# CSE 571 - Robotics
# Homework 1 - Bayes Filters and Gaussian Processes

Due Thursday, October 20 at 11:59PM

This homework involves three writing assignments and two programming assignments in MATLAB. In the first programming problem, you will be implementing a 1D Gaussian process for predicting outputs given training data. In the second programming problem, you will be using multiple Gaussian processes to learn the dynamics of a cartpole system based on interactions with a cartpole simulator. The zip file containing the code for this homework can be found on the class website (`https://courses.cs.washington.edu/courses/cse571/16au/`).

*Useful reading material:* Lecture notes, Chapter 2 of Probabilistic Robotics, Thrun, Burgard and Fox (pdf shared with class) and Chapter 2 of Gaussian Processes for Machine Learning, Rasmussen and Williams (Available online at: `http://www.gaussianprocess.org/gpml/chapters/`)

## 1 Writing assignments

### 1.1 Conditional Independence

Let $X$,$Y$ and $Z$ be three random variables. Assuming that $X$ and $Y$ are conditionally independent given $Z$:

$$p(x, y|z) = p(x|z)p(y|z)$$

show that:

$$p(x|z) = p(x|z, y)$$
$$p(y|z) = p(y|z, x)$$

### 1.2 Bayes Filter

A vacuum cleaning robot is equipped with a cleaning unit to clean the floor. The robot has a binary sensor to detect whether a floor tile is clean or dirty. Neither the cleaning unit nor the sensor are perfect.
From previous experience, you know that the robot succeeds in cleaning a dirty floor tile with a probability of

$$p(x_{t+1} = clean|x_t = dirty, u_t = vacuum\_clean) = 0.7,$$

where $x_t$ is the state of a floor tile at time $t$, $u_t$ is the control action applied at $t$ and $x_{t+1}$ is the resulting state after the action has been applied. Also you know that vacuum-cleaning the tile when it is clean does not make it dirty.
The probability that the sensor indicates that the floor tile is clean although it is dirty is given by $p(z = clean|x = dirty) = 0.3$, and the probability that the sensor correctly detects a clean tile is given by $p(z = clean|x = clean) = 0.9$.

Unfortunately, you have no knowledge about the current state of the floor tile. However, after cleaning the tile, the sensor of the robot indicates that it is clean. Compute the probability that the floor tile is clean after the robot has vacuum-cleaned it. Assume a prior distribution for the state at time $t$ as $p(x_t = clean) = c$, where $0 <= c <= 1$.

### 1.3 Gaussian Conditioning

Let $X$ and $Y$ denote two random variables that are jointly Gaussian:

$$\begin{aligned} p(x,y) &= \mathcal{N}(\mu^*, \Sigma) \\ &= \det(2\pi\Sigma)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\left((x,y)^T - \mu^*\right)^T \Sigma^{-1}\left((x,y)^T - \mu^*\right)\right\}, \end{aligned}$$

where $\mu^* = (\mu_x^*, \mu_y^*)^T$ and $\Sigma = \begin{pmatrix} \sigma_x^2 & \sigma_{xy}^2 \\ \sigma_{xy}^2 & \sigma_y^2 \end{pmatrix}$ are the mean and covariance, respectively. Show that conditioning on $Y$ results in a Gaussian over $X$:

$$\begin{aligned} p(x \mid y) &= \mathcal{N}(\mu, \sigma^2) \\ &= \left(2\pi\sigma^2\right)^{-\frac{1}{2}} \exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\} \end{aligned}$$

with $\mu = \mu_x^* + \frac{\sigma_{xy}^2}{\sigma_y^2}(y - \mu_y^*)$ and $\sigma^2 = \sigma_x^2 - \frac{\sigma_{xy}^4}{\sigma_y^2}$.

#### 1.3.1 Hints

- Use the definition of the conditional distribution and "complete the square" to get the answer
- Given $p(x,y)$ is jointly Gaussian, the marginal distribution of $y$ is also Gaussian: $p(y) = \mathcal{N}(\mu_y^*, \sigma_y^2)$
- For a 2x2 matrix positive definite matrix $A = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, $A^{-1} = \frac{1}{|A|}\begin{pmatrix} d & -c \\ -b & a \end{pmatrix}$; $|A| = ad - bc$
- You might want to simplify your expressions using the correlation coefficient ($\rho$), which is defined as: $\rho = \frac{\sigma_{xy}^2}{\sigma_x \sigma_y}$. For example: $\frac{\sigma_y^2}{|\Sigma|} = \frac{1}{\sigma_x^2(1-\rho^2)}$

## 2 Programming problems

### 2.1 Gaussian Process predictions (1D)

In this assignment, you will be implement a 1D Gaussian process that predicts outputs based on noisy training data. You will be given (noisy) 1D training data pairs $D_{train} = \{(x_1, y_1), (x_2, y_2) \dots\}$. Your task is to predict the output for a set of test queries $D_{test} = \{\hat{x}_1, \hat{x}_2, ...\}$, conditioned on the training data.

The assignment requires you to implement two separate kernel functions, namely the:

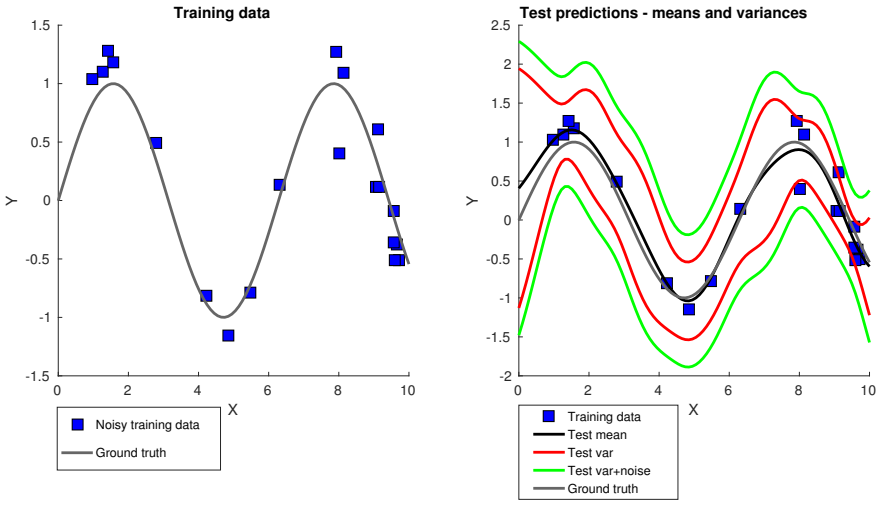


Figure 1: Gaussian process prediction using the Squared-Exponential kernel with default parameters.

- **Squared Exponential Kernel**: This is the kernel we discussed in class.

$$k(x_i, x_j) = \sigma_f^2 \; exp(-\frac{(x_i - x_j)^T M (x_i - x_j)}{2})$$

  where $\sigma_f$ is a scale factor for the kernel and $M$ is a metric measuring distance between two input vectors. In the 1D case, $M = \frac{1}{l^2}$ where $l$ is the length scale of the kernel.

- **Matern Kernel**: This kernel is used commonly in many machine learning applications.

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

  where $\nu$ and $l$ are (positive) parameters of the kernel and $r = |x_i - x_j|$. $K_\nu$ is a modified bessel function and $\Gamma$ is the gamma function. Good parameters settings for $\nu$ are 0.25 - 3.

The code for this section is in the file **gptest_1d.m** in the zip file. The function has the following syntax:

```
gptest_1d(interactive, useMaternKernel, noiseSigma, kernelLengthScale, kernelScaleFactor)
% Input:
%   interactive       - Flag for interactivity. If enabled, user can select
%                       training data by clicking (Default: false)
%   useMaternKernel   - Flag for using the matern kernel (Default: false)
%   noiseSigma        - Noise std. deviation (σ_n) (Default: 0.35)
%   kernelLengthScale - Length scale for the SE kernel (l) and Matern kernel (l) (Default: 1)
%   kernelScaleFactor - Scale factor for the SE kernel (σ_f) (Default: 1)
%                       If the matern kernel is chosen, ν = kernelScaleFactor (Default: 1.5)
```

The file can be used in two modes - interactive and non-interactive. If the interactive mode is chosen, the user can click on the GUI to create training samples. You are encouraged to use this to see how the Gaussian process

mean and uncertainty changes as more training points are given. You are also encouraged to play with the kernel parameters to see their effect on the GP. The code displays the training data and the predicted mean function and the respective variances (supplied by you) for the test queries. You can find the equations for the mean and variance predictions from the lecture slides. Note: the besselk function in MATLAB returns NaN values if the distance between points is zero. Set $k(x, x) = 1$ for these points as they are the same (high similarity = high kernel value). Fig. 1 shows results from a correct implementation of the SE kernel (using the default parameters).

## 2.2 Learning the dynamics of a cartpole using GPs

In this assignment, you will implement Gaussian processes to approximate the dynamics of a cartpole system. A cartpole is an underactuated system with a pendulum fixed to a cart and is controlled by applying a force on the cart. We represent the state $x$ and control $u$ of the cartpole system as follows:

```
% Cartpole state:
%       p         [m]     position of cart
%       dp        [m/s]   velocity of cart
%       dθ        [rad/s] angular velocity of the pendulum
%       θ         [rad]   angle of the pendulum
% Control:
%       u         [N]     force on cart
```

Our task is to learn a model of the cartpole dynamics:

$$x_{t+1} = f(x_t, u_t)$$

from data using Gaussian processes. We do this by interacting with a cartpole simulator which, over time provides us with the proper training data needed for the GP. We make two important modifications to the learning problem:

- First, we use an augmented state ($\hat{x}$) instead of the actual state ($x$) as input to the GP. The augmented state replaces the pendulum angle $\theta$ with the pair $[sin(\theta), cos(\theta)]$ in order to avoid the wrap-around issue with angles.
- Second, we predict delta-values of the state ($dx_t$), rather than the state directly. This reduces the difficulty of the problem as the model needs to only capture changes in state, centers the predictions (approximately) around zero and prevents wrap around issues when predicting $\theta$.

We now have the following inputs and outputs to the GP:

```
% GP dynamics model input:
%       [p, dp, dθ, sin(θ), cos(θ), u]
% GP dynamics model prediction:
%       [dp, d²p, d²θ, dθ] % delta-state, not next state
```

where $d^2p$ is the linear acceleration of the cart and $d^2\theta$ is the angular acceleration of the pendulum. The input of the GP is $6D$ while the output is $4D$. Since a GP only predicts a 1-dimensional output, we will train 4 separate GPs for this problem.

### 2.2.1 Learning the dynamics model

Let us now look at the training process for the Gaussian Process dynamics model. The system is setup to train in epochs, using data from the cartpole simulator to train a model that becomes more accurate with each epoch. Each epoch proceeds as follows:

1. At the start of an epoch, a random initial state $x_0$ and a sequence of $H$ controls are chosen (either randomly or via a preset policy) $U = \{u_0, u_1, ..., u_{H-1}\}$.

2. The system first propagates these controls through the cartpole simulator (a pre-specified parametric model) to generate a rollout of future states, as shown in Alg 1. We will use these predictions to (visually) evaluate how well our GP captures the dynamics of the cartpole. Additionally, these predictions will be added to the training data for the GP **at the end of the current epoch**.

**Algorithm 1** Rollout using Cartpole Simulator ($f_{SIM}$)

**Inputs**: Initial state: $x_0$, Controls: $U = \{u_0, u_1, ..., u_{H-1}\}$
**Output**: Future states: $\{x_1^s, x_2^s, \ldots, x_H^s\}$
**for** $t = 0 : H - 1$ **do**
  $x_{t+1}^s = f_{SIM}(x_t^s, u_t)$ ▷ Predict using the simulator

3. Next, we evaluate how well our GP model is able to predict the dynamics of the cartpole. To do this, we make predictions using the GP model, conditioned on the simulator rollouts from previous epochs. ***You need to implement this step***. Alg 2 shows pseudocode for generating the rollout. We compute the mean and variance of the predicted next state using the GP and propagate the mean prediction across time. We call this a *Mean-Rollout* since we discard the actual next state distribution and rollout only the mean.

**Algorithm 2** Mean-Rollout using GP ($f_{GP}$) - TO BE IMPLEMENTED

**Inputs**: Initial state: $x_0$, Controls: $U = \{u_0, u_1, ..., u_{H-1}\}$, Training inputs: $X$, Training targets: $Y$.
**Output**: Future states: $\{x_1, x_2, \ldots, x_H\}$
**for** $t = 0 : H - 1$ **do**
  $\hat{x}_t = g(x_t)$ ▷ Create augmented state
  **for** $k = 0 : 3$ **do** ▷ Predict using separate GP per output dimension
    $(\mu_t[k], \sigma_t[k]) = f_{GP}[k](\hat{x}_t, u_t)$ ▷ Predict delta-state mean and variance. $a[k] = k$th element of $a$
    $x_{t+1}[k] = x_t[k] + \mu_t[k]$ ▷ Compute next state's mean

4. The *Mean-Rollout* captures how well the GP does in expectation. In this step, we would like to see how the variance of the GP behaves as we predict across a long time horizon. Unlike the previous case, we will now make use of the complete Gaussian distribution predicted by the GP. Instead of using the mean delta-state, we will sample from the distribution around the mean, based on the predicted variance. This will result in trajectories that capture the uncertainties in the predictions over longer horizons. ***Once again, you need to implement this step***. Alg 3 shows the pseudocode for generating sampled rollouts. Ideally, to fully represent the uncertainty, we would generate multiple samples from the state distribution at each timestep of each trajectory. Unfortunately, this would result in the number of samples increasing exponentially over time. To avoid this, we only sample once from the distribution at each timestep and repeat this $N$ times to generate the rollouts. In practice, these samples will be near the Mean-Rollout initially and slowly move away as the uncertainty increases over time.

**Algorithm 3** Sampled Rollouts using GP ($f_{GP}$) - TO BE IMPLEMENTED

**Inputs**: Initial state: $x_0$, Controls: $U = \{u_0, u_1, ..., u_{H-1}\}$, Training inputs: $X$, Training targets: $Y$.
**Output**: Sampled Future states ($N$ samples per timestep): $\{x_1^0, x_2^0, \ldots, x_H^0\}, \{x_1^1, x_2^1, \ldots, x_H^1\}, \ldots, \{x_1^{N-1}, x_2^{N-1}, \ldots, x_H^{N-1}\}$

```
for j = 0 : N − 1 do                                   ▷ Generate N sampled trajectories
    for t = 0 : H − 1 do
        x̂_t^j = g(x_t^j)                               ▷ Create augmented state. ∀j, x_0^j = x_0
        for k = 0 : 3 do                               ▷ Predict using separate GP per output dimension
            (μ_t^j[k], σ_t^j[k]) = f_GP[k](x̂_t^j, u_t)   ▷ Predict delta-state mean and variance. a[k] = kth element of a
            s ∼ N(μ_t^j[k], σ_t^j[k])                  ▷ Sample a delta-state from the predicted Gaussian
            x_{t+1}^j[k] = x_t^j[k] + s                ▷ Compute next sampled state
```

5. Finally, at the end of each epoch, we compare the predictions from the GP (Mean/Samples) with the ground-truth from the cartpole simulator. The system displays the mean trajectory, the $N$ rolled out trajectories and plots the GP and simulator predictions with uncertainties. Also, the predictions from the cartpole simulator are added to the training data from the previous epoch. We use the augmented state and control pair $[\hat{x}_t, u_t]$ as the training inputs with the delta-states $dx_t$ as the targets.

### 2.2.2 Remarks:

A few points to note:

- The output is 4-dimensional. You need to create a separate GP per output dimension.
- The kernel to use for this problem is the Squared Exponential kernel. Unlike the previous problem, the kernel inputs are 6-dimensional. There is a different length scale for each dimension of the input (6 values per GP) and for each GP (4 GPs in total).
- The hyper-parameters (for each GP) are given to you for this task. You are encouraged to modify the hyper-parameters and see how the system behaves. Please reset the hyperparameters to the default values when submitting the code.
- With a successful implementation, you will see that the system does quite poorly at the start (the rollouts are all over the place) and starts to improve very fast. The predictions should match well against the simulator with low variance. One case where the system fails to do well even with a lot of training data is when the pendulum is upright as even a small force there can cause large changes in the state.

The code to be modified for this part of the homework is in **cartpole_test.m**. Some clarity on variable names in the code:

```
x               - Cartpole state
dx              - Delta-state / GP predictions
trainX          - Training inputs for the GP
trainY          - Training targets for the GP
kernelLengthScale  - Length scales for the GP (6 x 4 matrix, 6 dimensions per GP, 4 GPs in total)
```

You can find a video of a correct implementation for the first 10 epochs here (with default parameters):`https://courses.cs.washington.edu/courses/cse571/16au/homeworks/cartpole_res.avi`. Your results might be slightly different based on how you sample from the gaussian for the 10 rollouts. The results should have the general trend of high variance for the first few epochs (evidenced by the rollouts - transparent cartpoles moving all over) and lower variance as you get more data (tighter clumping of the rollouts).

## 3 Submission

You will be using Catalyst dropbox (`https://catalyst.uw.edu/collectit/dropbox/summary/barun/38890`) for submitting the homework. You need to submit the code including any helper functions you use. More instructions for submission will be available in the dropbox. The theory part of the homework can be submitted either in writing or electronically along with the code.