Probabilistic Robotics (CSE 571)
Homework 1 - Gaussian Processes

Due Sunday, October 18, 11:59 PM

This homework involves two programming assignments in MATLAB. In the first part, you will be implementing a 1D Gaussian process for predicting outputs given training data. In the second part, 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/cse571/15au/`).

Useful reading material: Lecture notes, Chapter 2 of Gaussian Processes for Machine Learning, Rasmussen and Williams (Available online at: `http://www.gaussianprocess.org/gpml/chapters/`)

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:

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

$$k(x_i, x_j) = \sigma_f^2 \; exp(-\frac{(x_i - x_j)^T M (x_i - x_j)}{2}) \tag{1}$$

  where σ_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.

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

  where ν and l are (positive) parameters of the kernel and $r = |x_i - x_j|$. K_ν is a modified bessel function and Γ is the gamma function. Good parameters settings for ν 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 to enable interactivity. If enabled, user can select
%                     training data by clicking (Default: false)
%   useMaternKernel - Flag to using the Matern kernel (Default:false)
%   noiseSigma      - Noise std. deviation (σ_n)
%   kernelLengthScale - length scale of the SE kernel (l) and Matern kernel (l)
%   kernelScaleFactor - Scale factor of the SE kernel (σ_f)
%                       If the Matern kernel is chosen, ν = kernelScaleFactor
```

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. 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.

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 z and control u of the cartpole system as follows:

```
% Cartpole state:
%       x         [m]     position of cart
%       dx        [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:

$$z_{t+1} = f(z_t, u_t) \tag{3}$$

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 ($\hat{s}$) instead of the actual state (s) as input to the GP. We replace the pendulum angle θ 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 (Δs), rather than the state directly. This is to prevent wrap around issues with θ and makes the system more robust in practice.

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

```
% GP dynamics model input:
%       [x, dx, dθ, sin(θ), cos(θ), u]
% GP dynamics model prediction:
%       [dx, d²x, d²θ, dθ] % delta-state, not next state
```

The system is setup to learn in epochs. Each epoch proceeds as follows:

- At the start of each epoch, a random initial state s_0 and a sequence of controls are chosen (either randomly or via a preset policy) $U = \{u_0, u_1, ..., u_{H-1}\}$.
- The system propagates these controls through the cartpole simulator and generates a rollout.
- The **same** sequence of controls are then propagated through the GP dynamics model to generate the GP mean and variance predictions. *You need to implement this step.* This should work as follows:

$$[\hat{s}_0, u_0] \rightarrow f_{GP} \rightarrow [\Delta s, var(\Delta s)] \rightarrow [\hat{s}_1, u_1] \rightarrow .... \tag{4}$$

- Next, the **same** sequence of controls are once again propagated through the GP dynamics model to generate N individual rollouts, but instead of using just the mean predictions, you have to sample from the Gaussian distribution around the mean. By doing this at each timestep, you can get a sense of how the uncertainty propagates through a long-horizon rollout. *You need to implement this step.* This should work as:

$$[\hat{s}_0, u_0] \rightarrow f_{GP} \rightarrow [\Delta s, var(\Delta s)] \rightarrow \bar{\Delta s} = \mathcal{N}(\Delta s, var(\Delta s)) \rightarrow [\hat{s}_1, u_1] \rightarrow .... \tag{5}$$

- The training data is augmented with the predictions from the cartpole simulator and the results from the current epoch are displayed. The system displays the mean trajectory, the N rolled out trajectories and plots the GP and simulator predictions.

The kernel to use for this problem is the Squared Exponential kernel. Few other points to note:

1. The output is 4-dimensional. You need to create a separate GP per output dimension.
2. The hyper-parameters (for each GP) are given to you for this task. There is a separate length scale for each dimension of the GP input (6D). You are encouraged to modify the hyper-parameters and see how the system behaves.
3. 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 quite 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**.

3 Submission

You will be using Catalyst dropbox (`https://catalyst.uw.edu/collectit/dropbox/barun/36444`) 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.