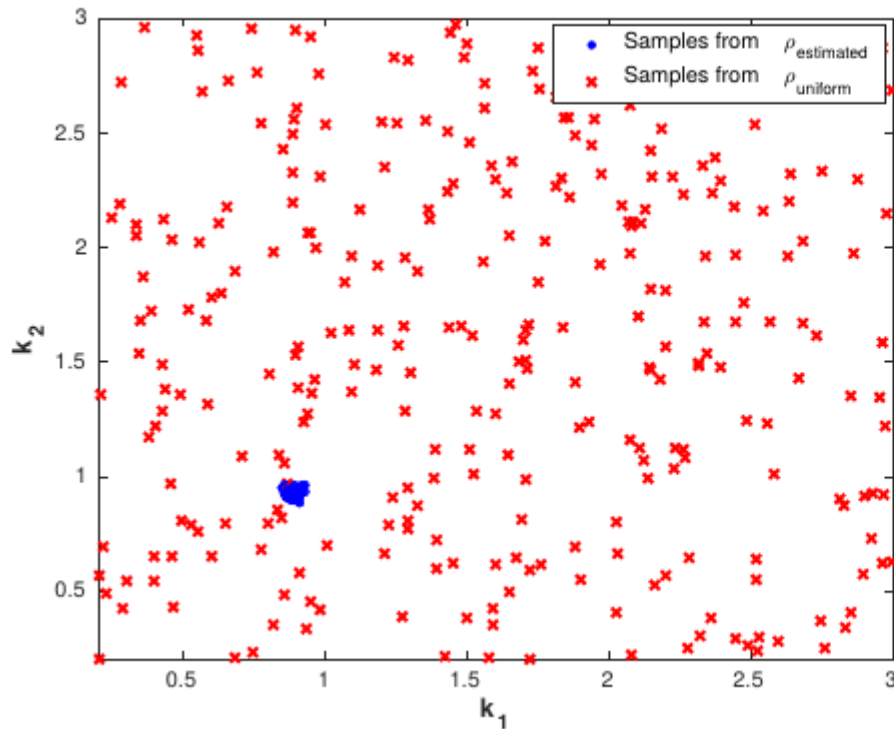


# Gaussian Process Regression & Gaussian Mixture Models

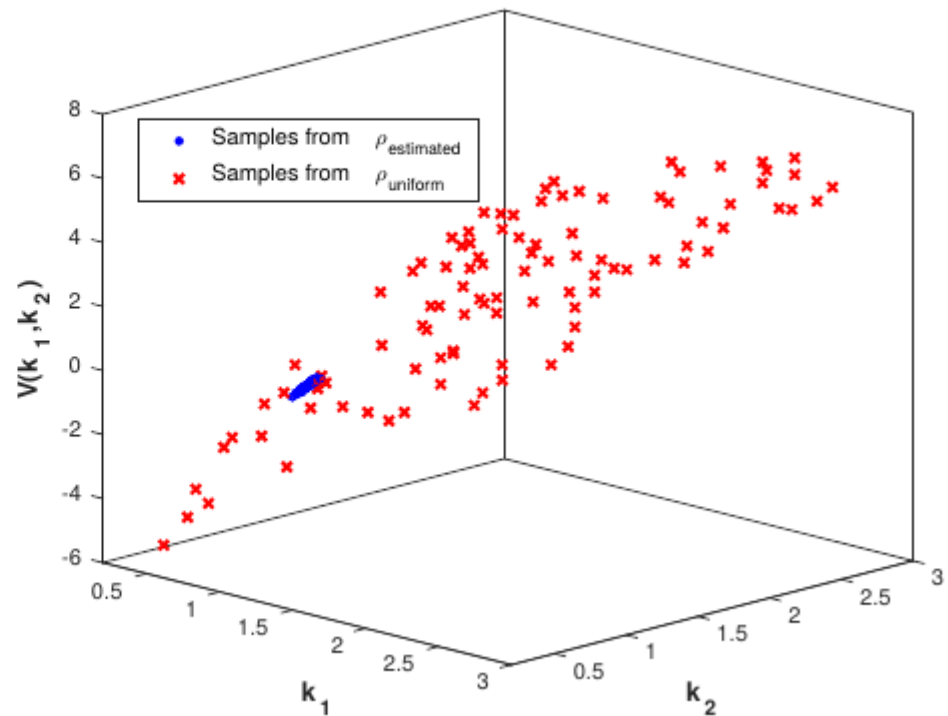
Simon Scheidegger  
simon.scheidegger@unil.ch  
January 23<sup>th</sup>, 2019

Cowles Foundation – Yale University

# Recall: Want to solve Dynamic Models on high-dimensional, irregularly-shaped state-spaces



Blue: ergodic set.  
Red: Computational domain



Blue: Value function, evaluated on ergodic set  
Red: Value function, evaluated on the entire comp. domain

# Today's Roadmap

## I. Introduction to Gaussian Process Regression (II)

- Predictions using noisy observations
- Estimating the Hyper-parameters
- More on Kernels

## II. Gaussian Mixture Models

- The basic idea
- The Expectation Maximization Algorithm (EM)

# Recall – Predictions using noise-free observations

Suppose we observe a training set

$$D = \{(x_i, f_i), i = 1 : N\}$$

where  $f_i = f(x_i)$  is the **noise-free observation** of the function evaluated at  $x_i$ .

Given a **test set  $X_*$**  of **size  $N_* \times D$** , we want to **predict the function outputs  $f_*$** .

If we ask the GP to **predict  $f(x)$  for a value of  $x$  that it has already seen**, we want the GP to return the **answer  $f(x)$  with no uncertainty**.

→ In other words, it should act as an interpolator of the training data. This will only happen if we **assume the observations are noiseless**.

# Recall: Joint Gaussian distributions

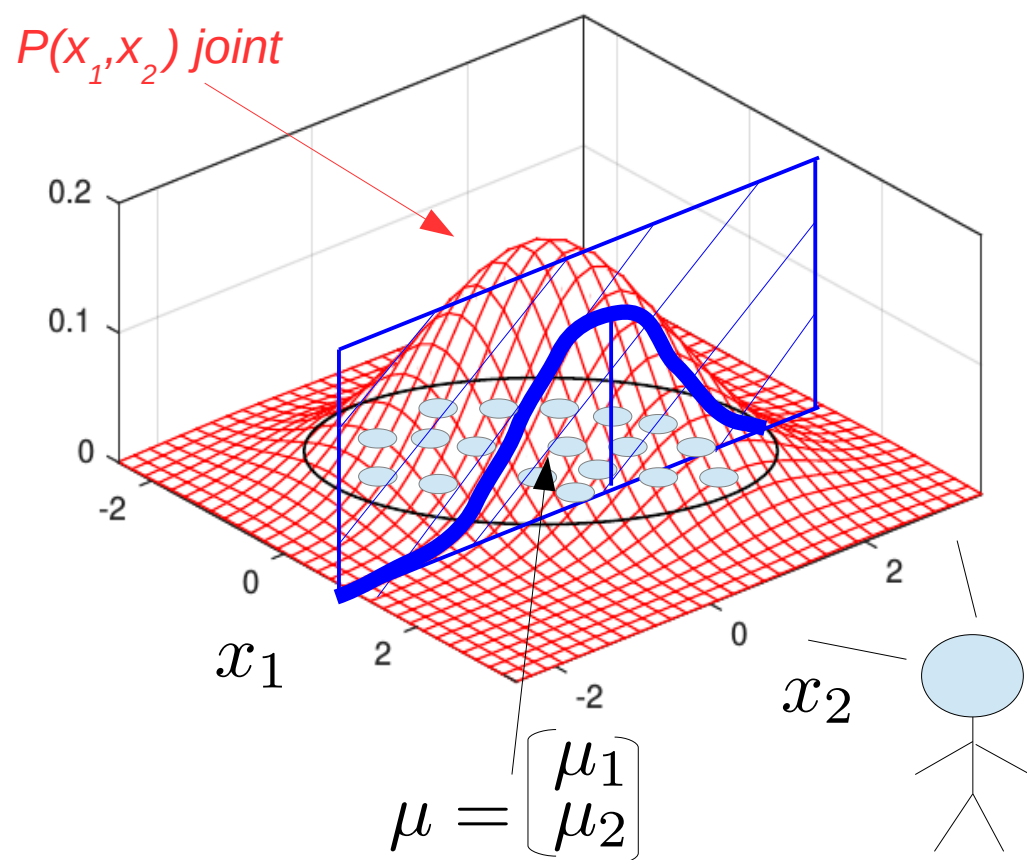
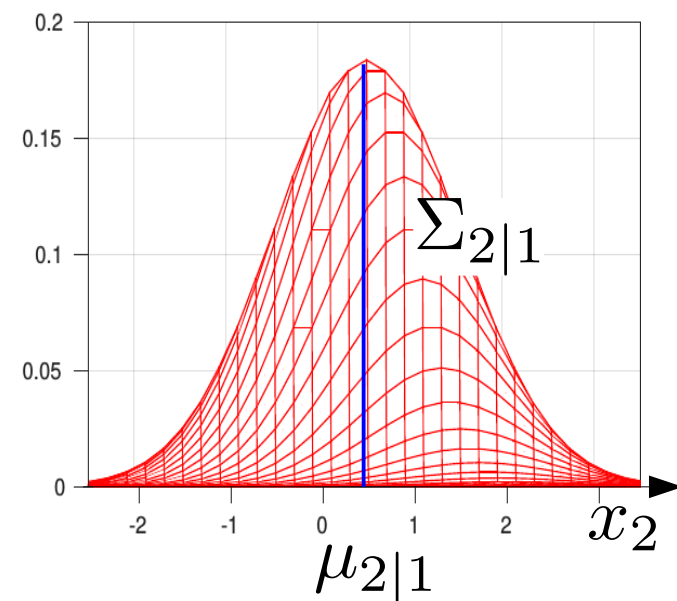
see, e.g., Rasmussen et al. (2005), Murphy (2012)

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \sim N \left( \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix}, \begin{bmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{bmatrix} \right)$$

Mean
Covariance

Conditional distribution

$$P(x_2 | X_1 = x_1)$$



# Recall: From Joint to Conditional distributions

see, e.g., Murphy (2012), chapter 4.

**Theorem 4.3.1** (Marginals and conditionals of an MVN). Suppose  $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2)$  is jointly Gaussian with parameters

$$\boldsymbol{\mu} = \begin{pmatrix} \boldsymbol{\mu}_1 \\ \boldsymbol{\mu}_2 \end{pmatrix}, \quad \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{11} & \boldsymbol{\Sigma}_{12} \\ \boldsymbol{\Sigma}_{21} & \boldsymbol{\Sigma}_{22} \end{pmatrix}, \quad \boldsymbol{\Lambda} = \boldsymbol{\Sigma}^{-1} = \begin{pmatrix} \boldsymbol{\Lambda}_{11} & \boldsymbol{\Lambda}_{12} \\ \boldsymbol{\Lambda}_{21} & \boldsymbol{\Lambda}_{22} \end{pmatrix} \quad (4.67)$$

Then the marginals are given by

$$\begin{aligned} p(\mathbf{x}_1) &= \mathcal{N}(\mathbf{x}_1 | \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_{11}) \\ p(\mathbf{x}_2) &= \mathcal{N}(\mathbf{x}_2 | \boldsymbol{\mu}_2, \boldsymbol{\Sigma}_{22}) \end{aligned} \quad (4.68)$$

and the posterior conditional is given by

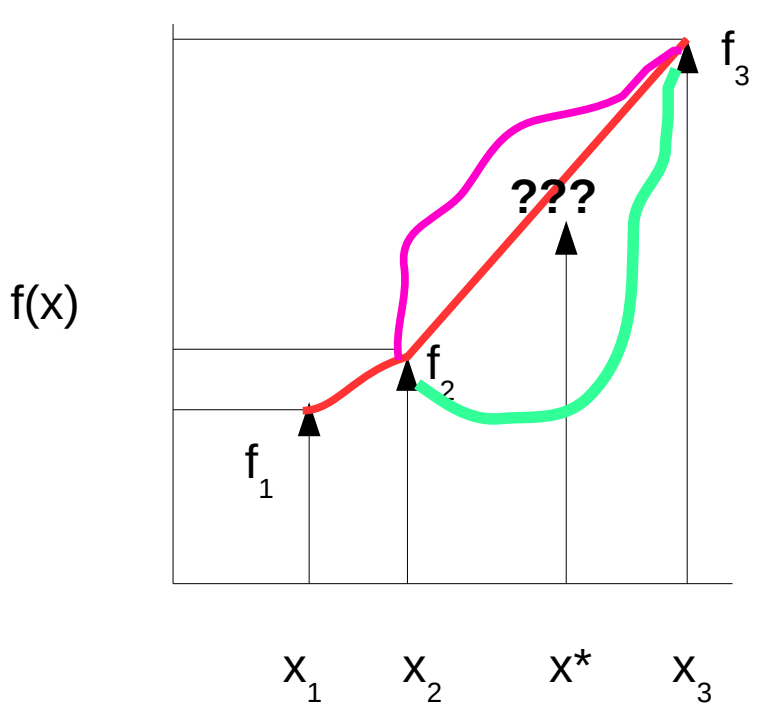
$$\begin{aligned} p(\mathbf{x}_1 | \mathbf{x}_2) &= \mathcal{N}(\mathbf{x}_1 | \boldsymbol{\mu}_{1|2}, \boldsymbol{\Sigma}_{1|2}) \\ \boldsymbol{\mu}_{1|2} &= \boldsymbol{\mu}_1 + \boldsymbol{\Sigma}_{12} \boldsymbol{\Sigma}_{22}^{-1} (\mathbf{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{11}^{-1} \boldsymbol{\Lambda}_{12} (\mathbf{x}_2 - \boldsymbol{\mu}_2) \\ &= \boldsymbol{\Sigma}_{1|2} (\boldsymbol{\Lambda}_{11} \boldsymbol{\mu}_1 - \boldsymbol{\Lambda}_{12} (\mathbf{x}_2 - \boldsymbol{\mu}_2)) \\ \boldsymbol{\Sigma}_{1|2} &= \boldsymbol{\Sigma}_{11} - \boldsymbol{\Sigma}_{12} \boldsymbol{\Sigma}_{22}^{-1} \boldsymbol{\Sigma}_{21} = \boldsymbol{\Lambda}_{11}^{-1} \end{aligned} \quad (4.69)$$

Two “blocks” of vectors

This Theorem allows you to go from joint to conditional distributions.

# Observations → Interpolation (II)

We assume that **f's (the height)** are **Gaussian distributed**, with **zero – mean** and some **covariance matrix K**.



$$\begin{pmatrix} f_1 \\ f_2 \\ f_3 \end{pmatrix} \sim N \left( \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{11} & K_{12} & K_{13} \\ K_{21} & K_{22} & K_{23} \\ K_{31} & K_{32} & K_{33} \end{bmatrix} \right)$$

Note:  $f_1$  and  $f_2$  should probably be more correlated, as they are nearby (compared to  $f_1$  and  $f_3$ ), e.g.,

$$\sim N \left( \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0.7 & 0.2 \\ 0.7 & 1 & 0.6 \\ 0.2 & 0.6 & 1 \end{bmatrix} \right)$$

Covariance matrix **constructed** by some “**measure of similarity**”, i.e., a **kernel function** (parametric ansatz), such as “squared exponential”. Parameters can be obtained e.g. via MLE (later).

$$\kappa(x, x') = \sigma_f^2 \exp\left(-\frac{1}{2\ell^2}(x - x')^2\right)$$

$\sigma_f^2$  – controls vertical variation.

$\ell$  – controls horizontal length scale.

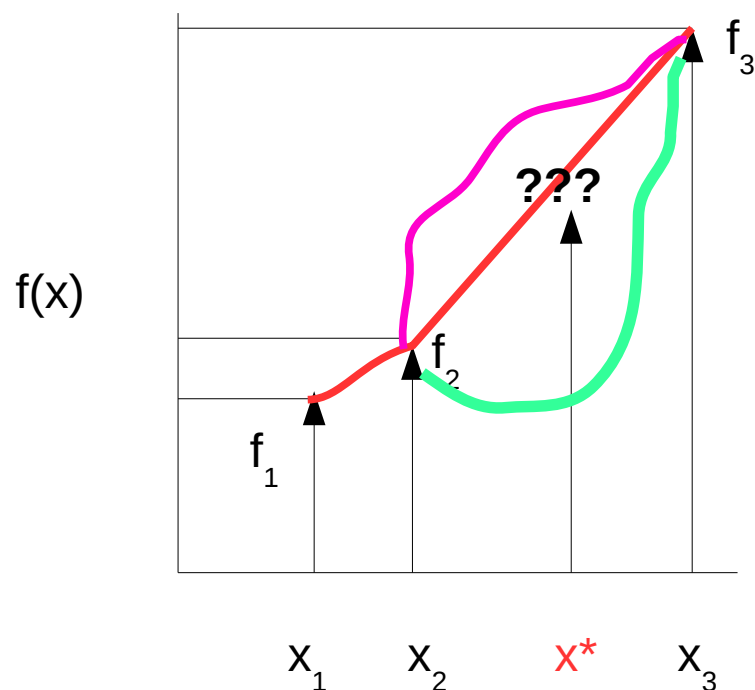
# Remember: Observations $\rightarrow$ Interpolation

Given data  $D = \{ (x_1, f_1), (x_2, f_2), (x_3, f_3) \} \rightarrow f(x^*) = f_*$  ?

$\rightarrow$  Assume  $f \sim N(0, K(\cdot, \cdot))$

$\rightarrow$  Assume  $f(x^*) \sim N(0, K(x^*, x^*))$

3d-Covariance  $K$  from  
the training data



$$\begin{pmatrix} f_1 \\ f_2 \\ f_3 \\ f_* \end{pmatrix} \sim N \left( \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K_{11} & K_{12} & K_{13} & K_{1*} \\ K_{21} & K_{22} & K_{23} & K_{2*} \\ K_{31} & K_{32} & K_{33} & K_{3*} \\ K_{*1} & K_{*2} & K_{*3} & K_{**} \end{bmatrix} \right)$$

$\rightarrow$  Joint distribution over  $f$  and  $f_*$ .

$\rightarrow$  We need the conditional of  $f_*$  given  $f$ .

$\rightarrow$  In this example, we “cut” in 3 dimensions.

$\rightarrow$  What is left is a 1-dimensional Gaussian, i.e., the Gaussian for  $f_*$

$$K(x_1, x_*) = K_{1*}$$



# Recall – Predictions using noise-free observations (II)

By definition of the GP, **the joint distribution** has the following form:

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

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

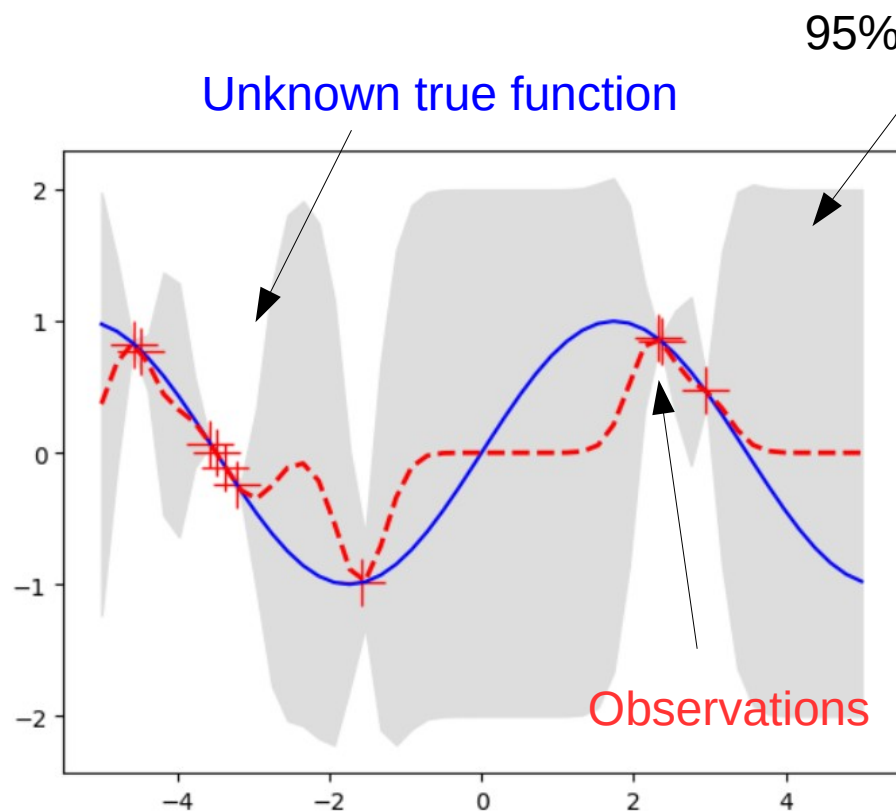
and where  $\mathbf{K}_{**} = \kappa(\mathbf{X}_*, \mathbf{X}_*)$  is  $N_* \times N_*$

By the standard rules for conditioning Gaussians, **the posterior** has the following form

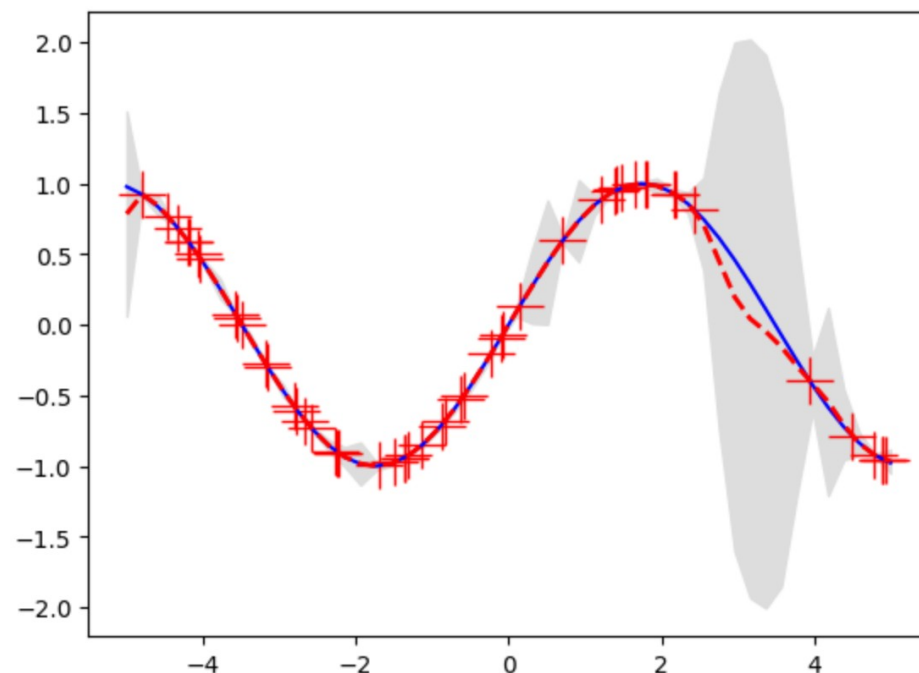
$$\begin{aligned} p(\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{f}) &= \mathcal{N}(\mathbf{f}_* | \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \\ \boldsymbol{\mu}_* &= \boldsymbol{\mu}(\mathbf{X}_*) + \mathbf{K}_*^T \mathbf{K}^{-1} (\mathbf{f} - \boldsymbol{\mu}(\mathbf{X})) \\ \boldsymbol{\Sigma}_* &= \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}^{-1} \mathbf{K}_* \end{aligned}$$

# Illustration of noiseless GPR prediction

Note: if you don't fix the seed, these pictures vary every time you run of the code.



**10 Training Points**



**50 Training Points**

We see that the model perfectly interpolates the training data, and that the predictive uncertainty increases as we move further away from the observed data.

## Illustration of noiseless GPR prediction (II)

- We use a squared exponential kernel, aka **Gaussian kernel or RBF kernel**.
- In  $1d$ , this is given by  $\kappa(x, x') = \sigma_f^2 \exp(-\frac{1}{2\ell^2}(x - x')^2)$
- Here  **$\ell$  controls the horizontal length scale** over which the function varies, and  **$\sigma_f$  controls the vertical variation**.
- On the right we show predictions from the posterior,  $p(f_* | X_*, X, f)$ .
- **We see that the model perfectly interpolates the training data, and that the predictive uncertainty increases as we move further away from the observed data.**

# The parameters in the Kernel

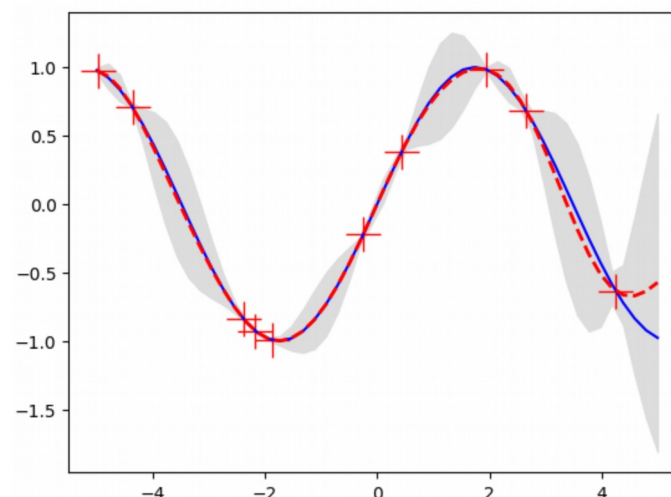
cf. `global_solution_yale19/Lecture_5/code/1d_gp_example.py`

$$\kappa(x, x') = \sigma_f^2 \exp\left(-\frac{1}{2\ell^2}(x - x')^2\right)$$

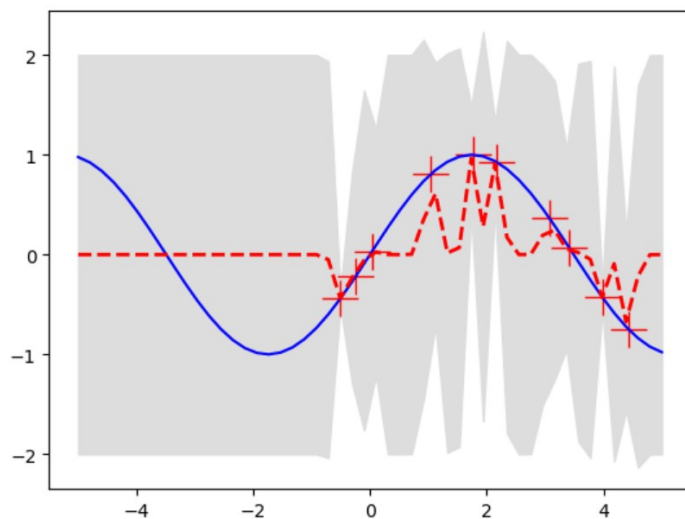
Let  $\sigma_f^2 = 1$

→ **Tuning the parameters by hand is not a good idea in general (in particular in high-dimensional settings).**

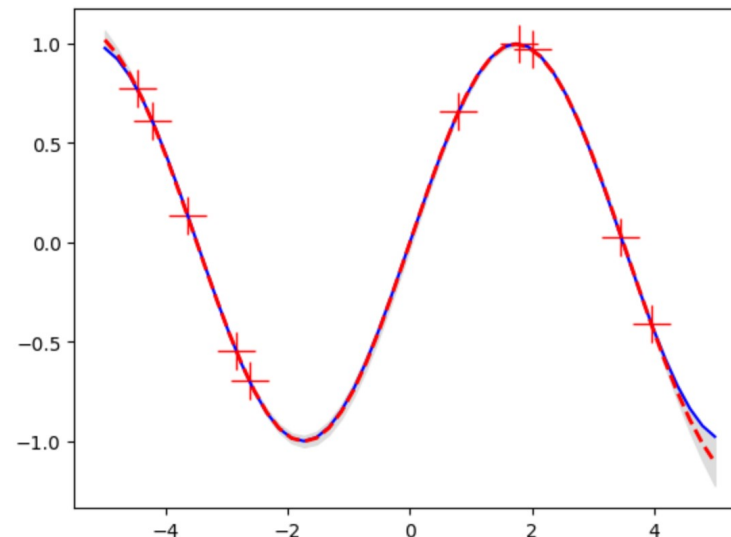
$\ell^2 = 1.0$



$\ell^2 = 0.01$



$\ell^2 = 10.0$



# GPR with noisy data

- In empirical setups, **measurement noise may arise** from our **inability to control all the influential factors** or from **irreducible (aleatory) uncertainties**.
- In **computer simulations**, measurement uncertainty may stem from quasi-random stochasticity, or chaotic behavior.
- Now let us consider the case where what we observe is a noisy version of the underlying function

$$y = f(x) + \epsilon, \text{ where } \epsilon \sim \mathcal{N}(0, \sigma_y^2)$$

## GPR with noisy data (II)

- In this case (presence of noise), the model is not required to interpolate the data, **but it must come “close”** to the observed data.
- The covariance of the observed noisy responses is

$$\text{cov}[y_p, y_q] = \kappa(\mathbf{x}_p, \mathbf{x}_q) + \sigma_y^2 \delta_{pq}$$

where  $\delta_{pq} = \mathbb{I}(p = q)$

- The second matrix is **diagonal** because we assumed the **noise terms were independently added to each observation**.

# The GPR with noisy data (III)

- The joint density of the observed data and the latent, noise-free function on the test points is given by

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

Latent function. Noise in the diagonal, rest is as before.

- where we are assuming the mean is zero, for notational simplicity.
- Hence the posterior predictive density is

$$\begin{aligned} p(\mathbf{f}_* | \mathbf{X}_*, \mathbf{X}, \mathbf{y}) &= \mathcal{N}(\mathbf{f}_* | \boldsymbol{\mu}_*, \boldsymbol{\Sigma}_*) \\ \boldsymbol{\mu}_* &= \mathbf{K}_*^T \mathbf{K}_y^{-1} \mathbf{y} \\ \boldsymbol{\Sigma}_* &= \mathbf{K}_{**} - \mathbf{K}_*^T \mathbf{K}_y^{-1} \mathbf{K}_* \end{aligned}$$

# Prediction at a single test point

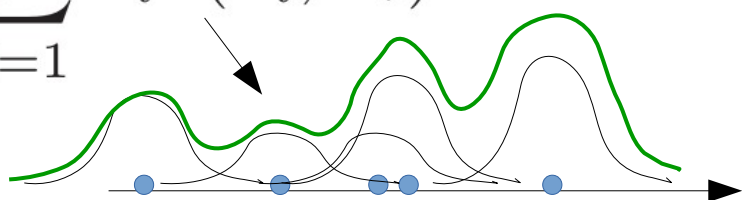
In the case of a single test input, this simplifies as follows

$$p(f_* | \mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(f_* | \mathbf{k}_*^T \mathbf{K}_y^{-1} \mathbf{y}, k_{**} - \mathbf{k}_*^T \mathbf{K}_y^{-1} \mathbf{k}_*)$$

where  $\mathbf{k}_* = [\kappa(\mathbf{x}_*, \mathbf{x}_1), \dots, \kappa(\mathbf{x}_*, \mathbf{x}_N)]$

and where  $k_{**} = \kappa(\mathbf{x}_*, \mathbf{x}_*)$  (=1)

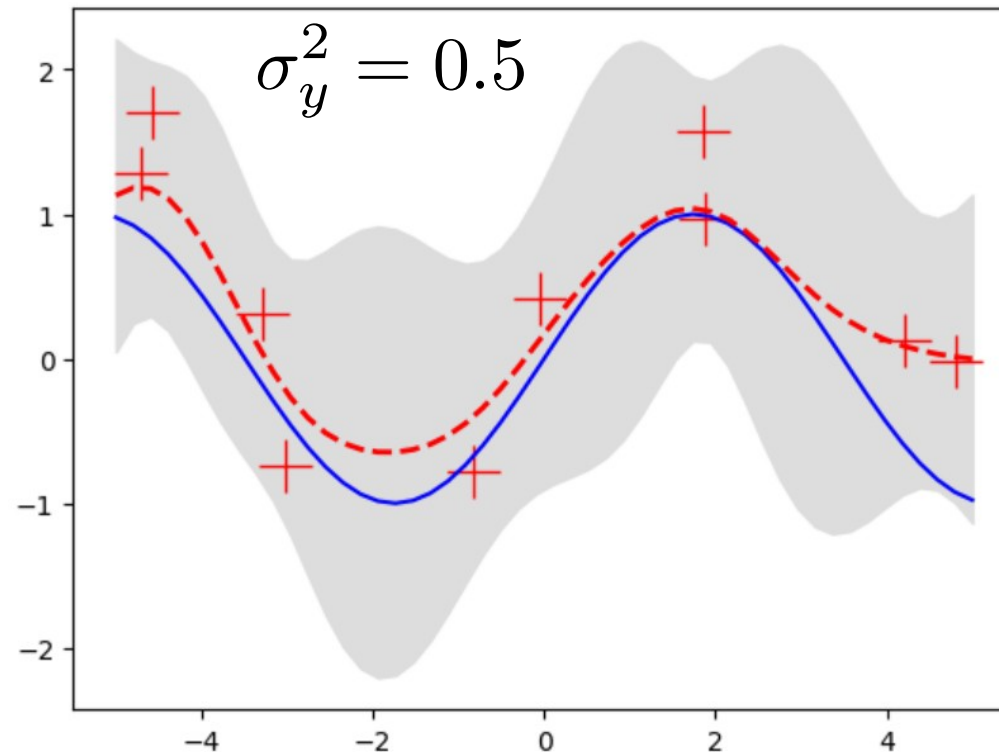
Again, we can write the **posterior mean** as **expansion of basis functions**

$$\bar{f}_* = \overset{1 \times N}{\mathbf{k}_*^T} \overset{N \times N}{\mathbf{K}_y^{-1}} \overset{N \times 1}{\mathbf{y}} = \sum_{i=1}^N \alpha_i \kappa(\mathbf{x}_i, \mathbf{x}_*) \quad \text{where } \alpha = \underbrace{\mathbf{K}_y^{-1} \mathbf{y}}_{\text{from training data}}$$




# Some Plots

cf. `global_solution_yale19/Lecture_5/code/1d_gp_example.py`



- Even in the regions where you have data, there is still uncertainty.
- In the noise-free version of GPR, the uncertainty is 0 at observation points.
- But we still have the same properties as before: where we have data, we are more certain compared to the case where we have no data.

# Noise improves numerical stability

- It is common to use small noise even if there is not any in the data.
- Cholesky fails when covariance is close to being semi-positive definite.
- Adding a small noise improves numerical stability.
- It is known as the “jitter” or as the “nugget” in this case.

# “Learning” the kernel parameters

- ♦ To **estimate the kernel parameters**, we could use **exhaustive search over a discrete grid of values**, with validation loss as an objective, but this can be quite slow.
- ♦ Here we consider an empirical Bayes approach, which will allow us to use continuous optimization methods, which are much faster.
- ♦ In particular, we will **maximize the marginal likelihood**.

# “Learning” the kernel parameters (II)

Marginal likelihood  $p(\mathbf{y}|\mathbf{X}) = \int p(\mathbf{y}|\mathbf{f}, \mathbf{X})p(\mathbf{f}|\mathbf{X})d\mathbf{f}$

Since  $p(\mathbf{f}|\mathbf{X}) = \mathcal{N}(\mathbf{f}|\mathbf{0}, \mathbf{K})$

and  $p(\mathbf{y}|\mathbf{f}) = \prod_i \mathcal{N}(y_i|f_i, \sigma_y^2)$

the (log-) marginal likelihood is given by

$$\log p(\mathbf{y}|\mathbf{X}) = \log \mathcal{N}(\mathbf{y}|\mathbf{0}, \mathbf{K}_y) = -\frac{1}{2}\mathbf{y}\mathbf{K}_y^{-1}\mathbf{y} - \frac{1}{2}\log |\mathbf{K}_y| - \frac{N}{2}\log(2\pi)$$

1<sup>st</sup> term: data fit term

2<sup>nd</sup> term: a model complexity term

3<sup>rd</sup> term: a constant.

# Maximizing the the marginal likelihood

- Let the kernel parameters (also called **hyper-parameters**) be denoted by  **$\theta$**
- One can show that the following holds.

$$\begin{aligned}\frac{\partial}{\partial \theta_j} \log p(\mathbf{y}|\mathbf{X}) &= \frac{1}{2} \mathbf{y}^T \mathbf{K}_y^{-1} \frac{\partial \mathbf{K}_y}{\partial \theta_j} \mathbf{K}_y^{-1} \mathbf{y} - \frac{1}{2} \text{tr}(\mathbf{K}_y^{-1} \frac{\partial \mathbf{K}_y}{\partial \theta_j}) \\ &= \frac{1}{2} \text{tr} \left( (\boldsymbol{\alpha} \boldsymbol{\alpha}^T - \mathbf{K}_y^{-1}) \frac{\partial \mathbf{K}_y}{\partial \theta_j} \right)\end{aligned}$$

where  $\boldsymbol{\alpha} = \mathbf{K}_y^{-1} \mathbf{y}$

## **Computational complexity:**

- It takes  $O(N^3)$  time to compute  $\mathbf{K}_y^{-1}$
- $O(N^2)$  time per hyper-parameter to compute the gradient.

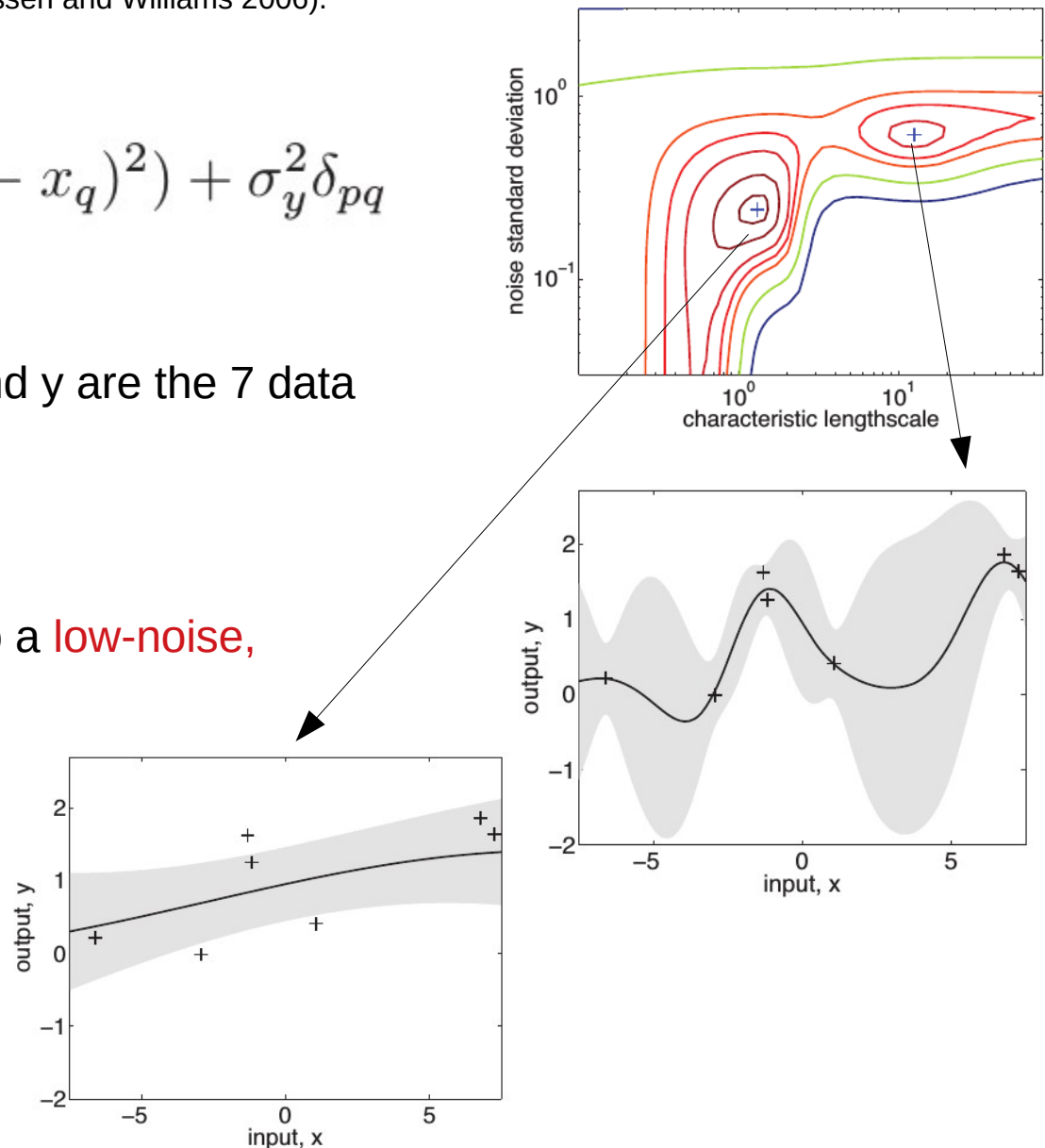
# Careful: Different optima correspond to different interpretations/believes

Fig. 5.5 of (Rasmussen and Williams 2006).

- We use the SE kernel

$$\kappa_y(x_p, x_q) = \sigma_f^2 \exp\left(-\frac{1}{2\ell^2}(x_p - x_q)^2\right) + \sigma_y^2 \delta_{pq}$$

- with  $\sigma_f^2 = 1$
- We plot  $\log p(\mathbf{y}|\mathbf{X}, \ell, \sigma_y^2)$  (where  $\mathbf{X}$  and  $\mathbf{y}$  are the 7 data points shown) as we vary  $\ell$  and  $\sigma_y^2$ .
- The two local optima are indicated by +.
- The bottom left optimum corresponds to a **low-noise, short-length scale solution**.
- The top right optimum corresponds to a **high-noise, long-length scale solution**.
- With only 7 data points, there is not enough evidence to confidently decide which is more reasonable.



# An example: noise and optimization

[https://scikit-learn.org/stable/auto\\_examples/gaussian\\_process/plot\\_gpr\\_noisy\\_targets.html](https://scikit-learn.org/stable/auto_examples/gaussian_process/plot_gpr_noisy_targets.html)  
 global\_solution\_yale19/Lecture\_5/code/GPR\_scikit\_noise.py

```
import numpy as np
from matplotlib import pyplot as plt

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C

np.random.seed(1)

def f(x):
    """The function to predict."""
    return x * np.cos(x)*np.sin(x)

# -----
# Mesh the input space for evaluations of the real function, the prediction and
# its MSE
x = np.atleast_2d(np.linspace(0, 10, 1000)).T

# Instantiate a Gaussian Process model
kernel = C(1.0, (1e-3, 1e3)) * RBF(10, (1e-2, 1e2))

# now the noisy case
X = np.linspace(0.1, 9.9, 20)
X = np.atleast_2d(X).T

# Observations and noise
y = f(X).ravel()
dy = 0.5 + 1.0 * np.random.random(y.shape)
noise = np.random.normal(0, dy)
y += noise

# Instantiate a Gaussian Process model
gp = GaussianProcessRegressor(kernel=kernel, alpha=dy ** 2,
                              n_restarts_optimizer=10)

# Fit to data using Maximum Likelihood Estimation of the parameters
gp.fit(X, y)

# Make the prediction on the meshed x-axis (ask for MSE as well)
y_pred, sigma = gp.predict(x, return_std=True)

# Plot the function, the prediction and the 95% confidence interval based on
# the MSE
plt.figure()
plt.plot(x, f(x), 'r:', label=u'$f(x) = x \cdot \sin(x)$')
plt.errorbar(X.ravel(), y, dy, fmt='r.', markersize=10, label=u'Observations')
plt.plot(x, y_pred, 'b-', label=u'Prediction')
plt.fill(np.concatenate([x, x[::1]]),
         np.concatenate([y_pred - 1.9600 * sigma,
                         (y_pred + 1.9600 * sigma)[::-1]]),
         alpha=.5, fc='b', ec='None', label='95% confidence interval')
plt.xlabel('$x$')
plt.ylabel('$f(x)$')
plt.ylim(-10, 20)
plt.legend(loc='upper left')

plt.show()
```

# A multi-d example

global\_solution\_yale19/Lecture\_5/code/scikit\_multi-d.py

```
import numpy as np
from matplotlib import pyplot as plt
import cPickle as pickle
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF, ConstantKernel as C

np.random.seed(1)

# Test function
def f(x):
    """The 2d function to predict."""
    return np.sin(x[0]) * np.cos(x[1])

# generate training data
n_sample = 100 #points
dim = 2 #dimensions

X = np.random.uniform(-1., 1., (n_sample, dim))
y = np.sin(X[:, 0:1]) * np.cos(X[:, 1:2]) + np.random.randn(n_sample, 1) * 0.005

# Instantiate a Gaussian Process model
kernel = C(1.0, (1e-3, 1e3)) * RBF(10, (1e-2, 1e2))
gp = GaussianProcessRegressor(kernel=kernel, n_restarts_optimizer=9)

# Fit to data using Maximum Likelihood Estimation of the parameters
gp.fit(X, y)

# Make the prediction on the meshed x-axis / training points
y_pred, sigma = gp.predict(X, return_std=True)

#Compute MSE
mse = 0.0
n_sample_test=50
Xtest1 = np.random.uniform(-1., 1., (n_sample_test, dim))
y_pred1, sigma = gp.predict(Xtest1, return_std=True)
for g in range(len(Xtest1)):
    delta = abs(y_pred1[g] - f(Xtest1[g]))
    mse += delta

mse = mse/len(y_pred)
print(".....")
print(" The MSE is ", mse[0])
print(".....")
```



# A multi-d example (II)

global\_solution\_yale19/Lecture\_5/code/scikit\_multi-d.py

```
#-----  
# Important -- save the model to a file  
with open('2d_model.pcl', 'wb') as fd:  
    pickle.dump(gp, fd, protocol=pickle.HIGHEST_PROTOCOL)  
    print("data written to disk")  
  
# Load the model and do predictions  
with open('2d_model.pcl', 'rb') as fd:  
    gm = pickle.load(fd)  
    print("data loaded from disk")  
  
# generate training data  
n_test = 50  
dim = 2  
Xtest = np.random.uniform(-1., 1., (n_test, dim))  
y_pred_test, sigma_test = gm.predict(Xtest, return_std=True)  
  
MSE2 = 0  
for a in range(len(Xtest)):  
    delta = abs(y_pred_test[a] - f(Xtest[a]))  
    MSE2 += delta  
  
MSE2 = MSE2/len(Xtest)  
print(".....")  
print(" The MSE 2 is ", MSE2[0])  
print(".....")  
#-----
```

# More on Kernels

<http://www.gaussianprocess.org/gpml/chapters/RW4.pdf>

See also e.g. "The Kernel Cookbook": <https://www.cs.toronto.edu/~duvenaud/cookbook/>

[https://scikit-learn.org/stable/modules/gaussian\\_process.html#kernels-for-gaussian-processes](https://scikit-learn.org/stable/modules/gaussian_process.html#kernels-for-gaussian-processes)

- Our **prior beliefs** about the response are **encoded in our choice of the mean and covariance functions**.
- The choice of an appropriate kernel is **based on assumptions** such as **smoothness** and **likely patterns to be expected in the data**.
- A sensible assumption is usually that **the correlation between two points decays with distance between the points according to some power function**.
- The choice of kernel determines almost all the generalization properties of a GP model.
- **You are the expert on your modeling problem - so you're the person best qualified to choose the kernel!**

# Stationary versus non-stationary Kernels

Two categories of kernels can be distinguished:

- ♦ **Stationary kernels:**

They depend only on the **distance** of two data points and **not on their absolute values**

$k(x_i, x_j) = k(d(x_i, x_j))$  and are thus invariant to translations in the input space

Stationary kernels can further be subdivided into **isotropic** and **anisotropic** kernels, where isotropic kernels are also invariant to rotations in the input space.

- ♦ **Non-stationary kernels:**

They depend also on the **specific values of the data points**.

# Squared Exponential Kernel

The SE kernel has become the **de-facto default kernel** for GPs.

$$k_{\text{SE}}(x, x') = \sigma^2 \exp \left( -\frac{(x - x')^2}{2\ell^2} \right)$$

This is probably because it has some nice properties:

- It is **universal**, and **you can integrate it against** most functions that you need to.
- Every function in its prior has infinitely many derivatives.
- It also has only two parameters:
  - The **lengthscale  $\ell$**  determines the length of the 'wiggles' in your function. In general, **you won't be able to extrapolate more than  $\ell$  units away from your data.**
  - The output **variance  $\sigma^2$**  determines the average distance of your function away from **its mean**. Every kernel has this parameter out in front; it's just a scale factor.

# Pitfalls for the SE kernel

- Most people who set up a GP regression (or classification) model end up using the SE kernel.
- They are a **quick-and-dirty solution** that will probably **work pretty well for interpolating smooth functions when  $N$  is a multiple of  $D$** , and when there are **no 'kinks'** in your function.
- If your function happens to have a **discontinuity** or is **discontinuous in its first few derivatives** (for example, the `abs()` function), then either **your length-scale will end up being extremely short**, and your posterior mean will become zero almost everywhere, or your posterior mean will have 'ringing' effects.
- Even if there are no hard discontinuities, the **length-scale will usually end up being determined by the smallest 'wiggle' in your function** - so you might end up **failing to extrapolate in smooth regions** if there is even a **small non-smooth** region in your data.
- **If your data is more than two-dimensional, it may be hard to detect this problem.** One indication is if the length-scale chosen by maximum marginal likelihood never stops becoming smaller as you add more data. This is a classic sign of **model misspecification**.

# Periodic Kernel



$$k_{\text{Per}}(x, x') = \sigma^2 \exp \left( -\frac{2 \sin^2(\pi |x - x'|/p)}{\ell^2} \right)$$

- The periodic kernel allows one to model functions which repeat themselves exactly.
- Its parameters are easily interpretable:
  - The period  **$p$**  simply determines the **distance between repetitions** of the function.
  - The **length-scale  $\ell$**  determines the length-scale function in the same way as in the SE kernel.

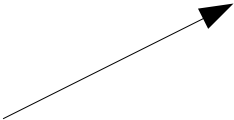
# Matérn kernel

- The **Matérn kernel** is a stationary kernel and a **generalization of the RBF kernel**.
- It has an **additional parameter  $\nu$  which controls the smoothness** of the resulting function. It is parameterized by a length-scale parameter  $\ell > 0$ , which can either be a scalar (isotropic variant of the kernel) or a vector with the same number of dimensions as the inputs (anisotropic variant of the kernel).
- The kernel is given by:

$$k(x_i, x_j) = \sigma^2 \frac{1}{\Gamma(\nu) 2^{\nu-1}} \left( \gamma \sqrt{2\nu} d(x_i/l, x_j/l) \right)^\nu K_\nu \left( \gamma \sqrt{2\nu} d(x_i/l, x_j/l) \right)$$

# Matérn kernel (II)

- As  $\nu \rightarrow 0$ , the Matérn kernel converges to the RBF kernel.
- When  $\nu = 1/2$ , the Matérn kernel becomes identical to the absolute exponential kernel.

$$k_{\text{mat}}(x, x') = \sigma^2 \left( 1 + \sqrt{3} \sum_{i=1}^l \frac{(x_i - x'_i)^2}{\ell_i^2} \right) \exp \left( -\sqrt{3} \sum_{i=1}^l \frac{(x_i - x'_i)^2}{\ell_i^2} \right)$$


- These are popular choices for learning functions that are not infinitely differentiable (as assumed by the RBF kernel) **but at least once ( $\nu = 3/2$ )** (if  **$\nu = 5/2$ , the kernel is twice differentiable**).



# Combining/Adding Kernels

- Roughly speaking, adding two kernels can be thought of as an **OR** operation.  
 → If you add together two kernels, then the resulting kernel will have high value if either of the two base kernels have a high value.

- **Linear plus Periodic Kernel**



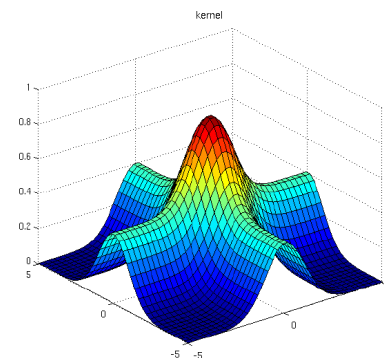
- A linear kernel plus a periodic results in functions which are periodic with increasing mean as we move away from the origin.

- **Adding across dimensions**

- Adding kernels which each depend only on a single input dimension results in a prior over functions which are a sum of one-dimensional functions, one for each dimension. That is, the function  $f(x,y)$  is simply a sum of two functions  $f_x(x) + f_y(y)$

- These kernels have the form:

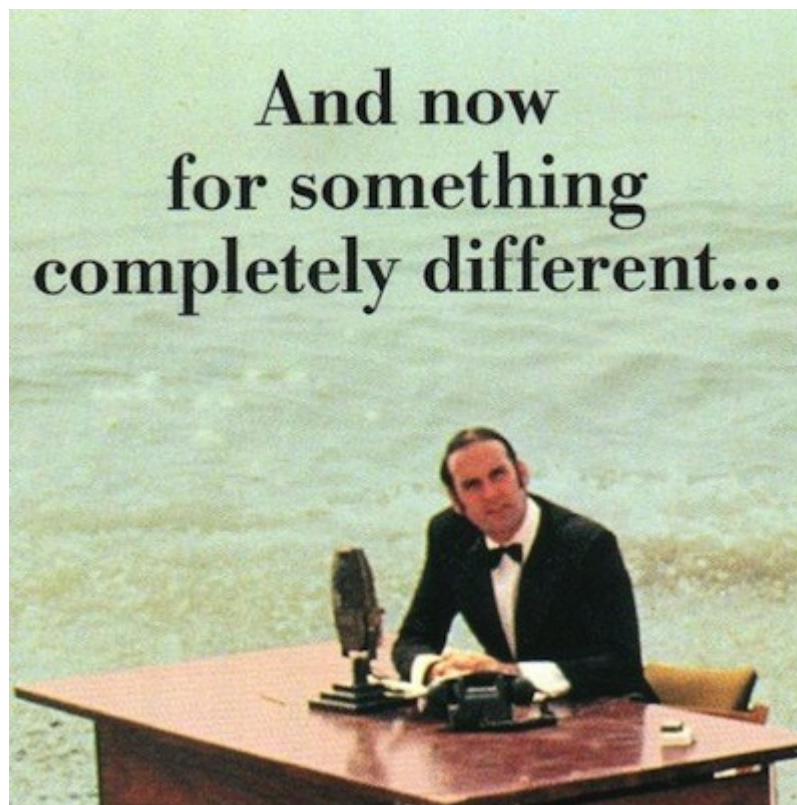
$$k_{\text{additive}}(x, y, x', y') = k_x(x, x') + k_y(y, y')$$



# Automatically choosing Kernels

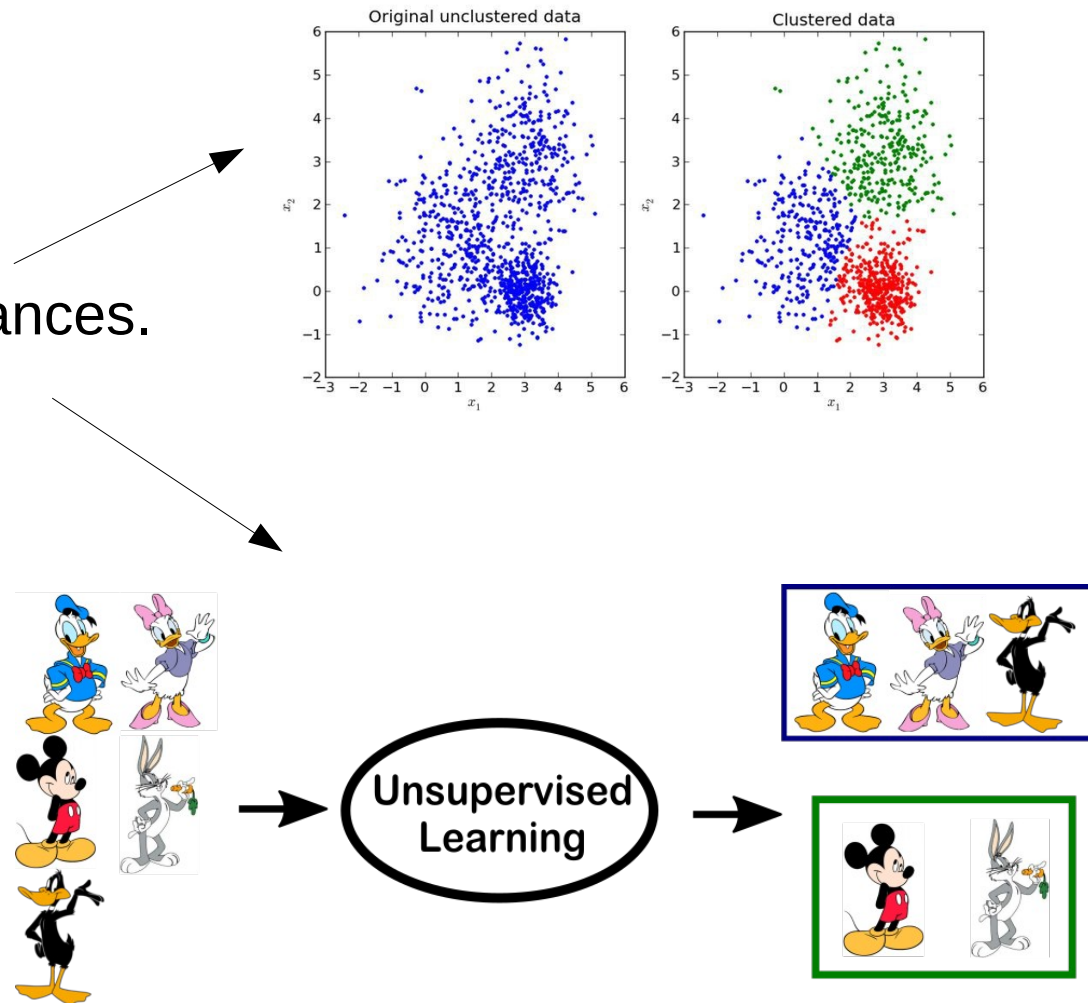
- Sometimes, it is not obvious which kernel is appropriate for your problem.
- In fact, you might decide that choosing the kernel is one of the main difficulties in doing inference
- Just as you don't know what the true parameters are, you also don't know what the true kernel is.
- Probably, you should try out a few different kernels at least, and compare their marginal likelihood on your training data.
- Automatic ways:
  - <https://arxiv.org/abs/1302.4922>  
(Structure Discovery in Nonparametric Regression through Compositional Kernel Search)
  - <https://github.com/jamesrobertlloyd/gp-structure-search>

## II. Gaussian Mixture Models (GMM)



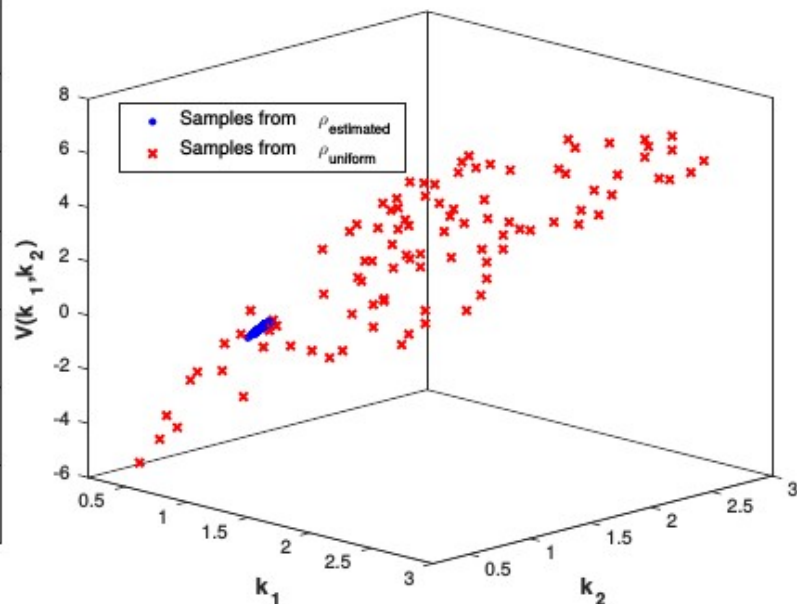
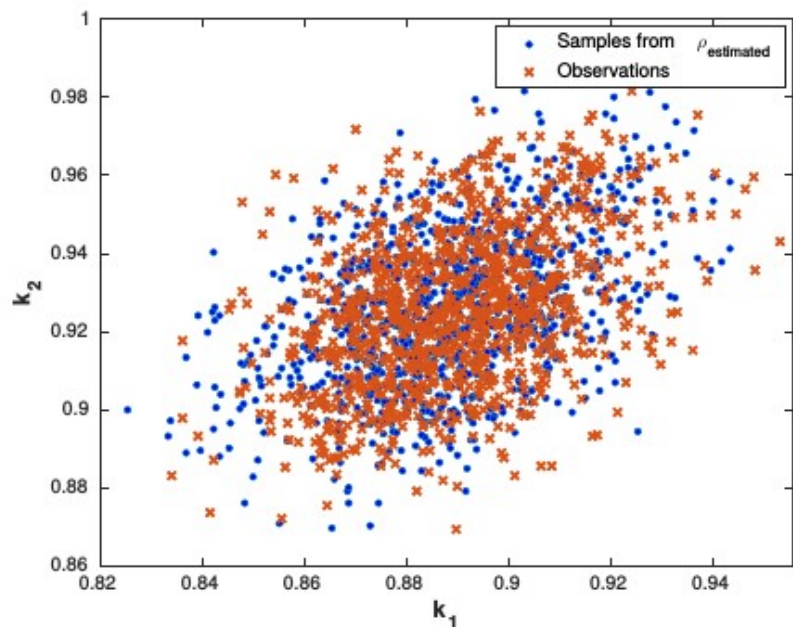
# Recall: Unsupervised Machine Learning

- Learning “what normally happens”.
- No output.
- Clustering: Grouping similar instances.
- Example applications:
  - Customer segmentation.
  - Image compression: Color quantization.
  - Bioinformatics: Learning motifs.



# Motivation – Ergodic Sets

Den Haan and Marcet (1994), Judd et al. (2010, and Maliar and Maliar (2015), Scheidegger & Bilonis (2017)



$$V(\mathbf{k}) = \max_{\mathbf{l}, \mathbf{c}, \mathbf{l}} \left( u(\mathbf{c}, \mathbf{l}) + \beta \mathbb{E} \left\{ V_{next}(\mathbf{k}^+) \right\} \right),$$

*s.t.*

$$\underline{k_j^+ = (1 - \delta) \cdot k_j + I_j + \varepsilon_j, \quad j = 1, \dots, D}$$

$$\Gamma_j = \frac{\zeta}{2} k_j \left( \frac{I_j}{k_j} - \delta \right)^2, \quad j = 1, \dots, D$$

$$\sum_{j=1}^D \left\{ c_j + I_j - \delta \cdot k_j \right\} = \sum_{j=1}^D \left\{ f(k_j, l_j) - \Gamma_j \right\},$$

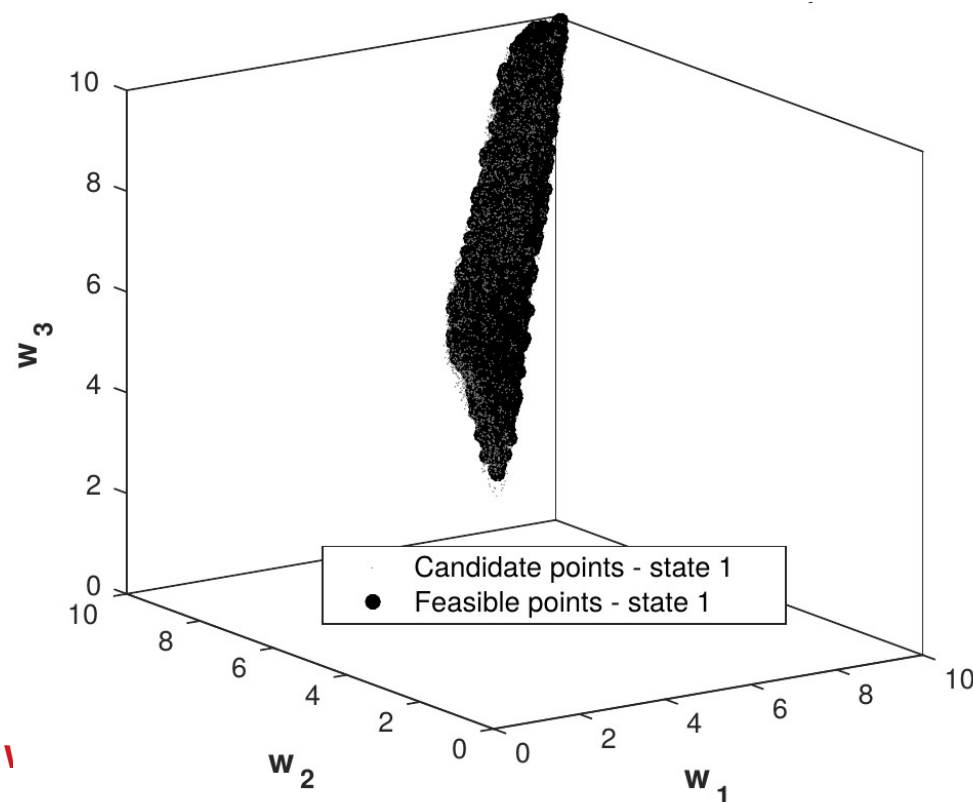
—————▶ Simulate the Model  
(cf. Lecture 6)

# Motivation – Feasible Sets

see, e.g., Abreu et al. (1986/1990), Fernandes & Phelan (2000)

See, e.g., dynamic incentive problems:

- Domain of interest is high-dimensional.
  - Irregularly-shaped.
  - You need to approximate Value- and Policy functions on such a domain.
  - In the multi-d setting, if you use grid-based methods, you spend way more than 99% of your resources in vain.
- Q: How can we represent such sets?
- Q: How can we generate “observations” from such sets?
- One way to do so are **Mixture of Gaussians** (see, e.g., Bishop (2006)).
- Approximate the set in a probabilistic fashion.



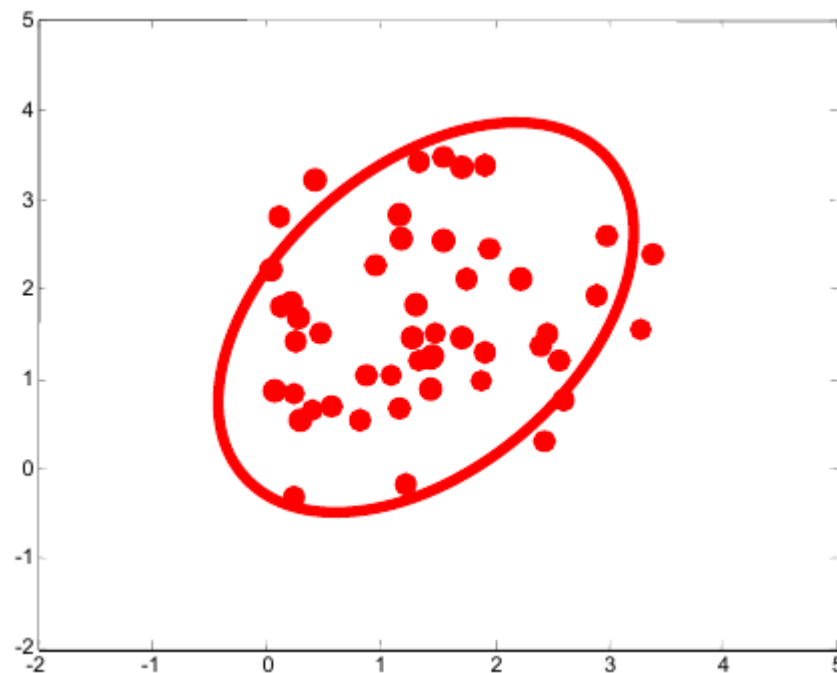
# Recall Multivariate Gaussians

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

Maximum Likelihood estimates given by:

$$\hat{\mu} = \frac{1}{N} \sum_i x^{(i)}$$

$$\hat{\Sigma} = \frac{1}{N} \sum_i (x^{(i)} - \hat{\mu})^T (x^{(i)} - \hat{\mu})$$





# Mixture of Gaussians – the basic idea

Consult e.g. the books by Bishop (2006) or Murphy (2012) for more details.

- Figure: plots of the ‘old faithful’ data.
- The blue curves show contours of **constant probability density**.
- On the left is a **single Gaussian distribution** which has been fitted to the data using maximum likelihood.

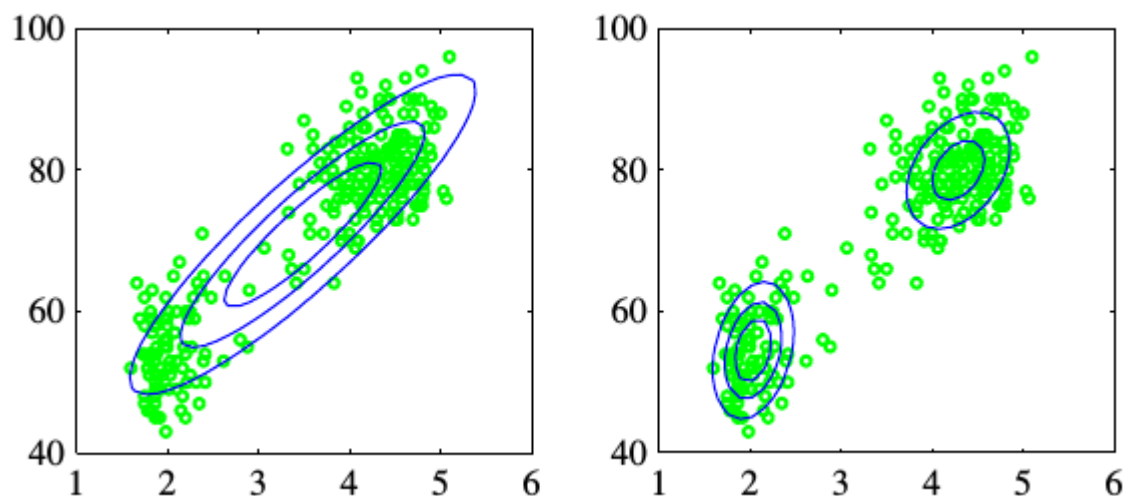


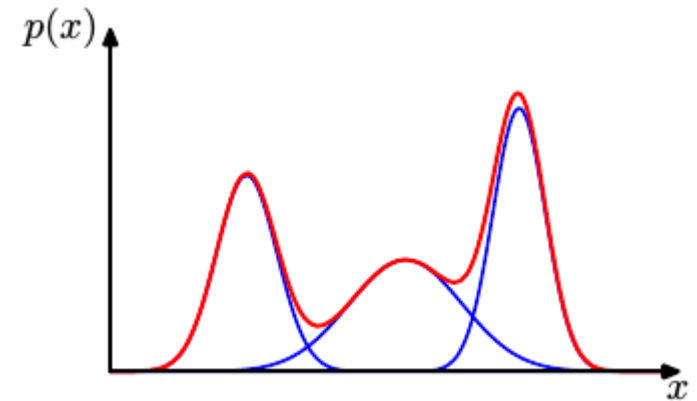
Fig. From Bishop (2006)

- Note that **this distribution fails to capture the two clumps in the data** and indeed places much of its probability mass in the central region between the clumps where the data are relatively sparse.
- On the right the distribution is given by a **linear combination of two Gaussians** which has been fitted to the data, and which gives a better representation of the data.



# GMM basics

- Example of a Gaussian mixture distribution  $p(x)$  in one dimension showing three Gaussians (each scaled by a coefficient) in blue and their sum in red.



- **Linear combinations of Gaussians** can give rise to very complex densities.
- By using a **sufficient number of Gaussians and adjusting their means covariances** as well as the coefficients in the linear combination, almost any density can be approximated with arbitrary accuracy.
- $k$  latent variables.

# Superposition of Gaussians

- Formally, a GMM is:  $p(\mathbf{x}) = \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$  **(A)**
- Each Gaussian density  $\mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$  is called **a component of the mixture**.
- It has its own **mean  $\boldsymbol{\mu}_k$**  and **covariance  $\boldsymbol{\Sigma}_k$** .
- $\pi_k$  : mixing coefficients.**

→ If we integrate both sides of **(A)** with respect to  $\mathbf{x}$ , and note that both  $p(\mathbf{x})$  and the individual Gaussian components are normalized, one obtains:

$$\sum_{k=1}^K \pi_k = 1. \quad 0 \leq \pi_k \leq 1.$$

$$\left[ p(\mathbf{x}) \geq 0 \quad \mathcal{N}(\mathbf{x} | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \geq 0 \right]$$

# Superposition of Gaussians (II)

- We therefore see that **the mixing coefficients satisfy the requirements to be probabilities.**
- From the sum and product rules, the marginal density is given by

$$p(\mathbf{x}) = \sum_{k=1}^K p(k)p(\mathbf{x}|k) \quad \longleftrightarrow \quad \pi_k = p(k)$$

- This is equivalent say that  **$\pi_k = p(k)$  as the prior probability of picking the k-th component.**
- The density  $N(\mathbf{x}|\mu_k, \Sigma_k) = p(\mathbf{x}|k)$  can be viewed as the probability of  $\mathbf{x}$  conditioned on  $k$ .
  - One can think of  $p(\mathbf{x}|k)$  as the pdf of cluster  $k$  (latent class labels).

# Responsibilities: Posterior Distribution of Cluster Labels

- The **posterior probabilities**  $p(k|\mathbf{x})$  are also known as *responsibilities*

$$\begin{aligned}\gamma_k(\mathbf{x}) &\equiv p(k|\mathbf{x}) \\ &= \frac{p(k)p(\mathbf{x}|k)}{\sum_l p(l)p(\mathbf{x}|l)} \\ &= \frac{\pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}{\sum_l \pi_l \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_l, \boldsymbol{\Sigma}_l)}\end{aligned}$$

- This represents the posterior probability that **point i belongs to cluster k**.
- This is sometimes known as **soft clustering**.
- We can represent the **amount of uncertainty in the cluster assignment** by using  $1 - \max_k \gamma_k(\mathbf{x})$ .

## The likelihood

- The form of the Gaussian mixture distribution is governed by the parameters  $\boldsymbol{\pi}$ ,  $\boldsymbol{\mu}$  and  $\boldsymbol{\Sigma}$ , where we have used the notation  $\boldsymbol{\pi} \equiv \{\pi_1, \dots, \pi_k\}$ ,  $\boldsymbol{\mu} \equiv \{\mu_1, \dots, \mu_k\}$ ,  $\boldsymbol{\Sigma} \equiv \{\Sigma_1, \dots, \Sigma_k\}$ .
- One way to set the values of these parameters is to use **maximum likelihood**.
- The log of the likelihood function is given by

$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

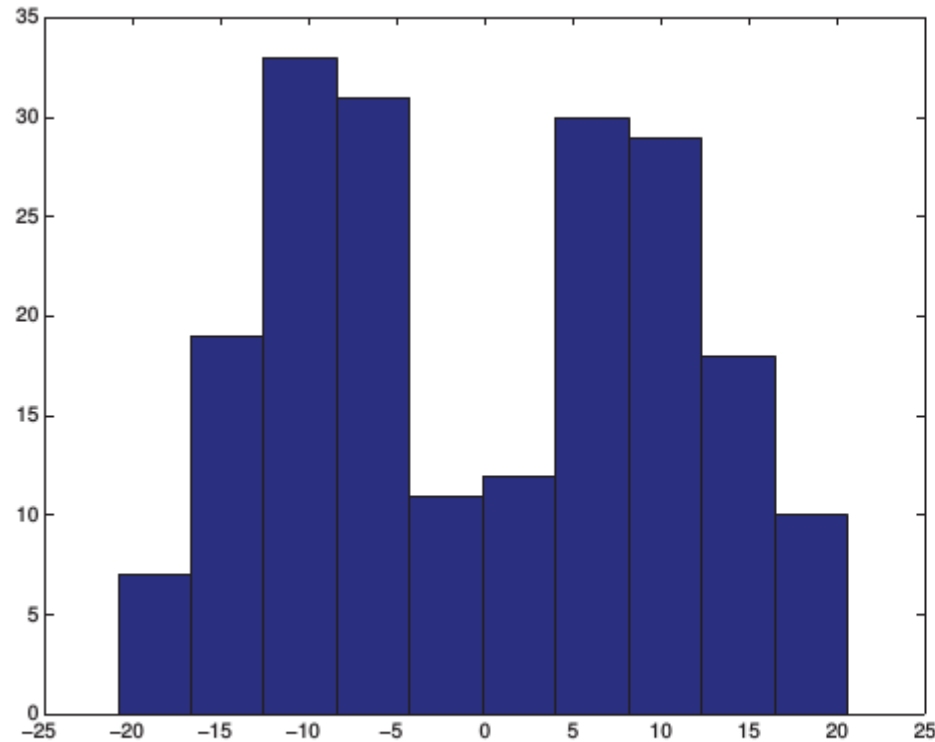
where  $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ .

# Problems with optimizing the likelihood

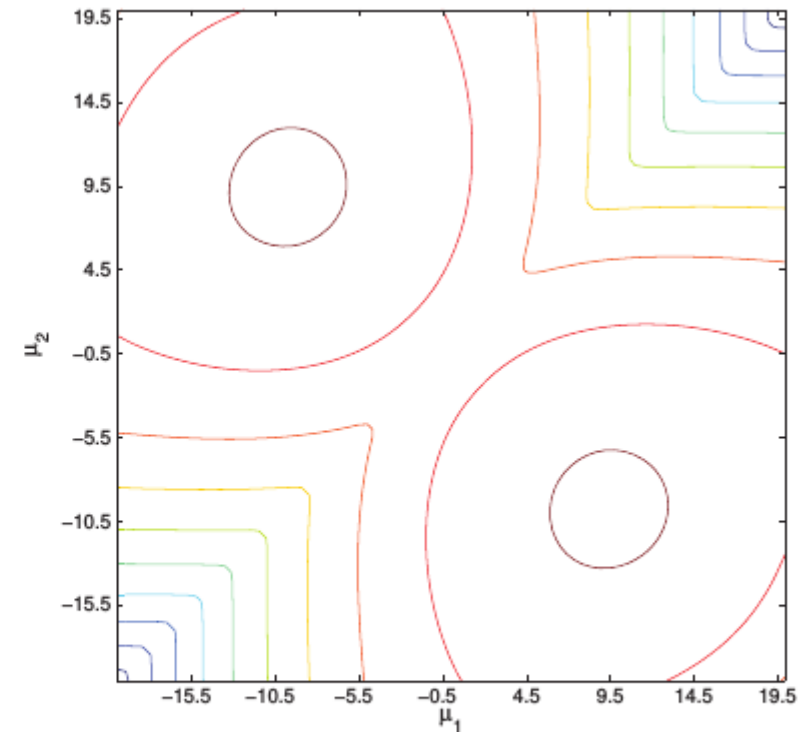
- The situation is now much more complex than with a single Gaussian, due to the **presence of the summation over  $k$  inside the logarithm**.
- As a result, **the maximum likelihood solution for the parameters no longer has a closed-form analytical solution**.
- One approach to maximizing the likelihood function is to use **iterative numerical optimization techniques**.
- Gradient methods could be used but are painful to implement.
  - **Non-convex optimization problem!** (multiple optima possible)

# Unidentifiability

Fig. From Murphy (2012)



(a)



(b)

**Left panel:**  $N = 200$  data points sampled from a mixture of 2 Gaussians in 1d, with  $\pi_k = 0.5$ ,  $\sigma_k = 5$ ,  $\mu_1 = -10$  and  $\mu_2 = 10$ .

**Right panel:** Likelihood surface  $p(D|\mu_1, \mu_2)$ , with all other parameters set to their true values.  
 → We see the two symmetric modes, reflecting the unidentifiability of the parameters.

# Examplein 1d

Observations  $x_1 \dots x_n$

- K=2 Gaussians with unknown  $\mu$ ,  $\sigma^2$
- Estimation trivial if we know the source of each observation

$$\mu_b = \frac{x_1 + x_2 + \dots + x_{n_b}}{n_b}$$

$$\sigma_b^2 = \frac{(x_1 - \mu_1)^2 + \dots + (x_n - \mu_n)^2}{n_b}$$





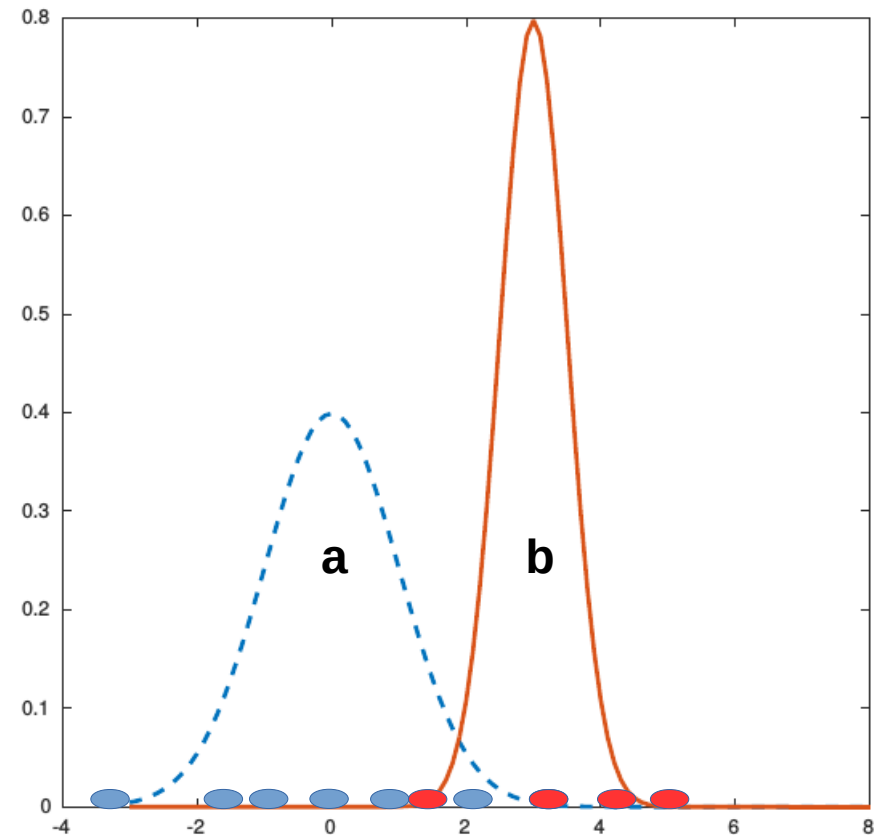
# Example: Expectation Maximization in 1d

see, e.g., Dempster et al. (1977)

Observations  $x_1 \dots x_n$

- K=2 Gaussians with unknown  $\mu$ ,  $\sigma^2$
- Estimation trivial if we know the source of each observation

$$\mu_b = \frac{x_1 + x_2 + \dots + x_{n_b}}{n_b}$$
$$\sigma_b^2 = \frac{(x_1 - \mu_b)^2 + \dots + (x_{n_b} - \mu_b)^2}{n_b}$$



# Example: Expectation Maximization in 1d (II)

- What if we don't know the source?
- If we knew parameters of the Gaussians ( $\mu$ ,  $\sigma^2$ )

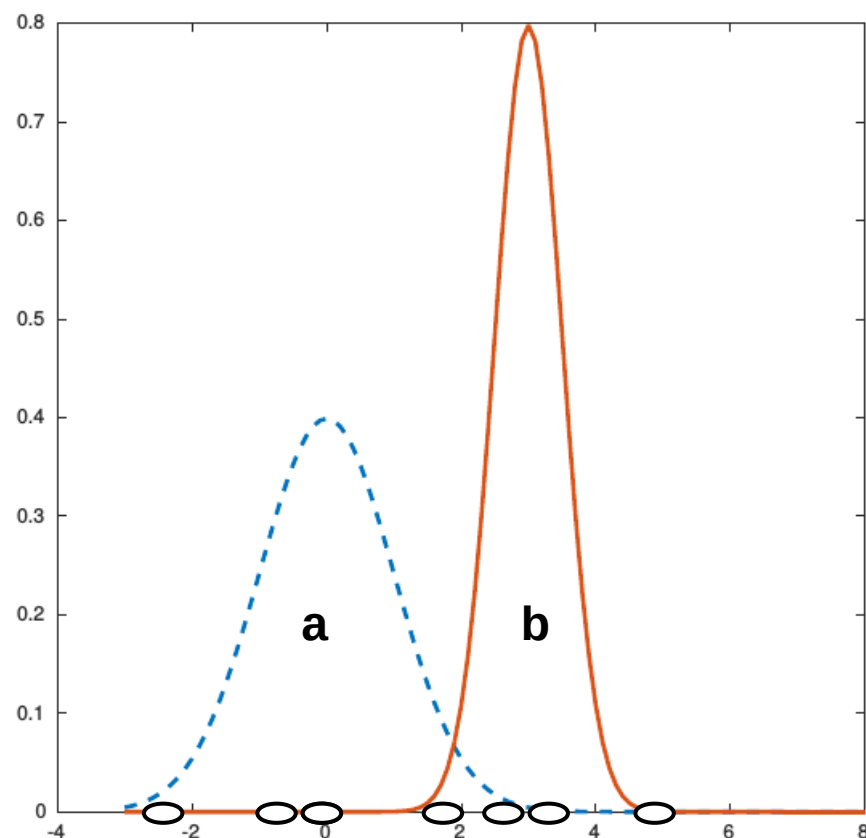


# Example: Expectation Maximization in 1d (II)

- What if we don't know the source?
  - If we knew parameters of the Gaussians ( $\mu$ ,  $\sigma^2$ )
- can guess whether point is more likely to be a or b.

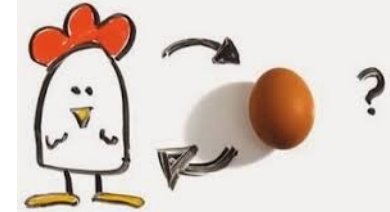
$$P(b | x_i) = \frac{P(x_i | b)P(b)}{P(x_i | b)P(b) + P(x_i | a)P(a)}$$

$$P(x_i | b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp\left(-\frac{(x_i - \mu_b)^2}{2\sigma_b^2}\right)$$



# EM Algorithm (in 1d)

see, e.g., Dempster et al. (1977), Bishop (2006), Murphy (2012) and references therein for details.



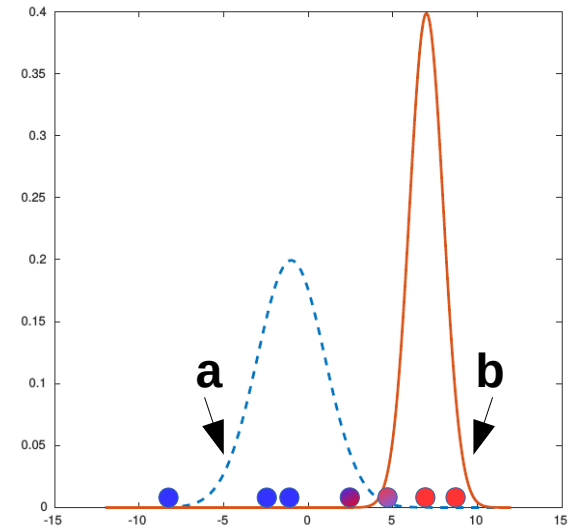
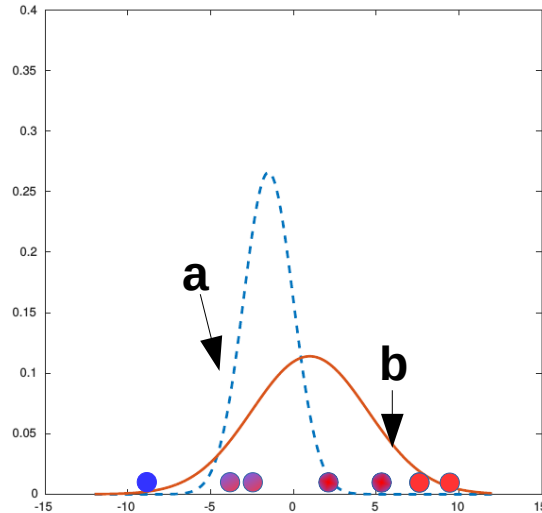
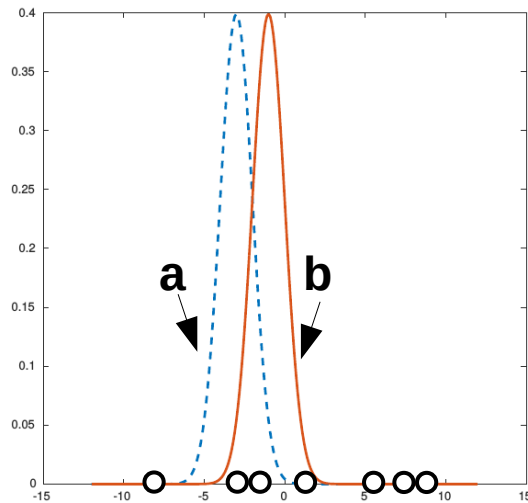
A fundamental problem:

- we need  $(\mu_a, \sigma_a^2)$  and  $(\mu_b, \sigma_b^2)$  to guess the source of the points.
- we need to know the source to estimate  $(\mu_a, \sigma_a^2)$  and  $(\mu_b, \sigma_b^2)$ .

## EM algorithm

1. **start** with two randomly placed Gaussians  $(\mu_a, \sigma_a^2)$  and  $(\mu_b, \sigma_b^2)$ .
2. **E(xpectation)-step:**
  - for each point:  $P(b|x_i)$  = does it look like it came from b?
3. **M(maximization)-step:**
  - adjust  $(\mu_a, \sigma_a^2)$  and  $(\mu_b, \sigma_b^2)$  to fit points assigned to them.
4. **Iterate until convergence.**

# EM in 1d



$$P(x_i | b) = \frac{1}{\sqrt{2\pi\sigma_b^2}} \exp\left(-\frac{(x_i - \mu_b)^2}{2\sigma_b^2}\right)$$

$$b_i = P(b | x_i) = \frac{P(x_i | b)P(b)}{P(x_i | b)P(b) + P(x_i | a)P(a)}$$

$$a_i = P(a | x_i) = 1 - b_i$$

$$\mu_b = \frac{b_1 x_1 + b_2 x_2 + \dots + b_n x_n}{b_1 + b_2 + \dots + b_n}$$

$$\sigma_b^2 = \frac{b_1 (x_1 - \mu_b)^2 + \dots + b_n (x_n - \mu_b)^2}{b_1 + b_2 + \dots + b_n}$$

$$\mu_a = \frac{a_1 x_1 + a_2 x_2 + \dots + a_n x_n}{a_1 + a_2 + \dots + a_n}$$

$$\sigma_a^2 = \frac{a_1 (x_1 - \mu_a)^2 + \dots + a_n (x_n - \mu_a)^2}{a_1 + a_2 + \dots + a_n}$$

→ We could also estimate priors:  
 $P(b) = (b_1 + b_2 + \dots + b_n) / n$   
 $P(a) = 1 - P(b)$

# EM in the multidimensional case

- Start with parameters describing each cluster
- Mean  $\mu_c$ , Covariance  $\Sigma_c$ , “size”  $\pi_c$
- **E-step (“Expectation”):**
  - For each observation  $x_i$
  - Compute “ $r_{ic}$ ”, the probability that it belongs to cluster  $c$ .
    - Compute its probability under model  $c$ .
    - Normalize to sum to one (over clusters  $c$ ).

$$r_{ic} = \frac{\pi_c \mathcal{N}(x_i ; \mu_c, \Sigma_c)}{\sum_{c'} \pi_{c'} \mathcal{N}(x_i ; \mu_{c'}, \Sigma_{c'})}$$

- If  $x_i$  is very likely under the  $c$ -th Gaussian, it gets high weight.
- Denominator just makes  $r$ 's sum to one.

# EM in the multidimensional case

- **M-step (“Maximization step”):**
  - For each cluster (Gaussian)  $x_c$
  - Update its parameters using the (weighted) data points

$$N_c = \sum_i r_{ic}$$

Total responsibility allocated to cluster  $c$

$$\pi_c = \frac{N_c}{N}$$

Fraction of total assigned to cluster  $c$

$$\mu_c = \frac{1}{N_c} \sum_i r_{ic} x_i$$

Weighted mean of assigned data

$$\Sigma_c = \frac{1}{N_c} \sum_i r_{ic} (x_i - \mu_c)^T (x_i - \mu_c)$$

Weighted covariance of assigned data  
(use new weighted means here)

# Expectation-Maximization: Summary

- Likelihood of the data

$$P(x_1, \dots, x_N) = \prod_{i=1}^N \sum_{k=1}^K P(x_i|k)P(k)$$

- Each step increases the log-likelihood of our model

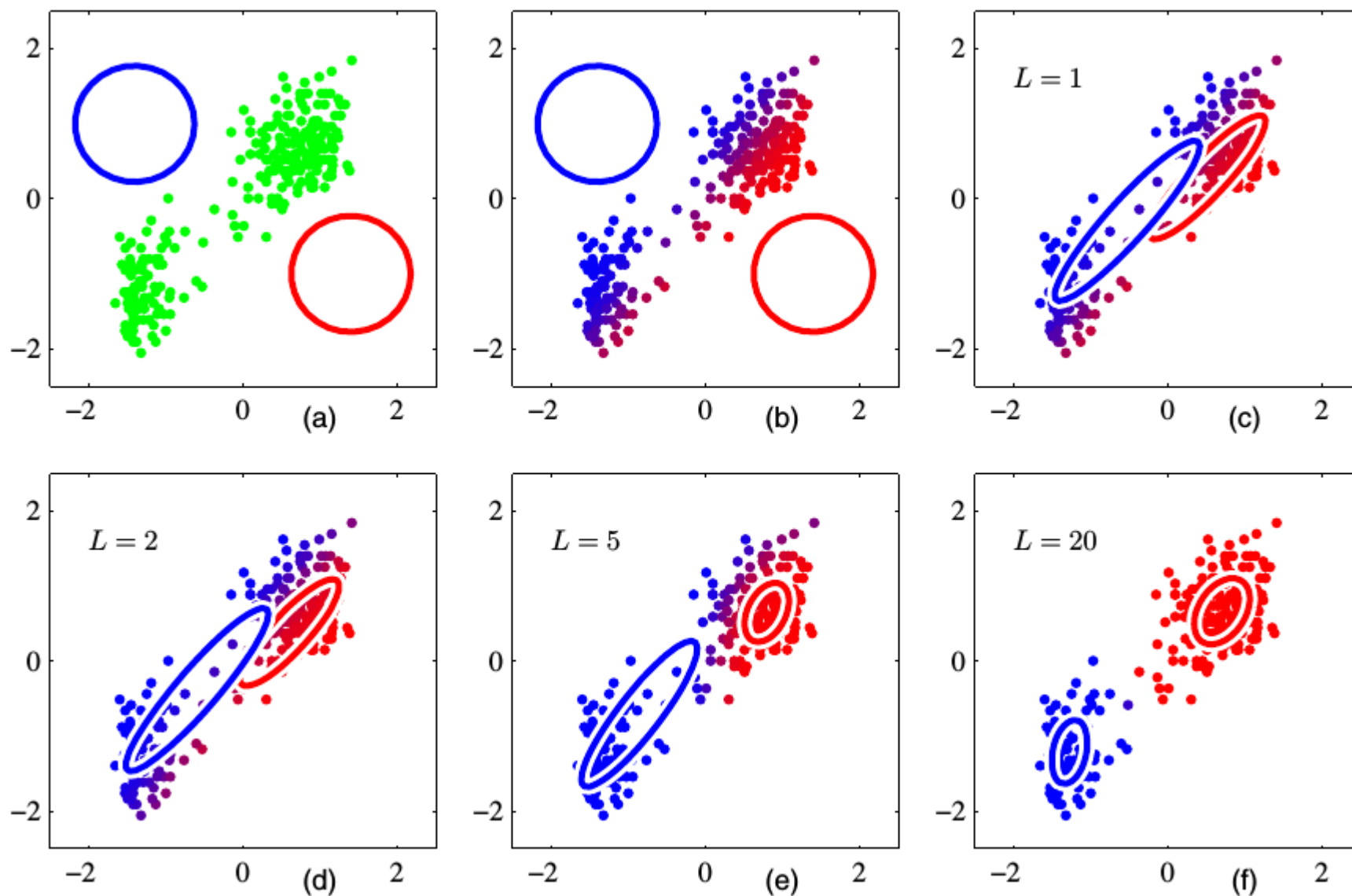
$$\ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$

- Iterate until convergence
  - Convergence guaranteed – another ascent method.
- Cannot discover  $k$ .



# Gaussian mixture models: $d > 1$

See Bishop (2006) for details



# Bayesian Information Criterion (BIC)

See, e.g., Alpaydin (2014), MIT Press

- How to pick  $k$ ?
- Probabilistic model: 
$$L = \ln p(\mathbf{X}|\boldsymbol{\pi}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \sum_{n=1}^N \ln \left\{ \sum_{k=1}^K \pi_k \mathcal{N}(\mathbf{x}_n | \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k) \right\}$$
  - Tries to “fit” the data (maximize likelihood)
- Choose  $k$  that makes  $L$  as large as possible?
  - $K = n$ : each data point has its own “source”
  - may not work well for new data points
- Split points into training set  $\mathbf{T}$  and validation set  $\mathbf{V}$ 
  - for each  $k$ : fit parameters of  $\mathbf{T}$
  - measure likelihood of  $\mathbf{V}$
  - sometimes still best when  $k = n$
- “Occam’s razor”:
  - Pick the “simplest” of all models that fits the data.
  - Assess, e.g., via Bayes Information Criterion (BIC):  $\max_p \{ L - 1/2 p \log(n) \}$
  - $L$ : Likelihood;  $p$ : # Parameters in the model – how simple is the model.

# About the EM Algorithm

## Some good things about EM:

- ♦ no learning rate (step-size) parameter.
- ♦ automatically enforces parameter constraints.
- ♦ very fast for low dimensions.
- ♦ **each iteration guaranteed to improve likelihood.**

## Some bad things about EM:

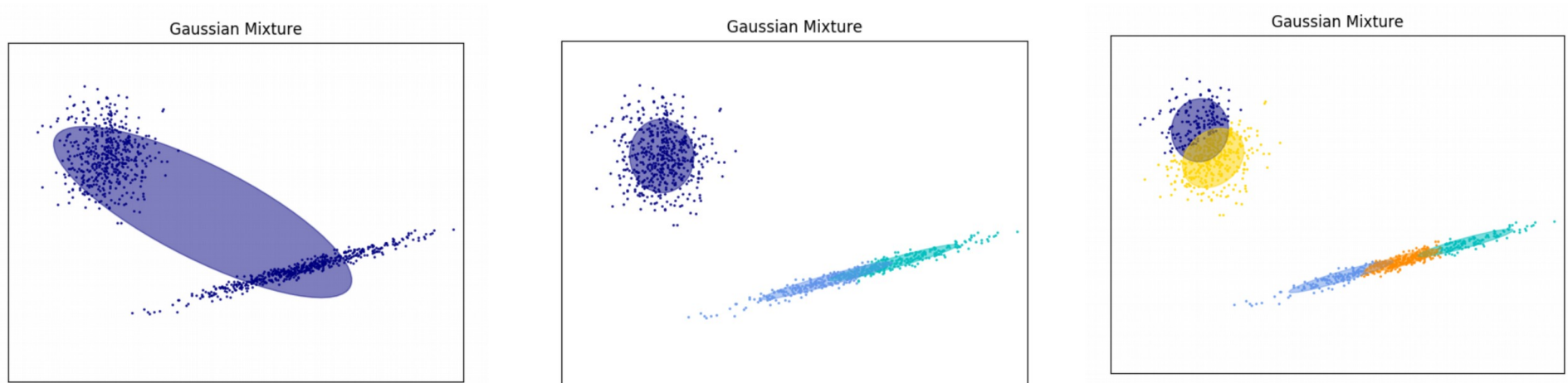
- ♦ can get stuck in local minima.
- ♦ can be slower than conjugate gradient (especially near convergence).
- ♦ requires expensive inference step.
- ♦ is a maximum likelihood/MAP (maximum a posterior) method.

# Hands-on example 1

<https://scikit-learn.org/stable/modules/mixture.html>

global\_solution\_yale19/Lecture\_5/code/GMM\_scikit\_example.py

- Plot the confidence ellipsoids of a mixture of two Gaussians obtained with Expectation Maximization (GaussianMixture class)
- The model has access to 1, 3, and 5 components with which to fit the data. Note that the Expectation Maximization model will necessarily use ALL components
- In the 5-component example, we can see that the Expectation Maximization model splits some components arbitrarily, because it is trying to fit too many components.



# Hands-on example 2

global\_solution\_yale19/Lecture\_5/code/code/BGMM\_data  
cf. [https://papers.ssrn.com/sol3/papers.cfm?abstract\\_id=3282487](https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3282487)

- We simulate a bunch of data (e.g., an ergodic set).
  - Its in a text file (**ergodic\_data.txt** – 3 dimensions)

- We apply GMM (**build\_density.py**)

- We can sample data from the fitted GMM model (**sample.py**)

