Piecewise Constant Adaptation Law

Aditya Gahlawat

1 Differential Equation Representation of PCA

While the piecewise constant adaptation (PCA) law can be implemented using `discrete callback` in Julia, for GPU parallelization, it is more straightforward to implement it as a dynamical system. For this purpose, given the simulation period Δ_t and sampling period $T_s = n \cdot T_s > 0$, for some $n \in \mathbb{N}$, we write

$$d\hat{\Lambda}_t = f_{ad}\left(t, X_t, \hat{X}_t\right), \quad \hat{\Lambda}_0 = 0_m, \tag{1}$$

where

$$f_{ad}\left(t, X_t, \hat{X}_t\right) = \begin{cases} \left(\frac{\lambda_s}{1-e^{\lambda_s T_s}}\right)\left(\hat{X}_t - X_t\right), & \text{if } \lfloor t/T_s \rfloor > \lfloor (t-\Delta_t)/T_s \rfloor \text{ and } t \geq T_s, \\ 0_m & \text{otherwise} \end{cases}.$$

Now, we use the `EM()`(`GPUEM()`) solvers that are based on the Euler-Maruyama method. Since (1) is diffusion-free, the Euler-Maruyama method reduces to the forward Euler method. Thus,

$$\hat{\Lambda}_{(k+1)\Delta_t} = \hat{\Lambda}_{k\Delta_t} + \Delta_t f_{ad}\left(k\Delta_t, X_{k\Delta_t}, \hat{X}_{k\Delta_t}\right), \quad k \in \mathbb{N}.$$

Following this, we see that

$$\begin{aligned} \hat{\Lambda}_t &= 0_m, \quad t \in [0, T_s), \\ \hat{\Lambda}_t &= \left(\frac{\lambda_s}{1-e^{\lambda_s T_s}}\right)\left(\hat{X}_{T_s} - X_{T_s}\right), \quad t \in [T_s, 2T_s), \\ \hat{\Lambda}_t &= \left(\frac{\lambda_s}{1-e^{\lambda_s T_s}}\right)\left(\hat{X}_{2T_s} - X_{2T_s}\right), \quad t \in [2T_s, 3T_s), \\ &\vdots \end{aligned}$$

and so on. This way, the PCA adaptation law can be incorporated as a differential equation object compatible with GPU parallelization.

2 Analysis

The system above is nothing but a sample-hold of the function $\left(\frac{\lambda_s}{1-e^{\lambda_s T_s}}\right)\left(\hat{X}_t - X_t\right)$, sampled at instances kT_s, $k \in \mathbb{N}$. To illustrate this, we test the dynamic sample-hold using the simple pendulum example. The code is at `test/DynamicPCS.jl`.

References

[1] Placeholder.