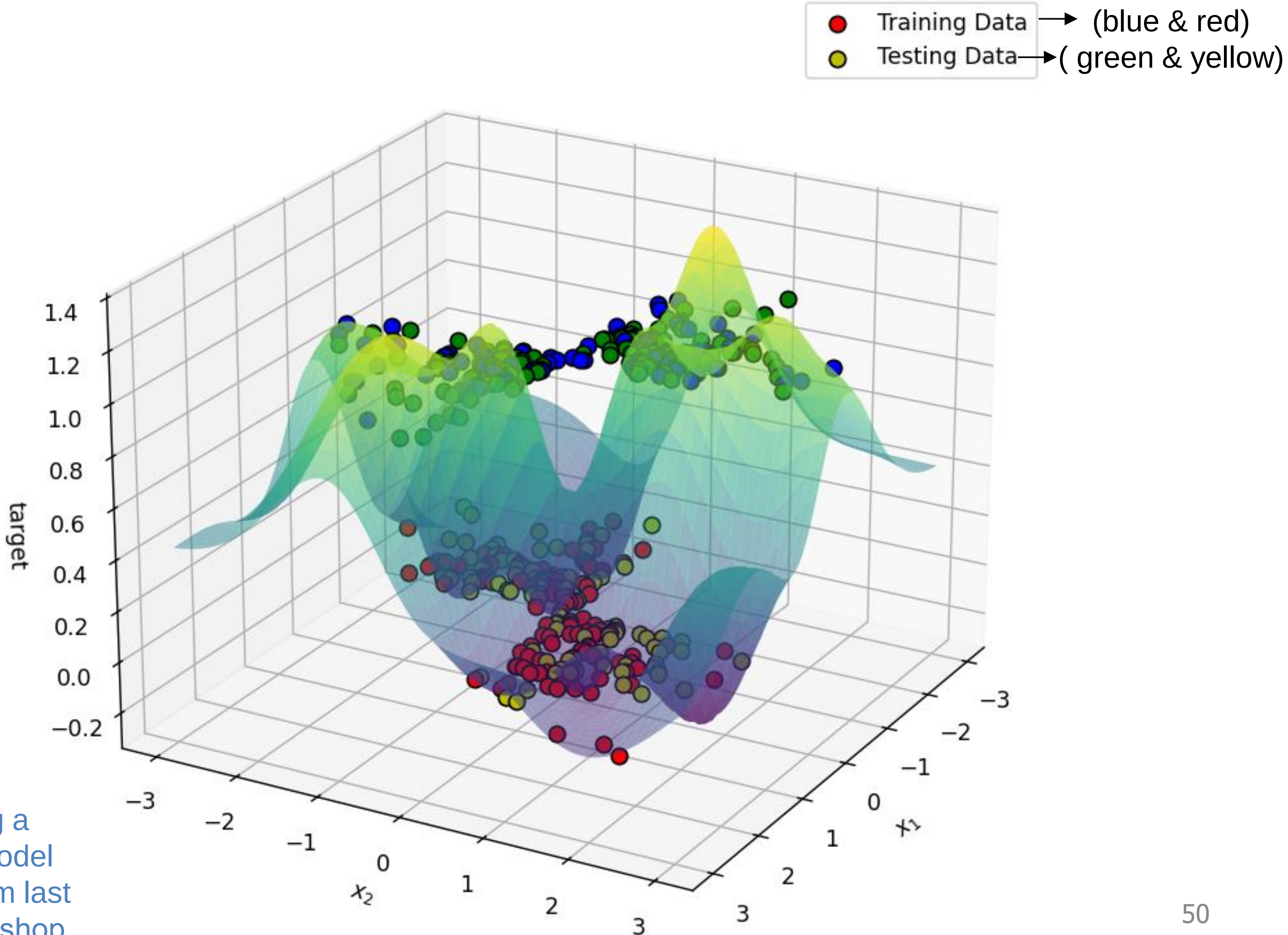
$$\theta^* = \arg\max_{\theta} \log p(\mathbf{y}|\mathbf{X}, \theta)$$

where

$$\begin{aligned} \log p(\mathbf{y}|\mathbf{X}, \theta) &= \log \mathcal{N}(\boldsymbol{y}|\mathbf{0}, \boldsymbol{K}_y) \\ &= -\frac{1}{2}\boldsymbol{y}^T \boldsymbol{K}_y^{-1} \boldsymbol{y} - \frac{1}{2}\log|\boldsymbol{K}_y| - \frac{N}{2}\log 2\pi \end{aligned}$$

In case of the RBF kernel, the parameters to optimise are l (smoothness of function) and σ_f (vertical variation). Methods to address that include gradient ascent/descent, L-BFGS, etc.

GPR on Non-Linearly Separable Data



Plot using a gpytorch model and data from last week's workshop

Today

- Preliminaries
  - Overview of linear algebra
  - Multivariate Gaussian distributions
  - The kernel trick revisited
- Gaussian processes for regression (GPR)
- **Gaussian processes for classification (GPC)**

With a focus on binary classification

How to do classification with GPs?

BASELINE: Treat the task as regression instead of classification, then normalise predicted outputs to produce values $[0 \ldots 1]$

This implies training a GPR (as done earlier):

$$p(\mathbf{f}_*|\mathbf{X}_*, \mathbf{X}, \mathbf{f}) = \mathcal{N}(\mathbf{f}_*|\boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*)$$

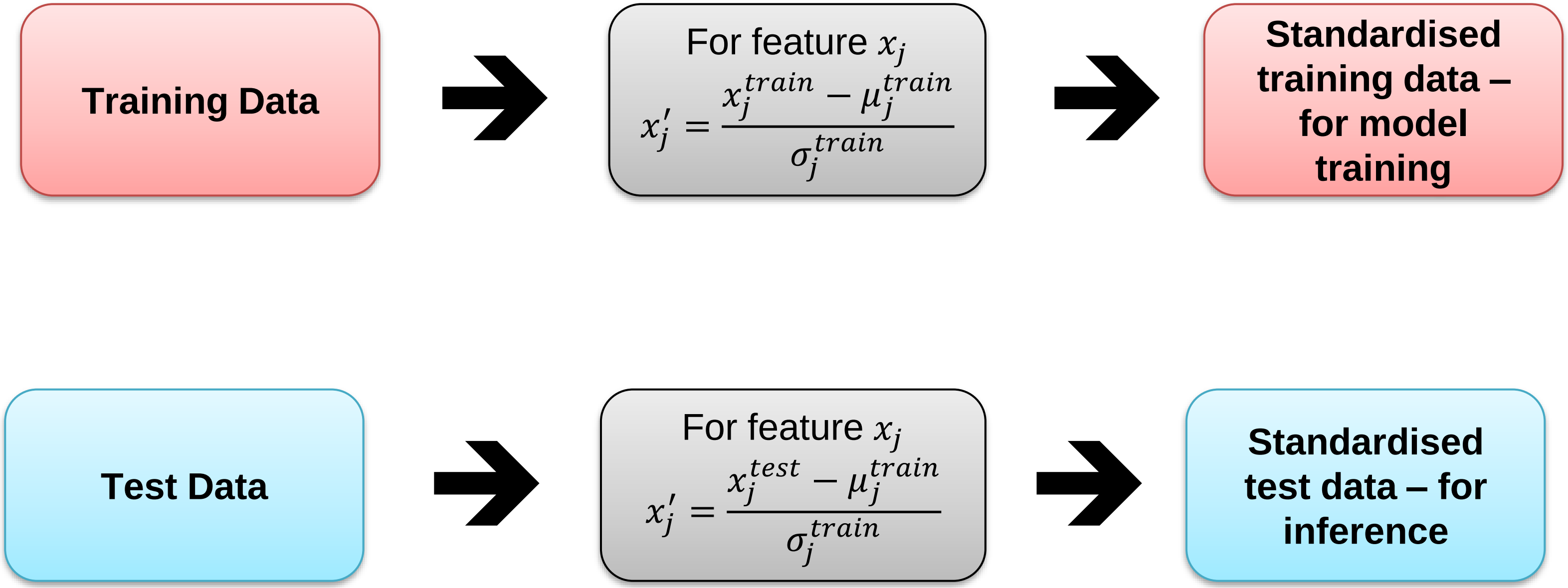
Use the learnt mean vector $\boldsymbol{\mu}_*$ and covariance matrix $\boldsymbol{\Sigma}_*$ to generate predicted outputs for test input x:

$$P(y_* = 1|\mathbf{X}, \mathbf{y}, \mathrm{x}_*) \approx \frac{\mathcal{N}(1;\ \mu_{x_*}, \sigma_{x_*})}{\mathcal{N}\left(1;\ \mu_{x_*}, \sigma_{x_*}\right) + \mathcal{N}(0;\ \mu_{x_*}, \sigma_{x_*})}$$

$$P(y_* = 0|\mathbf{X}, \mathbf{y}, \mathrm{x}_*) = 1 - P(\boldsymbol{y}_* = 1|\mathbf{X}, \mathbf{y}, \mathbf{x}_*)$$

Do we Need Data Standardisation?

Not for discrete Bayes Nets or when data is already in a similar scale, but consider it for regression-based models – including Gaussian Processes.



Training Data

For feature x_j

$$x_j' = \frac{x_j^{train} - \mu_j^{train}}{\sigma_j^{train}}$$

Standardised training data – for model training

Test Data

For feature x_j

$$x_j' = \frac{x_j^{test} - \mu_j^{train}}{\sigma_j^{train}}$$

Standardised test data – for inference

Gaussian Processes for Classification

Consider dataset $\{(\boldsymbol{x}_i, y_i) | i = 1, \dots, n\}$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ denotes data point i and $y_i \in \{0,1\}$ denotes its corresponding target.

Inference in a GP for binary classification follows two steps:

1. Compute the following distribution for the test cases $\boldsymbol{x}_*$

$$p(\boldsymbol{f}_* | \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*) = \int p(\boldsymbol{f}_* | \boldsymbol{X}, \boldsymbol{x}_*, \boldsymbol{f}) p(\boldsymbol{f} | \boldsymbol{X}, \boldsymbol{y}) \, d\boldsymbol{f}_*$$

2. Use that distribution to produce probabilistic predictions

$$P(\boldsymbol{y}_* = 1 | \mathbf{X}, \mathbf{y}, \mathbf{x}_*) = \int \sigma(\boldsymbol{f}_*) p(\boldsymbol{f}_* | \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}_*) \, d\boldsymbol{f}_*$$

Gaussian Processes for Classification

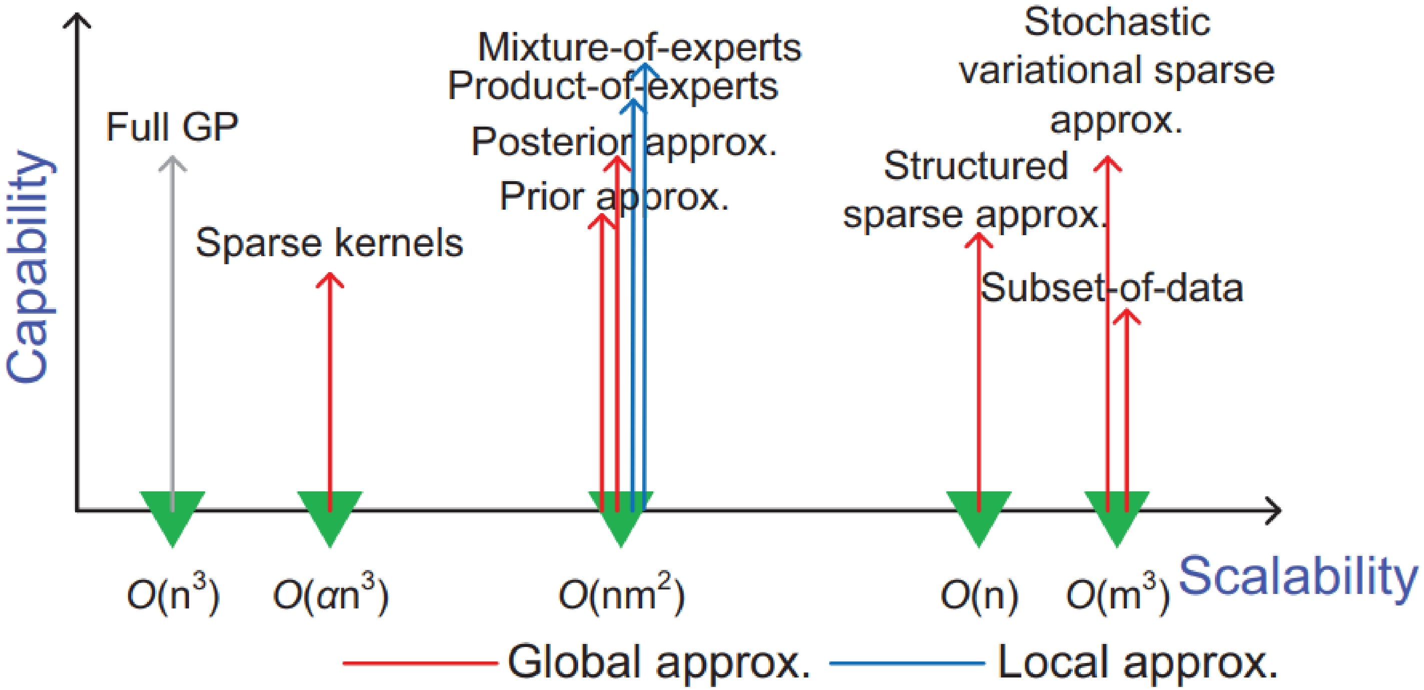
Solutions to $p(\boldsymbol{f}|\boldsymbol{X}, \boldsymbol{y})$ can be approximated via methods such as the following:

- *Laplace Approximation*
- *Expectation Propagation*
- *Variational Inference*

The above are not covered in this module due to extensive implementation details, high computational cost, and not clearly better than the baseline above.

For details, see C.E. Rasmussen & C.K.I. Williams. Gaussian Processes for Machine Learning. MIT Press, 2006 – chapter 3.

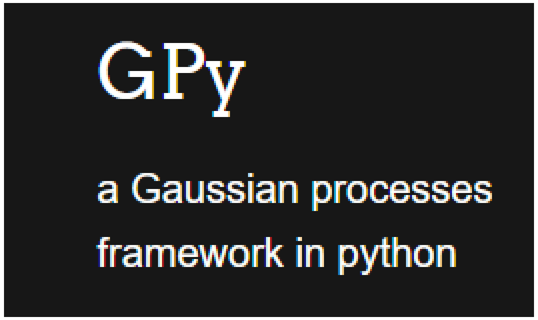
GPs: Computational Considerations



Sparse Gaussian Processes summarise data by m instead of n data points, which reduce time complexity from $O(n^3)$ to $O(nm^2)$.

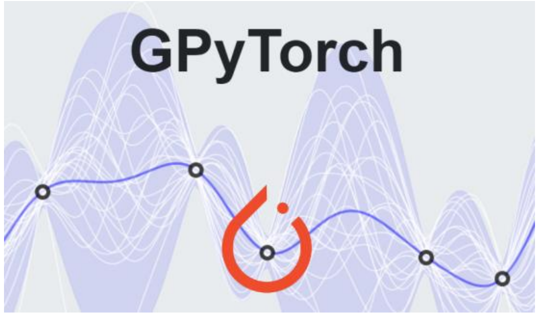
Image above from Liu et al, When Gaussian Process Meets Big Data: A Review of Scalable GPs. IEEE TNNLS, 2020

Other Implementions for GPR/GPC



GPy

a Gaussian processes framework in python



GPflow

Build Gaussian process models in python

scikit-learn

Machine Learning in Python

Workshop materials of this week mostly focused on GPyTorch

Ideas for your Assignment (optional)

1. Apply Gaussian Processes to continuous data (look for ***gpytorch_GPR.py*** in the workshop materials).

2. Compare and analise Gaussian Bayesian networks (or any of the previously discussed methods) vs. Gaussian Processes.

3. Compare Gaussian Processes using different kernel functions. Only the RBF kernel is included in the workshop materials.

Today

- Preliminaries of multivariate Gaussians
- Gaussian processes for regression
- Gaussian processes for binary classification

Readings:

- J. Wang. An intuitive Tutorial to Gaussian Process Regression. IEEE, Computing in Science and Engineering, 2024.

- C.E. Rasmussen & C.K.I. Williams. Gaussian Processes for Machine Learning. MIT Press, 2006. Chapter 2 (GPR), 3 (GPC)

- K. Murphy. Probabilistic Machine Learning: Advanced Topics, MIT Press, 2023. Chapter 18.

This and Next Week

Workshop (today):
Exercises related to Gaussian conditional probabilities, code for Gaussian Processes, and assignment support.

Lecture (next week):
Probabilistic reasoning over time
(with Riccardo Polvara)

Questions?