An application of the EM Algorithm and Gaussian Processes in Bioinformatics

Student Sebastian Ciobanu
sebastian.ciobanu@info.uaic.ro

Associate Professor Liviu Ciortuz
ciortuz@info.uaic.ro

Contents

1 Introduction

Bioinformatics is an interdisciplinary field that develops methods and software tools for understanding biological data [2]. The problem tackled in this paper is related to *gene expression*. We will use the EM algorithm for Gaussian Process Mixture Model (EM/GPMM) to solve it. In the second section, we will present the problem we would like to solve. This problem was taken from a Machine Learning course at MIT, more specifically: [1]. In the third section, we will discuss about the theory that must be known in order to solve the problem. In the fourth section, based on the information gained from the third section, we will solve the initial problem. The last section is for conlusions.

2 The problem

The text was taken from: [1].

The motivating application is as follows: we would like to understand how the expression levels of genes (in a cell) change with time. The expression level of a gene indicates, roughly, the amount of gene-product in the cell.

The experimental data we have available are noisy measurements of this level (for each gene) at certain time points. More precisely, we have m genes $(g_1, \ldots, g_m)$ and measurements are available for r timepoints $(t_1, \ldots, t_r)$. The data matrix is $Y = [y_1, \ldots, y_m]^T$ where each row $y_i^T = [y_{i1}, ..., y_{ir}]$ corresponds to the expression levels of gene g_i across the r timepoints.

Our goal is to estimate continuous functions, each capturing the expression level of one gene. These can also be called expression curves. Clearly, if we treat each gene's expression as independent from others, we would need to perform m unrelated regression tasks. We can do better, however, since we expect that there are groups of genes with similar expression curves.

We hypothesize that the expression curves can be grouped into k classes and that, within each group, the functions look very similar. When estimating a curve we can pool together data from all the genes in the same class. We have two problems to solve:

- Figure out which class each gene belongs to
- Given class assignments, estimate the expression curves

To perform **regression**, we will use **Gaussian Process Regression (GPR)**.

To perform **clustering**, we wil use **the EM algorithm (for Gaussian Process Mixture Model) - EM/GPMM**.

Some rows of the **data** available for this problem are present in Figure 1 on page 3:

	Expression level at t_1	Expression level at t_2	...	Expression level at t_{30}
y_1	-0.09900893	0.2818237	...	0.1033819
y_2	-0.00857957	0.1534053	...	0.08532405
y_3	-0.03709729	-0.00954268	...	0.01441636
...	...	...	...	...

t_1	t_2	...	t_{30}
0.0000000	0.2166616	...	6.2831853

Figure 1: The data we work with

3 Models and Algorithms

In this section, we will present independently the models and algorithms that we used to solve the problem. The ideas that are mentioned are on applicable formulas and simple examples. No proofs (of correctness, convergence etc.) are given.

3.1 Optimization

The optimization problem that we consider is as follows:

Given a differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, find $\max_{x\in\mathbb{R}^n} f(x)$ or $\min_{x\in\mathbb{R}^n} f(x)$.

There is a difference between an optimal solution (or optimum) and an optimal value:

- **optimal solution** $= \text{argmax}_{x\in\mathbb{R}^n} f(x)$
- **optimal value** $= \max_{x\in\mathbb{R}^n} f(x)$

A **local optimal solution** is a solution that is optimal within a neighbourhood centered at that solution, e.g. an interval centered at that solution if $n = 1$.

A **global optimal solution** is a solution that is optimal on the entire domain of the function, i.e. $\mathbb{R}^n$.

3.1.1 Mathematical optimization: Derivatives

It is known that:

If $f : \mathbb{R} \to \mathbb{R}$ is differentiable and $f'(x^*) = 0$ then x^* is a local or global optimal solution or not (if so, it is called to be a **saddle point**). Moreover, if f is **concave or convex**, then x^* is a global optimal solution (max or min, respectively).

Example:

$f(x) = x^2 - 4x + 3$

$f'(x) = 2x - 4$

$f'(x^*) = 0 \Rightarrow 2x^* - 4 = 0 \Rightarrow x^* = 2$
It is known that f is convex, so x^* is a minimum.
optimal solution: $x^* = 2$
optimal value: $f(x^*) = (x^*)^2 - 4x^* + 3 = 2^2 - 4 \cdot 2 + 3 = -1$

If $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable and $\frac{\partial f}{\partial v}(v^*) = \mathbf{0}$ then v^* is a local or global optimal solution or not (if so, it is called to be a **saddle point**). Moreover, if f is **concave or convex**, then v^* is a global optimal solution (max or min, respectively).

Example:
$f(v) = f(x, y) = x^2 + y^2, v = (x, y)$
$\frac{\partial f}{\partial x} = 2x$
$\frac{\partial f}{\partial y} = 2y$
$\frac{\partial f}{\partial v} = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}) = (2x, 2y)$
$\frac{\partial f}{\partial v}(v^*) = \mathbf{0} \Rightarrow \frac{\partial f}{\partial x}(x^*) = 0$ and $\frac{\partial f}{\partial y}(y^*) = 0 \Rightarrow 2x^* = 0$ and $2y^* = 0 \Rightarrow x^* = 0$ and $y^* = 0 \Rightarrow (x^*, y^*) = (0, 0) \Rightarrow v^* = (0, 0)$
It is known that f is convex, so v^* is a minimum.
optimal solution: $v^* = (x^*, y^*) = (0, 0)$
optimal value: $f(v^*) = f(x^*, y^*) = (x^*)^2 + (y^*)^2 = 0$

More on derivatives, gradients, checking the convexity/concavity of a function: [4], [5].

3.1.2 Numerical optimization: Gradient ascent/descent

It is known that:

If $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable and convex, then the following algorithm converges to the global minimum:

Result: x^*
$x^* = x_0$;
while *1* **do**
  $x^*_{new} = x^* - \alpha \cdot \frac{\partial f}{\partial x}(x^*)$;
  if $|f(x^*_{new}) - f(x^*)| < \epsilon$ **then**
    break;
  end
  $x^* = x^*_{new}$;
end

Algorithm 1: Gradient descent

If $f : \mathbb{R}^n \to \mathbb{R}$ is differentiable and concave, then the following algorithm

converges to the global maximum:

Result: x^*
$x^* = x_0$;
while *1* **do**
  $x^*_{new} = x^* + \alpha \cdot \frac{\partial f}{\partial x}(x^*)$;
  if $|f(x^*_{new} - f(x^*)| < \epsilon$ **then**
    break;
  end
  $x^* = x^*_{new}$;
end

Algorithm 2: Gradient ascent

We mention that x_0 (initial solution), α (learning rate) and ϵ are parameters to be given to the algorithms. The learning rate α should be relatively small in order to make the algorithm to converge. The condition $|f(x^*_{new}) - f(x^*)| < \epsilon$ is the numerical version of $f(x^*_{new}) == f(x^*)$.

There are extensions of this algorithm and other (more powerful) numerical optimization methods, but we chose to work only with this version because it is very simple.

Moreover, if the function f is not convex/concave and if $\frac{\partial f}{\partial x}(x_0) \neq \mathbf{0}$ then we can also apply one of the algorithms to find a value $f(x^*) > f(x_0)$ or $f(x^*) < f(x_0)$.

One can use one of these algorithms when solving $\frac{\partial f}{\partial x}(x) = \mathbf{0}$ is difficult (if it is easy, then the mathematical approach is more appropiate: efficient and reliable).

Example:
$f(x) = x^2 - 4x + 3$
$f'(x) = 2x - 4$

It is known that f is convex, so we will use gradient descent. We took $x_0 = 0$, $\alpha = 0.001$, $\epsilon = 1e - 20$.

Result: x^*
$x^* = x_0$;
while *1* **do**
  $x^*_{new} = x^* - \alpha \cdot (2x^* - 4)$;
  if $|f(x^*_{new}) - f(x^*)| < \epsilon$ **then**
    break;
  end
  $x^* = x^*_{new}$;
end

Algorithm 3: Gradient descent

optimal solution: $x^* = 2$
optimal value: $f(x^*) = (x^*)^2 - 4x^* + 3 = 2^2 - 4 \cdot 2 + 3 = -1$

Observations:

- if $\alpha = 1$, then the algorithm stops immediately, because $f(x^*) = f(x^*_{new}) = 3$.

- if $\epsilon = 1e-5$, then $x^* = 1.950041$ and $f(x^*) = -0.9975041$.

Example:
$f(v) = f(x, y) = x^2 + y^2, v = (x, y)$
$\frac{\partial f}{\partial x} = 2x$
$\frac{\partial f}{\partial y} = 2y$
$\frac{\partial f}{\partial v} = (\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}) = (2x, 2y)$
It is known that f is convex, so we will use gradient descent. We took $v_0 = (x_0, y_0) = (1, 1)$, $\alpha = 0.001$ and $\epsilon = 1e-100$.

Result: (x^*, y^*)
$(x^*, y^*) = (x_0, y_0)$;
while *1* **do**
  $(x^*_{new}, y^*_{new}) = (x^*, y^*) - \alpha \cdot (2x^*, 2y^*)$;
  if $|f(x^*_{new}, y^*_{new}) - f(x^*, y^*)| < \epsilon$ **then**
    break;
  end
  $(x^*, y^*) = (x^*_{new}, y^*_{new})$;
end

Algorithm 4: Gradient descent

optimal solution: $(x^*, y^*) = (1.11842e-49, 1.11842e-49)$
optimal value: $f(v^*) = f(x^*, y^*) = (x^*)^2 + (y^*)^2 = 2.501728e-98$
Observations:

- sometimes, we do not get the exact value of the optimal solution

More on gradient descent/ascent, variations of it and other numerical optimization methods: [6], [7].

3.2 Regression

Machine learning can be divided into two main parts: supervised learning and unsupervised learning. Supervised learning is when the instances (table rows) have an output label. Unsupervised learning is when the instances (table rows) do not have an output label.

In supervised learning, for a given **model** (a function that maps the inputs into outputs), we often talk about two algorithms:

- **training** algorithm: this algorithm needs training data; the model is fitted to the training data
- **testing** algorithm: this algorithm needs testing data; the trained/fitted model is tested on new data and the accuracy is reported

A supervised learning task is **regression**, when the output label is not from a finite set, i.e. is any real number.

Usually, a regression task is modelled as follows:

Let $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$ be a training set of **i.i.d.** (independent and identically distributed) samples from some unknown distribution.

$y^{(i)} = f(x^{(i)}) + \epsilon^{(i)}, i = 1, \ldots, m$ + some assumptions

For example, in classical linear regression the assumptions are:

- $f(x)$ is rewritten as $f(x; \beta) = x \cdot \beta$
- $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$

3.2.1 Gaussian Processes

A **stochastic (random) process** is an indexed collection of random variables. More formally:

$f : \mathcal{X} \to \mathcal{F}(\Omega, \mathbb{R})$ - stochastic process

$\mathcal{X}$ - index set

$\mathcal{F}(\Omega, \mathbb{R})$ - set of random variables

If one thinks more about it, then he realizes that the gaussian process model induces a probability distributiopn over functions if $\mathcal{X}$ is finite or over approximations of functions if $\mathcal{X}$ is infinite (one approximates $\mathcal{X}$ with a finite number of indexes). So we can write:

$\mathcal{X} = \{x_1, \ldots, x_n\}$ or $\mathcal{X} \approx \{x_1, \ldots, x_n\}$

$$\overrightarrow{f} \overset{\text{not.}}{=} \begin{bmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_n) \end{bmatrix}$$

$$v \overset{\text{not.}}{=} \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

$p_{\overrightarrow{f}}(v) = p_{f(x_1),\ldots,f(x_n)}(v_1, \ldots, v_n)$, where p is a probability mass function (pmf) or a probability density function (pdf). In the case of a pmf we have: $p_{\overrightarrow{f}}(v) = P(\overrightarrow{f} = v) = P(f(x_1) = v_1, \ldots, f(x_n) = v_n)$.

A **gaussian process** is a stochastic process with the following property: $\forall x_1, \ldots, x_n \in \mathcal{X} : (f(x_1), \ldots, f(x_n))$ has a multivariate normal distribution. So, the above p will be a probability density function (pdf). In the following figures (Figure 2 on page 8; Figure 3 on page 8), we have plotted some **functions** or approximations of functions (**red points**) with indexes from the interval [-1,10]. The **black line** is the **mean function**. The **green lines** represent **the deviations from the mean function** ($\pm 2 \cdot$standard deviation at each point). One can notice that the mean and the deviations do not change, but, as we will see later, these will become important when we talk about Gaussian Process

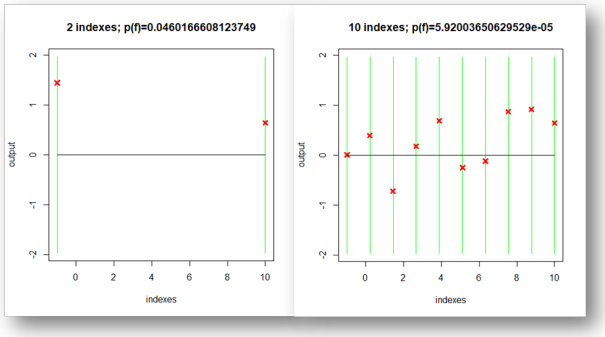

Figure 2: 2 sample functions from the a GP

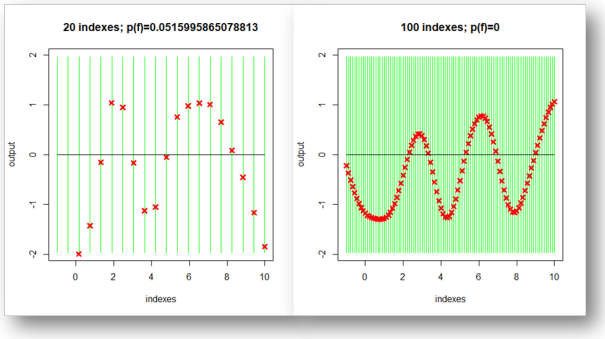

Figure 3: Other 2 sample functions from the a GP

Regression (then they will change). The number of considered indexes and the value of p for the plotted function are written above each plot.

One can notice that, as the number of indexes increases, a continuous function is approximated better and better.

The gaussian process can be parameterized as follows:

Let $S = \{(x^{(i)}, y^{(i)})\}_{i=1}^{m}$ be a training set of i.i.d. samples from some unknown distribution.

$f(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot))$

$m(x) = E[f(x)]$ - **mean function**

$k(x, x') = E[(f(x) - m(x))(f(x') - m(x'))]$ - **covariance function**

$\forall x_1, \ldots, x_m \in \mathcal{X}$ (i.e. for any finite set of indexes):

$$\begin{bmatrix} f(x_1) \\ \vdots \\ f(x_m) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(x_1) \\ \vdots \\ m(x_m) \end{bmatrix}, \begin{bmatrix} k(x_1, x_1) & \ldots & k(x_1, x_m) \\ \vdots & \ddots & \vdots \\ k(x_m, x_1) & \ldots & k(x_m, x_m) \end{bmatrix} \right)$$

Example:

$\mathcal{X} = \mathbb{R}$

$f(\cdot) \sim \mathcal{GP}(m(\cdot), k(\cdot, \cdot))$

$m(x) = 0, \forall x \in \mathcal{X}$

$k(x, x') = e^{-\|x - x'\|^2}$

$1, 2, 3 \in \mathcal{X}$

So:

$$\begin{bmatrix} f(1) \\ f(2) \\ f(3) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} m(1) \\ m(2) \\ m(3) \end{bmatrix}, \begin{bmatrix} k(1,1) & k(1,2) & k(1,3) \\ k(2,1) & k(2,2) & k(2,3) \\ k(3,1) & k(3,2) & k(3,3) \end{bmatrix} \right)$$

$$\begin{bmatrix} f(1) \\ f(2) \\ f(3) \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1.000000 & 0.3678794 & 0.01831564 \\ 0.36787944 & 1.0000000 & 0.36787944 \\ 0.01831564 & 0.3678794 & 1.000000 \end{bmatrix} \right)$$

Let v be a vector:

$$v = \begin{bmatrix} 0.461789 \\ 0.5474731 \\ 0.5863273 \end{bmatrix}$$

$p_{\vec{f}}(v) = p_{f(1),f(2),f(3)}(0.461789, 0.5474731, 0.5863273) = 0.05528701$ (the result was computed by evaluating the pdf of the multivariate normal distribution at the point indicated by v)

3.2.2 Gaussian Process Regression

We restate that a regression task can be modelled as follows:

$y^{(i)} = f(x^{(i)}) + \epsilon^{(i)}, i = 1, \ldots, m$ + some assumptions

The assumptions of Gaussian Process Regression (GPR) are:

- $f(\cdot) \sim \mathcal{GP}(0, k_{ini}(\cdot, \cdot))$

- $\epsilon^{(i)} \sim \mathcal{N}(0, \sigma^2)$

or we can combine both into one assumption if the data we work with does not have duplicates in terms of input ($x^{(i)}$).

- $f(\cdot) \sim \mathcal{GP}(0, k(\cdot, \cdot))$ where:

$$k(x, x') = k_{ini}(x, x') + \sigma_n^2 \delta(x, x') \text{ and } \delta(x, x') = \begin{cases} 1 & x = y \\ 0 & x \neq y \end{cases}$$

The mean function does not have to meet any constraints.

The covariance function must be a **kernel** function with the meaning from the field of kernelized machine learning or kernel methods. An example of such a method is Support Vector Machines (SVM).

- **Training**: in the same time with testing
- **Testing**: Let $T = \{(x_*^{(i)}, y_*^{(i)})\}_{i=1}^m$ be a set of i.i.d. testing points drawn from the same unknown distribution as the training set. The following notations are from [3].

$$X = \begin{bmatrix} -(x^{(1)})^T- \\ \vdots \\ -(x^{(m)})^T- \end{bmatrix} \in \mathbb{R}^{m \times n}; \overrightarrow{f} = \begin{bmatrix} f(x^{(1)}) \\ \vdots \\ f(x^{(m)}) \end{bmatrix}, \overrightarrow{\epsilon} = \begin{bmatrix} \epsilon^{(1)} \\ \vdots \\ \epsilon^{(m)} \end{bmatrix}, \overrightarrow{y} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(m)} \end{bmatrix} \in \mathbb{R}^m;$$

$$X_* = \begin{bmatrix} -(x_*^{(1)})^T- \\ \vdots \\ -(x_*^{(m)})^T- \end{bmatrix} \in \mathbb{R}^{m_* \times n}; \overrightarrow{f_*} = \begin{bmatrix} f(x_*^{(1)}) \\ \vdots \\ f(x_*^{(m)}) \end{bmatrix}, \overrightarrow{\epsilon_*} = \begin{bmatrix} \epsilon_*^{(1)} \\ \vdots \\ \epsilon_*^{(m)} \end{bmatrix}, \overrightarrow{y_*} = \begin{bmatrix} y_*^{(1)} \\ \vdots \\ y_*^{(m)} \end{bmatrix} \in \mathbb{R}^{m_*}$$

$$\begin{bmatrix} \overrightarrow{f} \\ \overrightarrow{f_*} \end{bmatrix} | X, X_* \sim \mathcal{N}\left(\overrightarrow{0}, \begin{bmatrix} K_{ini}(X, X) & K_{ini}(X, X_*) \\ K_{ini}(X_*, X) & K_{ini}(X_*, X_*) \end{bmatrix}\right)$$

$\overrightarrow{f} \in \mathbb{R}^m$ such that $\overrightarrow{f} = [f(x^{(1)}) \dots f(x^{(m)})]^T$

$\overrightarrow{f_*} \in \mathbb{R}^m$ such that $\overrightarrow{f_*} = [f(x_*^{(1)}) \dots f(x_*^{(m)})]^T$

$K_{ini}(X, X) \in \mathbb{R}^{m \times m}$ such that $(K_{ini}(X, X))_{ij} = k_{ini}(x^{(i)}, x^{(j)})$

$K_{ini}(X, X_*) \in \mathbb{R}^{m \times m_*}$ such that $(K_{ini}(X, X_*))_{ij} = k_{ini}(x^{(i)}, x_*^{(j)})$

$K_{ini}(X_*, X) \in \mathbb{R}^{m_* \times m}$ such that $(K_{ini}(X_*, X))_{ij} = k_{ini}(x_*^{(i)}, x^{(j)})$

$K_{ini}(X_*, X_*) \in \mathbb{R}^{m_* \times m_*}$ such that $(K_{ini}(X_*, X_*))_{ij} = k_{ini}(x_*^{(i)}, x_*^{(j)})$

$$\begin{bmatrix}\overrightarrow{\epsilon} \\ \overrightarrow{\epsilon_*}\end{bmatrix} \sim \mathcal{N}\left(\overrightarrow{0}, \begin{bmatrix}\sigma^2 I & \overrightarrow{0} \\ \overrightarrow{0}^T & \sigma^2 I\end{bmatrix}\right)$$

From $y^{(i)} = f(x^{(i)}) + \epsilon^{(i)}, i = 1, \ldots, m$ it follows:

$$\begin{bmatrix}\overrightarrow{y} \\ \overrightarrow{y_*}\end{bmatrix} | X, X_* = \begin{bmatrix}\overrightarrow{f} \\ \overrightarrow{f_*}\end{bmatrix} + \begin{bmatrix}\overrightarrow{\epsilon} \\ \overrightarrow{\epsilon_*}\end{bmatrix} \sim \mathcal{N}\left(\overrightarrow{0}, \begin{bmatrix}K_{ini}(X,X) + \sigma^2 I & K_{ini}(X, X_*) \\ K_{ini}(X_*, X) & K_{ini}(X_*, X_*) + \sigma^2 I\end{bmatrix}\right)$$

$$\overrightarrow{y_*} | \overrightarrow{y}, X, X_* \sim \mathcal{N}(\mu^*, \Sigma^*)$$

where

$$\mu^* = K_{ini}(X_*, X)(K_{ini}(X, X) + \sigma^2 I)^{-1}\overrightarrow{y}$$

$$\Sigma^* = K_{ini}(X_*, X_*) + \sigma^2 I - K_{ini}(X_*, X)(K_{ini}(X, X) + \sigma^2 I)^{-1} K_{ini}(X, X_*)$$

The red part is, in fact, the testing algorithm. We can rewrite it as follows if the data we work with does not have duplicates in terms of input ($x^{(i)}$):

$$\overrightarrow{y_*} | \overrightarrow{y}, X, X_* \sim \mathcal{N}(\mu^*, \Sigma^*)$$

where

$$\mu^* = K_{ini}(X_*, X)K(X, X)^{-1}\overrightarrow{y}$$

$$\Sigma^* = K(X_*, X_*) - K_{ini}(X_*, X)K(X, X)^{-1}K_{ini}(X, X_*)$$

$K(X, X) \in \mathbb{R}^{m \times m}$ such that $(K(X, X))_{ij} = k(x^{(i)}, x^{(j)})$

$K(X_*, X_*) \in \mathbb{R}^{m_* \times m_*}$ such that $(K(X_*, X_*))_{ij} = k(x_*^{(i)}, x_*^{(j)})$

Furthermore, notice that if we are interested in $\overrightarrow{f_*} | \overrightarrow{y}, X, X_* \sim \mathcal{N}(\mu^*, \Sigma^*)$ (**which we actually use when plotting**, as we will see later) then μ^* stays the same and Σ^* changes slightly:

$$\Sigma^* = K_{ini}(X_*, X_*) - K_{ini}(X_*, X)K(X, X)^{-1}K_{ini}(X, X_*)$$

The simplest way of predicting $\overrightarrow{y_*}$ is by using just μ^*.

- **Visualization**
  - **Before regression/training/testing**, i.e. applying the definition of gaussian processes on a large number of indexes (points): we did this in the previous plots; the difference is that now we approximate the function with lines between points. We also place multiple **(sample) functions** on the same plot (the functions that are not black).

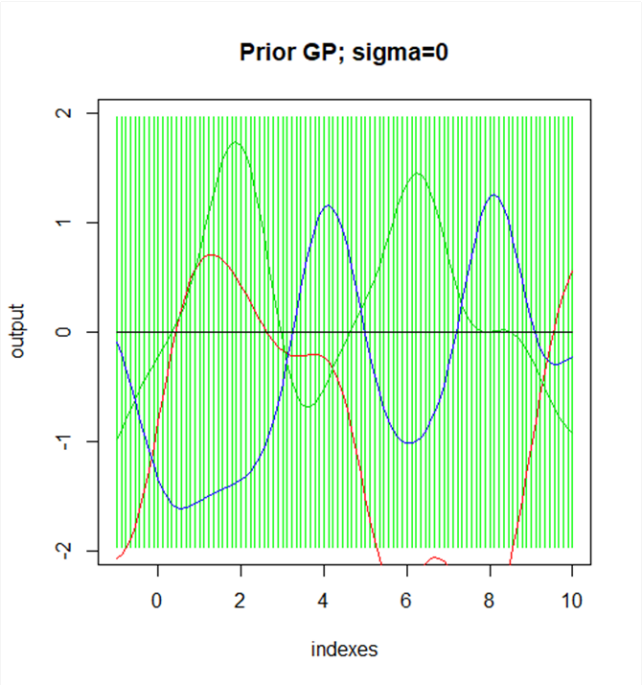

Figure 4: Sample functions from a prior GP using $k(x, x') = 1^2 \cdot e^{-\frac{\|x-x'\|^2}{2 \cdot 1^2}}$

We repeat that the **black line** is the **mean function** and that the **green lines** represent **the deviations from the mean function** (± 2·standard deviation at each point). See Figure 4 on page 12 and Figure 5 on page 13.

$$f(\cdot) \sim \mathcal{GP}(0, k(\cdot, \cdot))$$

An important observation is that the function k determines the shape of functions: in the second plot, the functions are periodic, unlike in the first one.

$f(\cdot) \sim \mathcal{GP}(0, k(\cdot, \cdot))$ or $\overrightarrow{f} \sim \mathcal{N}(\overrightarrow{0}, K)$, where K is the covariance matrix (according to function k) made from any set of indexes, is also called **prior GP**.

– **After regression/training/testing**, i.e. testing on a large number of indexes (points). The meaning of colours is the same as before. See Figure 6 on page 14 and Figure 7 on page 15.

$\overrightarrow{f_*} | \overrightarrow{y}, X, X_* \sim \mathcal{N}(\mu^*, \Sigma^*)$

The red function is a sample function.

In the first plot, we did not use σ in our model (or equivalently, we

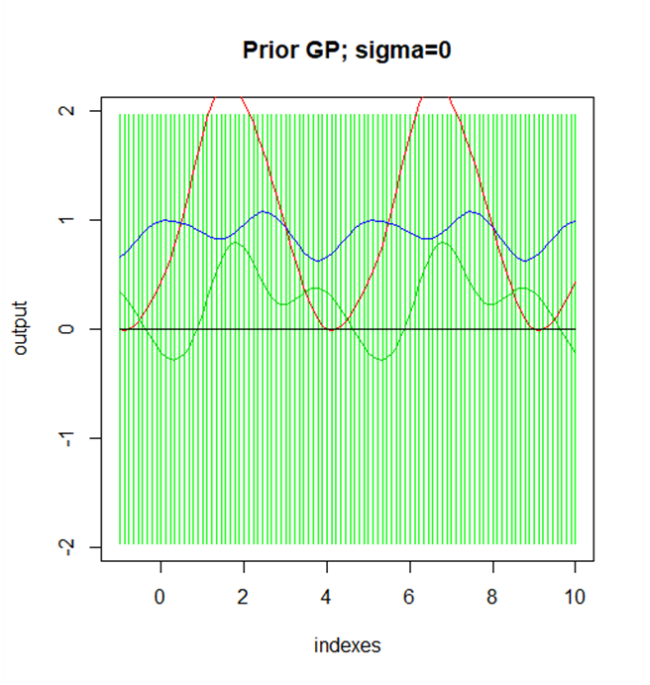

Figure 5: Sample functions from a prior GP using $k(x, x') = 1^2 \cdot e^{-\frac{2}{2^2}\sin^2(\pi\frac{x-x'}{5})}$

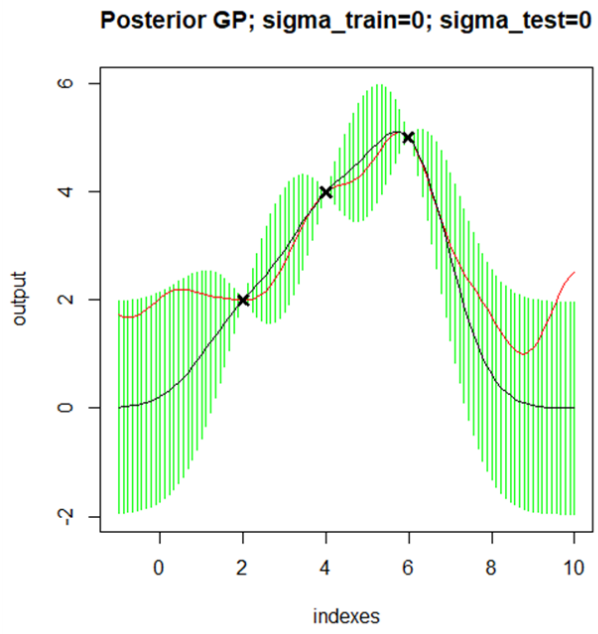

Figure 6: Sample functions from a posterior GP using $k_{ini}(x, x') = 1^2 \cdot e^{-\frac{\|x-x'\|^2}{2 \cdot 1^2}}$: and no σ

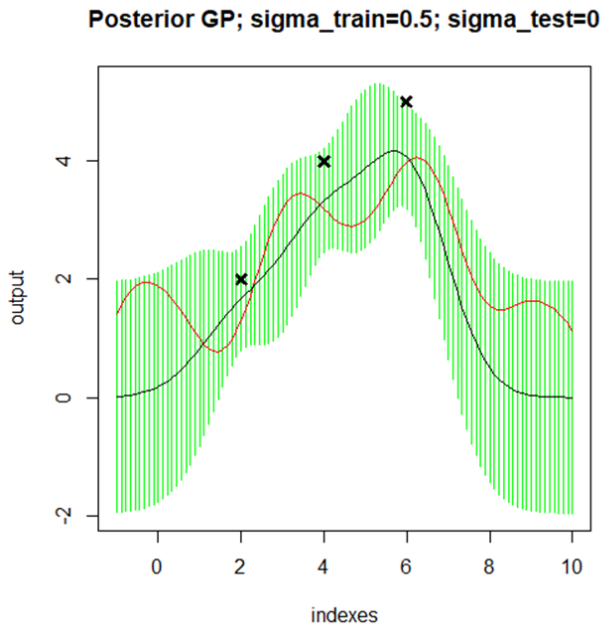

Figure 7: Sample functions from a posterior GP using $k_{ini}(x, x') = 1^2 \cdot e^{-\frac{\|x-x'\|^2}{2 \cdot 1^2}}$ and $\sigma = 0.5$

set σ to 0). The effect is that we do interpolation. In the second plot, σ was set to 0.5.
$\overrightarrow{f_*}|\overrightarrow{y}, X, X_* \sim \mathcal{N}(\mu^*, \Sigma^*)$ is also called **posterior GP**.

Example:
$S = \{(x^{(1)} = 1, y^{(1)} = 0.5), (x^{(2)} = 2, y^{(2)} = 1.1), (x^{(3)} = 2.5, y^{(3)} = 1.5))\}$
$T = \{(x_*^{(1)} = 1.5, y_*^{(1)}), (x_*^{(2)} = 3, y_*^{(2)})\}$
$k_{ini}(x, x') = e^{-\|x-x'\|^2}$
$\sigma = 0.1$
$k(x, x') = e^{-\|x-x'\|^2} + 0.01\delta(x, x')$

$$X = \begin{bmatrix} 1 \\ 2 \\ 2.5 \end{bmatrix}; \overrightarrow{y} = \begin{bmatrix} 0.5 \\ 1.1 \\ 1.5 \end{bmatrix}; X_* = \begin{bmatrix} 1.5 \\ 3 \end{bmatrix}$$

$$\overrightarrow{f_*}|\overrightarrow{y}, X, X_* \sim \mathcal{N}(\mu^*, \Sigma^*)$$

where

$$\mu^* = K_{ini}(X_*, X)(K_{ini}(X, X) + \sigma^2 I)^{-1}\overrightarrow{y}$$
$$\Sigma^* = K_{ini}(X_*, X_*) - K_{ini}(X_*, X)(K_{ini}(X, X) + \sigma^2 I)^{-1}K_{ini}(X, X_*)$$

$$\begin{aligned}
\mu^* &= K_{ini}(X_*, X)(K_{ini}(X, X) + \sigma^2 I)^{-1}\overrightarrow{y} \\
&= \begin{bmatrix} 0.40168643 & 0.4016864 & 0.2569139 \\ 0.04299215 & 0.2569139 & 0.4016864 \end{bmatrix} \cdot \\
&\cdot \left(\begin{bmatrix} 0.4662158 & 0.2569139 & 0.1219799 \\ 0.2569139 & 0.4662158 & 0.4016864 \\ 0.1219799 & 0.4016864 & 0.4662158 \end{bmatrix} + \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \right)^{-1} \begin{bmatrix} 0.5 \\ 1.1 \\ 1.5 \end{bmatrix} \\
&= \begin{bmatrix} 0.7288251 \\ 1.3990457 \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\Sigma^* &= K_{ini}(X_*, X_*) - K_{ini}(X_*, X)(K_{ini}(X, X) + \sigma^2 I)^{-1}K_{ini}(X, X_*) \\
&= \begin{bmatrix} 0.4662158 & 0.1219799 \\ 0.1219799 & 0.4662158 \end{bmatrix} - \begin{bmatrix} 0.40168643 & 0.4016864 & 0.2569139 \\ 0.04299215 & 0.2569139 & 0.4016864 \end{bmatrix} \cdot \\
&\cdot \left(\begin{bmatrix} 0.4662158 & 0.2569139 & 0.1219799 \\ 0.2569139 & 0.4662158 & 0.4016864 \\ 0.1219799 & 0.4016864 & 0.4662158 \end{bmatrix} + \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \right)^{-1} \begin{bmatrix} 0.4016864 & 0.04299215 \\ 0.4016864 & 0.25691388 \\ 0.2569139 & 0.40168643 \end{bmatrix} \\
&= \begin{bmatrix} 0.0179178697 & 0.0004834491 \\ 0.0004834491 & 0.0748235579 \end{bmatrix}
\end{aligned}$$

For visualization, in the following plots (Figure 8 on page 17 and Figure 9 on page 17) the indexes considered were obtained by dividing the interval [0,4] into 99 disjoint intervals of the same length (so there are 100 indexes).

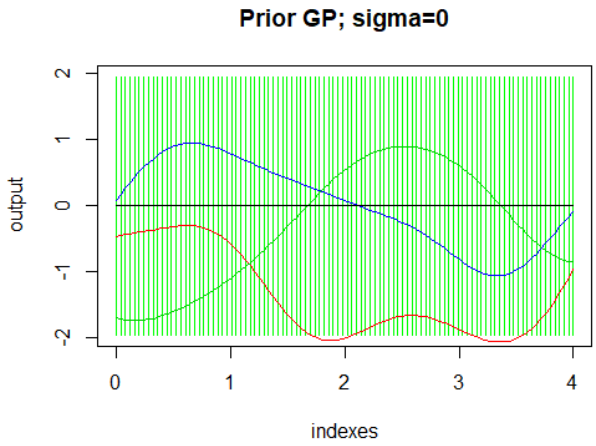

Figure 8: Prior GP example

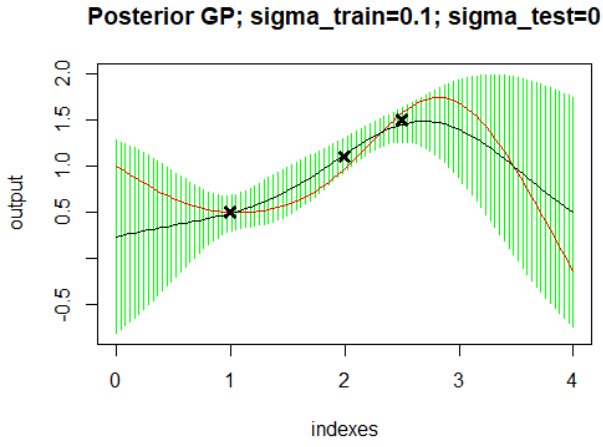

Figure 9: Posterior GP example

More on Gaussian variables properties, Gaussian Processes (and their visualization), matrices, function shapes given by kernel functions: [3], [8], [9], [10], [11].

3.3 Clustering

If regression is a supervised learning task, clustering is an unsupervised learning task. The objective is to group (cluster) instances (data) into groups such that similar instances are in the same group and dissimilar instances are in different groups.

The most popular algorithms that do clustering are: k-means, hierarchical clustering algorithms and EM for probability distribution (usually gaussian) mixture model.

3.3.1 The EM algorithm

As indicated above, the EM algorithm can be used to fit a mixture distribution to the data we work with.

An instance generated from a **probability mixture model** can be described as follows:

- throw a (not necessarily fair) die with k sides and let the result be i
- generate a number (or a vector if the distribution is multivariate) according to the i-th distribution.

The number of distributions (k) must be known beforehand.

In fact, the EM algorithm is a **general algorithmic schema**, i.e. the specific algorithm differs from problem to problem (only the general structure remains the same). It is an **iterative algorithm**: the two iterative steps are E (Expectation) and M (Maximization). It is used at maximizing a function called **log-likelihood of observed data**. As suggested, there is another type of data besides observed data: **latent data**. So, the problem to solve must be modelled in such a way that there are data of both types.

The **likelihood function** is of the form $f(h) = p(D|h)$, where p is a pmf or pdf function, D are the observed data and h is a hypothesis/model, i.e. a true or false affirmation, a machine learning model, or the parameters of a probability distribution etc. This task is considered easier.

An important idea is that the maximization of the log-likelihood of the observed data is considered hard. The EM algorithm achieves this by maximizing (or just increasing the value of) a different function (called Q in the image above) at each iteration. This task is considered easier.

Another important note is that the EM algorithm does not find, in general, the global maximum, but a local one.

Example:
$D = \{1, 2, 3\}$

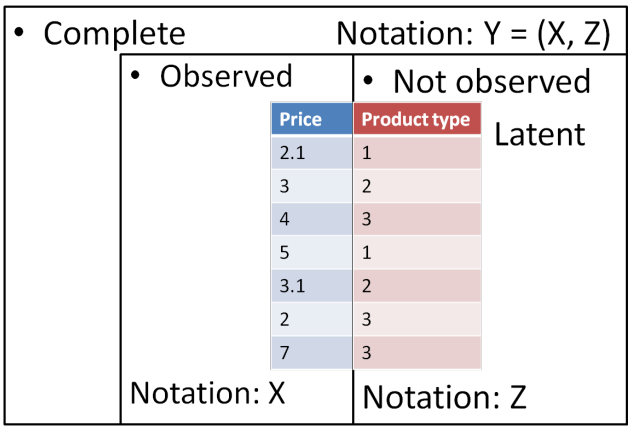

Figure 10: An example of complete, observed and latent data

$h_0 = \{\mu = 0, \sigma = 1\}$ - parameters of a gaussian distribution

$f(h_0) = p(D|h_0) = p(1, 2, 3|\mu = 0, \sigma = 1) \overset{\text{iid}}{=} p(1|\mu = 0, \sigma = 1) \cdot p(2|\mu = 0, \sigma = 1) \cdot p(3|\mu = 0, \sigma = 1) = \ldots = 5.78987e - 05$ (the result was computed by evaluating the pdf of the univariate normal distribution at the points 1, 2 and 3)

The **log-likelihood function** is of the form $f(h) = \log p(D|h)$, where log is the natural logarithm function, and the other three have the same meaning as above. It can be noticed that maximizing the likelihood is equivalent to maximizing the log-likelihood. Why using the log function? Because:

- $p(D|h)$ is a product of numbers in the interval [0,1], so the result will be a very small number; as the computer uses approximations of real numbers, the result could be approximated to 0 even though it is not 0; a more reliable result in such cases would be to compute $\log p(D|h)$ and then $e^{\log p(D|h)}$
- in maximing the likelihood function, derivatives usually are involved; the derivative of a product ($p(D|h)$) is more complicated to compute than the derivative of a sum ($\log p(D|h)$)
- for computers, multiplication is slower than addition

An example of observed data and latent data is given in Figure 10 on page 19 (when taken together they are called **complete data**).

The log-likelihood function can be optimized using other numerical methods, but usually it is optimized using the EM algorithm.

A high-level view of the schema is given in Figure 11 on page 20.

Example:

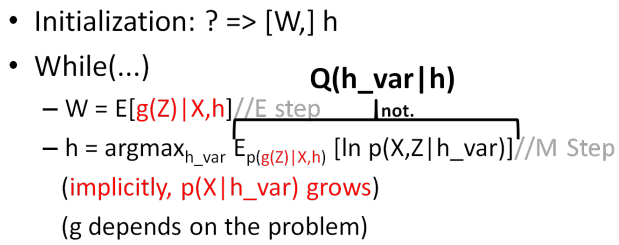

Figure 11: EM schema

Consider the case of Gaussian Mixture Model (**EM/GMM**). We will not derive the E and M steps here, but we will give the final formulas and apply them on a small dataset.

The data we work with is: $D = \{-9, -8, -7, -6, -5, 5, 5, 6, 6, 7, 7, 8, 8, 9, 9\}$

The iterative steps are (πs are the selection probabilities - the probabilities of getting a certain die side, i.e. a certain gaussian distribution -, μs are the mean parameters of the gaussian distributions, σs are the standard deviations of the gaussian distributions).

E step:

$$\gamma_{ij}^{(t+1)} \overset{\text{not.}}{=} \frac{\pi_j^{(t)} \mathcal{N}(x_i; \mu_j^{(t)}, (\sigma_j^2)^{(t)})}{\sum_{l=1}^{k} \pi_l^{(t)} \mathcal{N}(x_i; \mu_l^{(t)}, (\sigma_l^2)^{(t)})}$$

M step:

$$\pi_j^{(t+1)} = \frac{\sum_{i=1}^{n} \gamma_{ij}^{(t+1)}}{n}$$

$$\mu_j^{(t+1)} = \frac{\sum_{i=1}^{n} \gamma_{ij}^{(t+1)} x_i}{\sum_{i=1}^{n} \gamma_{ij}^{(t+1)}}$$

$$(\sigma_j^2)^{(t+1)} = \frac{\sum_{i=1}^{n} \gamma_{ij}^{(t+1)} (x_i - \mu_j^{(t+1)})^2}{\sum_{i=1}^{n} \gamma_{ij}^{(t+1)}}$$

We take $k = 2$.

Initialisation:

$\pi_1^{(0)} = 0.5$ and $\pi_2^{(0)} = 0.5$

$\mu_1^{(0)} = -20$ and $\mu_2^{(0)} = -10$

$\sigma_1^{(0)} = \sqrt{20}$ and $\sigma_2^{(0)} = \sqrt{4}$

The log-likelihood of the observed data is: -232.1261.

After iteration 1:

$$\gamma^{(1)} = \begin{bmatrix} 0.02401616 & 9.759838e-01 \\ 0.01974875 & 9.802513e-01 \\ 0.01974875 & 9.802513e-01 \\ 0.02401616 & 9.759838e-01 \\ 0.03540966 & 9.645903e-01 \\ 0.99999167 & 8.332980e-06 \\ 0.99999167 & 8.332980e-06 \\ 0.99999938 & 6.189250e-07 \\ 0.99999938 & 6.189250e-07 \\ 0.99999996 & 3.763689e-08 \\ 0.99999996 & 3.763689e-08 \\ 1.00000000 & 1.873831e-09 \\ 1.00000000 & 1.873831e-09 \\ 1.00000000 & 7.638147e-11 \\ 1.00000000 & 7.638147e-11 \end{bmatrix}$$

$\pi_1^{(1)} = 0.6748614$ and $\pi_2^{(1)} = 0.3251386$
$\mu_1^{(1)} = 6.832651$ and $\mu_2^{(1)} = -7.005503$
$\sigma_1^{(1)} = 2.069065$ and $\sigma_2^{(1)} = 1.411809$
The log-likelihood of the observed data is: -37.20656.
After iteration 2:

$$\gamma^{(2)} = \begin{bmatrix} 7.406501e-13 & 1.000000e+00 \\ 1.257249e-11 & 1.000000e+00 \\ 2.790427e-10 & 1.000000e+00 \\ 8.097699e-09 & 1.000000e+00 \\ 3.072514e-07 & 9.999997e-01 \\ 1.000000e+00 & 2.074463e-16 \\ 1.000000e+00 & 2.074463e-16 \\ 1.000000e+00 & 2.863638e-19 \\ 1.000000e+00 & 2.863638e-19 \\ 1.000000e+00 & 3.023353e-22 \\ 1.000000e+00 & 3.023353e-22 \\ 1.000000e+00 & 2.441284e-25 \\ 1.000000e+00 & 2.441284e-25 \\ 1.000000e+00 & 1.507670e-28 \\ 1.000000e+00 & 1.507670e-28 \end{bmatrix}$$

$\pi_1^{(2)} = 0.6666667$ and $\pi_2^{(2)} = 0.3333333$
$\mu_1^{(2)} = 7$ and $\mu_2^{(2)} = -7$
$\sigma_1^{(2)} = 1.414215$ and $\sigma_2^{(2)} = 1.414214$
The log-likelihood of the observed data is: -36.03039.
After iteration 3:

$$\gamma^{(3)} = \begin{bmatrix} 8.720493e-28 & 1.000000e+00 \\ 9.563014e-25 & 1.000000e+00 \\ 1.048695e-21 & 1.000000e+00 \\ 1.150016e-18 & 1.000000e+00 \\ 1.261127e-15 & 1.000000e+00 \\ 1.000000e+00 & 3.152544e-16 \\ 1.000000e+00 & 3.152544e-16 \\ 1.000000e+00 & 2.874752e-19 \\ 1.000000e+00 & 2.874752e-19 \\ 1.000000e+00 & 2.621436e-22 \\ 1.000000e+00 & 2.621436e-22 \\ 1.000000e+00 & 2.390438e-25 \\ 1.000000e+00 & 2.390438e-25 \\ 1.000000e+00 & 2.179793e-28 \\ 1.000000e+00 & 2.179793e-28 \end{bmatrix}$$

$\pi_1^{(3)} = 0.6666667$ and $\pi_2^{(2)} = 0.3333333$
$\mu_1^{(3)} = 7$ and $\mu_2^{(2)} = -7$
$\sigma_1^{(3)} = 1.414215$ and $\sigma_2^{(2)} = 1.414214$
The log-likelihood of the observed data is: -36.03039.

Because there is just a slight change (although not visible) in the values of parameters, the algorithm is stopped.

As expected, the values of the log-likelihood of the observed data increase iteration by iteration. As noted earlier, perhaps the value -36.03039 is not the optimal value, so the algorithm should be restarted with a different initialization multiple times and, in the end, one should retain the highest value of the log-likelihood function.

The formulas from the M step are derived from maximizing the function Q. This solution is a closed-form solution, but, as we will see later, this is not always the case.

If one wants to perform clustering with the EM algorithm and takes into consideration the probabilities from the matrix γ then the clustering is called **soft clustering**. If we assign each instance to the cluster that has the highest probability of containing that instance, then the clustering is called **hard clustering**. In our example, looking at $\gamma^{(3)}$ the clusters given by hard-clustering are: $C_1 = \{-9, -8, -7, -6, -5\}, C_2 = \{5, 5, 6, 6, 7, 7, 8, 8, 9, 9\}$.

More on the EM algorithm: [12], [13], [14].

3.3.2 The EM algorithm for Gaussian Process Mixture Model (EM/GPMM)

As a difference from the previous example, now we will work with a Gaussian Process Mixture, not a Gaussian Mixture.

We will not derive the E and M steps here, but we will give the final formulas. We used a certain type of kernel, called The **Squared exponential kernel (or**

The Gaussian RBF kernel).

The hypothesis h_j of a certain gaussian process is given by the parameters $(\sigma_f)_j, \rho_j, (\sigma_n)_j$.

$$k_{ini}(t,t') = \sigma_f^2 \exp\left(-\frac{(t-t')^2}{2\rho^2}\right)$$

$$k(t,t') = \sigma_f^2 \exp\left(-\frac{(t-t')^2}{2\rho^2}\right) + \sigma_n^2 \delta(t,t')$$

$$k_j^{(t)}(t,t') = ((\sigma_f)_j^{(t)})^2 \exp\left(-\frac{(t-t')^2}{2((\rho)_j^{(t)})^2}\right) + ((\sigma_n)_j^{(t)})^2 \delta(t,t')$$

$K_j^{(t)} \overset{\text{not.}}{=} K(h_j^{(t)}) \overset{\text{not.}}{=} K_j^{(t)}(X,X)$, where X is a matrix representing the training set, as discussed earlier

E step:

$$\gamma_{ij}^{(t+1)} \overset{\text{not.}}{=} \frac{\pi_j^{(t)} \mathcal{N}(x_i; 0, K_j^{(t)}))}{\sum_{l=1}^{k} \pi_l^{(t)} \mathcal{N}(x_i; 0, K_l^{(t)}))}$$

M step:

$$\pi_j^{(t+1)} = \frac{\sum_{i=1}^{n} \gamma_{ij}^{(t+1)}}{n}$$

$$\theta_l \in \{(\sigma_f)_l, (\sigma_n)_l, \rho_l\}$$

$$\frac{\partial Q}{\partial \theta_l} = \frac{1}{2} \sum_{i=1}^{n} \left(\gamma_{il}^{(t+1)} x_i^T (K_l^{(t)})^{-1} \left(\frac{\partial K_l}{\theta_l}\right)^{(t)} (K_l^{(t)})^{-1} x_i \right) - \frac{\sum_{i=1}^{n} \gamma_{il}^{(t+1)}}{2} \text{Tr}\left((K_l^{(t)})^{-1} \left(\frac{\partial K_l}{\theta_l}\right)^{(t)} \right)$$

$$\frac{\partial k(x,y)}{\partial \sigma_f} = 2\sigma_f \exp\left(-\frac{(x-y)^2}{2\rho^2}\right)$$

$$\frac{\partial k(x,y)}{\partial \sigma_n} = 2\sigma_n \delta(x,y)$$

$$\frac{\partial k(x,y)}{\partial \rho} = \sigma_f^2 \exp\left(-\frac{(x-y)^2}{2\rho^2}\right) \frac{(x-y)^2}{\rho^3}$$

To find $\theta_l^{(t+1)}$ we used the gradient ascent method.

An important note is that the maximization of the Q function does not give rise to a closed-form solution as in the case of EM/GMM. A numerical method must be used in order to increase the function Q (note: here, we just locally maximize the function Q and not necessarily globally maximize it; this works because the function Q does not need to be globally maximized, but just increased from the last iteration).

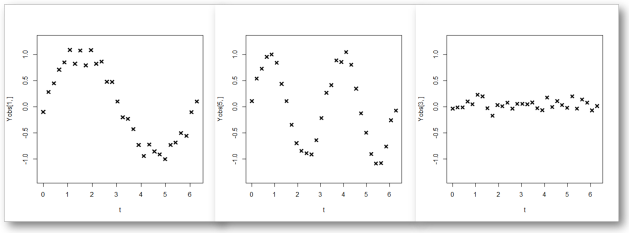

Figure 12: The three types of expression curves in our data

4 Solving the problem

Given the information from the previous section solving the problem becomes easy. In out dataset, there are 90 genes. The expression curves of all 90 genes follow one of the three patternsin Figure 12 on page 24.

We would like to cluster the 90 genes according to the 30 gene expression levels (we cluster the vectors $y_1, \ldots, y_{90}$) and then, in each cluster, we perform regression in a way that includes all the instances from that cluster (we perform regression independently on each y_j, but the parameters we use for regression are the same for a certain cluster).

To do this, we use the algorithm EM/GPMM from the previous subsection, which does exactly what we need.

The **halting condition** imposed was:

$$|\text{log-likelihood}_{\text{obs}}^{(t+1)} - \text{log-likelihood}_{\text{obs}}^{(t)}| < \epsilon.$$

4.1 Alternatives or Why using EM/GPMM?

In general, when using EM for clustering what we care more about are the membership probabilities of each instance to every cluster (the output of the E step).

In this problem, we also care about the parameters of each gaussian process (the output of the M step), because with them we carry out regression. Given the expression levels of a gene at $t_1, \ldots, t_{30}$, we find the cluster this instance would belong to (we look at the cluster that has the highest probability of containing this instance - hard clustering) and we carry out Gaussian Process Regression on this data with the parameters corresponding to the cluster we discovered this instance belongs to.

In other words, by looking at the output of the E step, we perfom clustering and by looking at the ouput of the M step, we perform regression. So, the two tasks can be performed quite naturally with EM/GPMM and are interdependent (in the while loop, the output of the E step goes as input for the M step).

A natural question would be why not using just EM for (Multivariate) Gaussian Mixture Model instead of EM/GPMM. The answer is that the output from

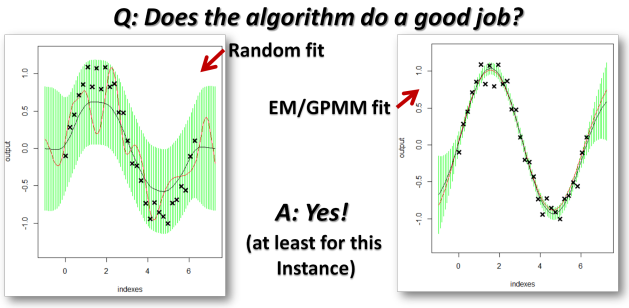

Figure 13: Random fit vs. EM/GPMM fit

the M step of the EM/GMM algorithm does not help us in carrying out regression.

Simple **alternatives** to this approach involve discarding the idea of doing both tasks (clustering and regression) at the same time with a single algorithm: perform a hard-clustering (using hierarchical clustering algorithms or k-means or EM/GMM) and then, in each cluster, fit the parameters of a gaussian process using the principle of maximum likelihood estimation (MLE).

4.2 Results

A natural question that comes to mind is whether the solution makes the regression task (which was our main goal) better in some way or not. As seen in Figure 13 on page 25, the deviations from the mean are decreased when working with EM/GPMM and so we have more confidence in what the values on the real expression curve are.

For a given number of clusters k, we ran for multiple times the algorithm and, in the end, took only the result with the best (highest) value of the log-likelihood of the observed data.

The values of k we tried were: 2, 3, 4, 5.

We visually present the results in **two versions** (Figure 14 on page 26; Figure 15 on page 26; Figure 16 on page 27; Figure 17 on page 27). In both versions, we have a plot for each cluster. In the first version, what is plotted is one sample function per type of curve. In the second version, all the mean functions resulted from performing GP Regression on a cluster are put on the same plot.

It can be noticed that when $k = 2$ then a cluster contains only a type of curves, and the other one contains two types of curves. When $k = 3$, the result is the natural one knowing that our data contain only 3 expression curves. When $k = 4$, a natural cluster is split into two. When $k = 5$, a natural cluster is split into two and a cluster is empty.

We repeat ourselves saying that those results were obtained after running the algorithm **multiple** times for a given k. Running the algorithm just once

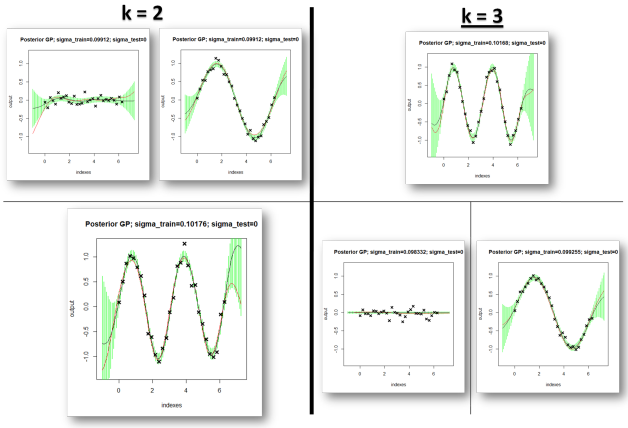

Figure 14: Results: first version, $k \in \{2, 3\}$

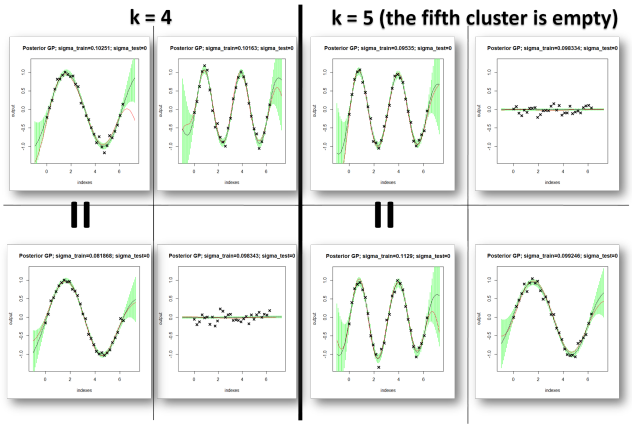

Figure 15: Results: first version, $k \in \{4, 5\}$

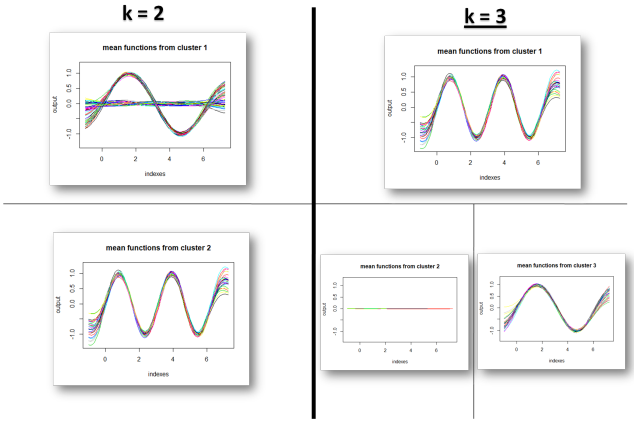

Figure 16: Results: second version, $k \in \{2, 3\}$

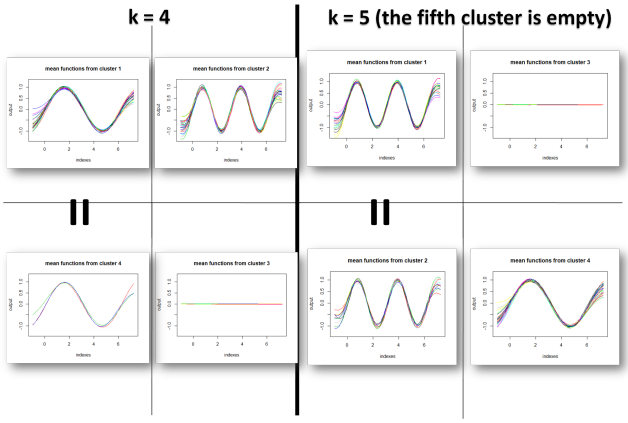

Figure 17: Results: second version, $k \in \{4, 5\}$

does not, in general, give the results from above (including the case when $k = 3$).

4.3 Using the implemented program

The algorithm was implemented in **Matlab**. The visualizations were implemented in **R**. We will present the files with the code that is executed when the algortihm is run or when the plots are made. Being confortable with those files, one could easily modify a parameter or the dataset in order to get its own results.

Rerun the algorithm:
File **my-solution/main.m**:

```
% Reading data
Yobs=importdata('Yobs.dat');
t=importdata('t.dat');

% Introducing the parameters of our algorithm
k = 3; % number of clusters
l_rate = 0.000001; % learning rate for gradient ascent
eps = 1e-6; % used in checking if a value is zero
nIterQ = 1000; % maximum number of iterations for gradient
    ascent
[W,V,L,L_first] = em_gp(t, Yobs, k, l_rate, eps, nIterQ); %
     EM/GPMM
% W - matrix of probabilities
% V - matrix with the GP parameters for each cluster
%     one cluster on a line; on a line: pi, sigma_f, rho,
    sigma_n
% L - the log-likelihood of the observed data at the end of
     the algorithm
% L_first - the log-likelihood of the observed data at the
    beginning of the algorithm

their_L = their_log-likelihood(V,Yobs,t,W);
% the log-likelihood of the observed data computed slightly
     differently using
% the function given in the official solutions of the
    problem

classes = get_classes(W); % perfom hard clustering

% Output the results
if ~exist('output','dir')
mkdir('output');
end

dlmwrite(strcat('output/values_',int2str(k),'.txt'),[
    L_first L their_L],' ');
dlmwrite(strcat('output/classes_',int2str(k),'.txt'),
    classes,' ');
dlmwrite(strcat('output/V_',int2str(k),'.txt'),V,' ');
```

Process (Visualize) results of the algorithm:
File **process-result-in-r/main_samples.R**:

```
# Code to include
```

```
source("kernels.R")
source("functions.R")
source("my-functions.R")

# Reading data
t <- read.delim("t.dat",header = FALSE)[[1]]
Yobs <- read.delim("Yobs.dat",header = FALSE,sep = "")

# The plotted range on vertical axis
ylim_interval <- get_ylim(Yobs)

# Three types of curves:
plot(t,Yobs[1,],ylim=ylim_interval,pch=4,lwd=3)
plot(t,Yobs[5,],ylim=ylim_interval,pch=4,lwd=3)
plot(t,Yobs[3,],ylim=ylim_interval,pch=4,lwd=3)
```

File **process-result-in-r/main_em.R**:

```
# Code to include
source("kernels.R")
source("functions.R")
source("my-functions.R")
library(MASS)

# sigma = std dev from epsilon noise
# sigma = 0 => noise-free

# Number of cluster
k <- 5
# Read the output from Matlab code
vars_from_file <- get_vars_from_file(k = k, their = FALSE)

############ Data ############
t <- read.delim("t.dat",header = FALSE)
Yobs <- read.delim("Yobs.dat",header = FALSE,sep="")

# The plotted range on vertical axis
ylim_interval <- get_ylim(Yobs)

############ Prior GP ############
# Prior GP (with noise); class_index is from {1,...,k}
sample_prior_GP(vars_from_file,t,class_index = 1)

############ Posterior GP ############
# Posterior GP
# index is from {1,...,m=90}
index <- 1
# random fit for the instance #index
posterior_GP_random(index = index,t,Yobs)
# EM/GPMM fit for the instance #index
posterior_GP_from_file(vars_from_file,index = index,t,Yobs)

# Get a random instance index from the cluster #class_index
# random_index is from {1,...,m=90}; class_index is from
    {1,...,k}
random_index <- posterior_GP_from_class(class_index=1,t,
    Yobs)
```

```
############ All cluster means into a plot ############
# class_index is from {1,...,k}
plot_mean_functions_from_cluster(vars_from_file,
    cluster_index = 4)
```

5 Conclusions and future work

We presented an application of the EM algorithm and Gaussian Processes in Bioinformatics. The problem is on gene expression. The solution is using the EM/GPMM algorithm in order to perform simultaneously a clustering task and a regression task. We presented the problem, then the theoretical background needed. Given this theoretical background, we discussed how the problem can be easily solved and then presented the results.

Some important observations not mentioned above, but that have the status of conclusions are:

- the initialization matters a lot in order to converge to a good solution
- the EM algorithm can be initialized with a matrix of probabilities from a given initial (random) hard clustering
- there are multiple similarities between MLE for a distribution (or GP) and the EM algorithm for a distribution (or GP) mixture model.
- at the M step, there are cases when there are no closed-form solutions, so a numerical method should be used
- there are more powerful numerical methods than the gradient (ascent) method

As future work, we could try to compare these results with the results from the simple alternatives we referred to earlier. Moreover, in the literature there are also approaches that do not use Gaussian Processes, but Dirichlet Processes and we could try to implement this approach, too. Furthermore, all the results we obtained or would obtain by these other methods could be tested and compared by us on different datasets. The results could be reported also in a numerical manner, not only visually, i.e. in a way that a number describes the quality of the clustering.

References

[1] Tommi Jaakkola: MIT, 6867 Machine Learning, Fall 2006, HW5, pr. 2
https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-867-machine-learning-fall-2006/assignments/hw5.pdf

[2] Wikipedia. *Bioinformatics.*
https://en.wikipedia.org/wiki/Bioinformatics

[3] Andrew Ng. Stanford, Machine Learning, Fall 2017
http://cs229.stanford.edu/section/cs229-gaussian_processes.pdf

[4] J. Stewart. *Calculus: chapters 2, 3, 14.*

[5] Jim Lambers. USM, MAT 419/519, Summer Session 2011-12, Lecture 6 Notes
http://www.math.usm.edu/lambers/mat419/lecture6.pdf

[6] Sebastian Ruder. *An overview of gradient descent optimization algorithms.*
https://arxiv.org/abs/1609.04747

[7] Mark Schmidt. University of British Columbia, CPSC 540: Machine Learning, Winter 2017
https://www.cs.ubc.ca/ schmidtm/Courses/540-W17/L3.pdf

[8] Richard E. Turner. *An introduction to Gaussian processes for probabilistic inference.*
http://gpss.cc/gpss13/assets/Sheffield-GPSS2013-Turner.pdf

[9] Matrices - Introduction
https://www.mathsisfun.com/algebra/matrix-introduction.html

[10] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning.*
http://www.gaussianprocess.org/gpml/

[11] David Duvenaud. *The Kernel Cookbook: Advice on Covariance functions.*
https://www.cs.toronto.edu/ duvenaud/cookbook/

[12] Chuong B Do and Serafim Batzoglou. *What is the expectation maximization algorithm?.*
https://www.cmi.ac.in/ madhavan/courses/dmml2017/literature/EM_algorithm_2coin_example.pdf

[13] Andrew Ng. Stanford, Machine Learning, Fall 2017
http://cs229.stanford.edu/notes/cs229-notes8.pdf

[14] Liviu Ciortuz. UAIC, Machine Learning, Fall 2017
https://profs.info.uaic.ro/ ciortuz/ML.ex-book/SLIDES/ML.ex-book.SLIDES.Cluster.pdf
https://profs.info.uaic.ro/ ciortuz/ML.ex-book/SLIDES/ML.ex-book.SLIDES.EM.pdf