Learning for Adaptive and Reactive Robot Control
Instructions for Practical 1

Professor: Aude Billard

Spring Semester

Introduction

In this practical, you will apply the Model Predictive Control (MPC) and Dynamical Systems (DS) methods developed during previous exercise sessions to control the motion of the end-effector of a 7 degree-of-freedom Panda robot in 3D. You will apply and compare the SEDS and LPVDS methods to learn the DS. We provide you with different datasets to compare the performance of the algorithms.

1. The first of these datasets is generated from a theoretical DS, which allows for a ground truth comparison. The other two datasets are closer to reality.

2. The second dataset is generated from a MPC controller as you saw in the first exercise session. It shows you an example of how kinematically and dynamically feasible trajectories can be generated using MPC and subsequently learned to have a closed-form control law. To appreciate the advantage to have a closed-form solution, you will test the reactivity of the system to disturbances you can inject on-line.

3. The third dataset is generated from human demonstrations. This is a noisy dataset. You will then observe how noise affects performance of the learning and discuss methods to reduce these imperfections through some pre-processing.

If you skipped the lecture 4 exercises, you need to run the `setup_code_ch3.m` file to compile some libraries before starting the practical.
If you're using MacOS, there are additional steps to follow BEFORE running `setup_code_ch3.m`. Instructions are available in the READ_ME in the `lecture4-learning-control-laws/mac_setup` directory.

1 Dataset 1: Theoretical DS

The first dataset called `theoretical_DS_dataset.m` was generated by sampling path integrals from a *known* linear dynamical system using the script `dataset/generate_theoretical_DS.m`. It serves as a warm-up for the practical session, and as a reference on the ideal performances you can have with SEDS, as it was designed to match the SEDS formulation:

$$\dot{\mathbf{x}} = f(\mathbf{x}; \Theta) = \sum_{k=1}^{K} \gamma_k(\mathbf{x})(A^k\mathbf{x} + \mathbf{b}^k) \tag{1}$$

with $f(\mathbf{x}, \Theta)$ our theoretical dynamical system. To simplify things, we only use one linear DS (K=1), so $\gamma_k(\mathbf{x}) = 1$ and thus:

$$\dot{\mathbf{x}} = f(\mathbf{x}; \Theta) = A\mathbf{x} + \mathbf{b} \tag{2}$$

Here we build A from a rotation of $\pi/4$ around the x_1 axis, and $\mathbf{b}$ to have the system converging to the target $\mathbf{x}^* = \begin{bmatrix}0.5 & 0 & 0.3\end{bmatrix}^T$:

$$A = -R_x(\pi/4) \approx \begin{bmatrix}-1 & 0 & 0\\ 0 & -0.525 & 0.851\\ 0 & -0.851 & -0.525\end{bmatrix} \qquad \mathbf{b} = -A * \mathbf{x}^* \approx \begin{bmatrix}0.5\\ -0.255\\ 0.158\end{bmatrix} \tag{3}$$

TASK 1 Use the SEDS algorithm to learn a DS that fits this dataset. To do this, use the file `learning_3D_DS.m` which is a template file containing code to import a dataset and export a dynamical system for simulation. At line 60, add your code to learn SEDS, using solution code from exercise `ch3_ex3_seDS.m` (step 2a, 2b, 3 and 5) of lecture 4. Answer the following:

1. Determine the best parameters to fit SEDS. How many gaussians do you need at minimum for optimal performance (optimal mean-square error)? Change the variable `do_ms_bic` to 0, and define the variable `nb_gaussians` manually to test your hypothesis.
2. Change the boolean `do_ms_bic` back to 1 to determine automatically the best number of Gauss functions. Does this match your estimate?
3. Test each of the two initialization algorithm with the best number of Gauss functions from BIC and from your own estimate. Does this influence the estimate?
4. Test the two objective functions (MSE or Likelihood) with the optimal set of Gauss functions from BIC and your own. Does this influence the result?
5. Write you choice of parameters and the best evaluation metrics (step 4) you obtained with them:

6. [Optional] You can determine numerically A and $\mathbf{b}$ (equation 3) by using the correspondence between the SEDS model and a mixture of linear DS (book p.58). In the case we have one gaussian (K=1):

$$\begin{cases} A = \Sigma_{\dot{x}x}(\Sigma_x)^{-1} \\ \mathbf{b} = \boldsymbol{\mu}_{\dot{x}} - A\boldsymbol{\mu}_x \end{cases} \tag{4}$$

   Σ is the 6-by-6 covariance matrix and $\boldsymbol{\mu}$ is the 6-by-1 mean vector, respectively called `Sigma` and `Mu` in `ch3_ex3_seDS.m`. The upper-left corner (3-by-3 matrix) of Σ is Σ_x, and the bottom-left corner is $\Sigma_{\dot{x}x}$. Similarly, the three first rows of $\boldsymbol{\mu}$ correspond to $\boldsymbol{\mu}_x$ and the three bottom rows to $\boldsymbol{\mu}_{\dot{x}}$. Because the positions vector have been shifted into the attractor reference frame, $\mathbf{b} = \mathbf{x}^* = \begin{bmatrix}0 & 0 & 0\end{bmatrix}^T$.

TASK 2 Once you are satisfied with your results for task 2, you can launch the file `DS_control.m` to simulate the panda robot following your DS. The simulation uses the file `ds_control.mat`

that stores your DS. To create this file, you have to run the last part of `learning_3D_DS.m` with the correct option `usingSEDS` = *true*. During the simulation, you can use the keyboard to create force disturbances along the three Euler axes. The key correspondence is:

- `A` and `D` generate forces pointing on either direction of the $\hat{\mathbf{x}}$ axis
- `Q` and `E` generate forces pointing on either direction of the $\hat{\mathbf{y}}$ axis
- `W` and `S` generate forces pointing on either direction of the $\hat{\mathbf{z}}$ axis

TASK 3 Verify that the panda robot converges to the attractor from different initial positions, and that the path follows a spiral motion of the theoretical dataset.

TASK 4 [Optional]: Try changing the file `dataset/generate_theoretical_DS.m` to implement your own linear or nonlinear DS. Recall the conditions that need to be satisfied to have a stable DS and make sure to implement them correctly in 3D. How does the parameters of the dataset generation affects the results of the learned DS ?

1. `optSim.dt`: integration timestep
2. `optSim.iMax`: maximum number of integration steps
3. `optSim.tol`: tolerance on velocity at attractor (difference to zero)
4. `nPoints`: length of each trajectory in the dataset

2 Dataset 2: Learning Model Predictive Control

The second dataset has been generated from a Model Predictive Control (MPC) trajectory planner. Similarly to the exercises of lecture 1, this algorithm generates kinematically feasible trajectories from a random position to a target position. For this practical, we updated the model to work with the 7 degree-of-freedom Panda robot, added a model of the robot's dynamic, and orientation tracking. All these new features make this algorithm very slow to run, making it impossible to use in real-time applications.

The goal of this section is to approximate the MPC algorithm using SEDS and LPVDS. To do this, we provide you with a training and a testing dataset composed each of 10 trajectories computed offline by the MPC. You can open them the same way in the template file `learning_3D_DS.m` by loading `MPC_train_dataset.mat` or `MPC_test_dataset.mat` at line 26.

TASK 1 Try learning the MPC dataset using SEDS:

1. Create a new file from `learning_3D_DS.m` to learn the MPC dataset using SEDS. We suggest you start with the following parameters :
   - `nb_gaussians`: 12
   - Initialization algorithm 1
   - Objective function: Likelihood

   What is the best result you can get with these parameters ? Can a combination of them improve your fit ?

2. You can improve your solution by increasing the SEDS parameter `options.max_iter`. How much better is the solution ? You can use the `MPC_test_dataset.mat` to make sure you are not overfitting the `MPC_train_dataset.mat`. To do this, load `MPC_test_dataset.mat`, and run the *Plot Resulting DS* section, then the *Step 4 (Evaluation)* section, **without** re-solving SEDS.

TASK 2 To obtain a better fit, we will try using LPVDS.

Here, there are two main parameters which will affect your computations : the *GMM initialization* to estimate the number of gaussians K and the *set of constraints* used for the GMMs during optimization.

These parameters have three possible methods each, which are presented here again for simplicity.
Methods for GMM initialization :

- GMM-EM Model Selection via BIC (`est_options.type=1`)
- CRP-GMM : Bayesian non-parametric mixture model (`est_options.type=2`)
- Physically-Consistent non-parametric mixture model (`est_options.type=0`)

Types of constraints for GMMs during optimization :

- (O1) $\left\{ \left(A^k\right)^T + A^k \prec 0, \quad b^k = -A^k x^* \quad \forall k = 1, \dots, K \right.$
- (O2) $\left\{ \left(A^k\right)^T P + P A^k \prec 0, \quad b^k = 0 \quad \forall k = 1, \dots, K; \quad P = P^T > 0 \right.$

- (O3) $\left\{ \left(A^k\right)^T P + PA^k \prec Q^k, \quad Q^k = \left(Q^k\right)^T \prec 0, \quad b^k = -A^k x^* \quad \forall k = 1, \ldots, K \right.$.

1. Create a new file from `learning_3D_DS.m` to learn the MPC dataset using LPVDS. Use the code from the file `ch3_ex4_lpvDS.m` (step 2, 3 and 5) of lecture 4.
2. Change the variable `est_options.type` to try the different GMM initialization. Which one works best ?
3. Change the variable `constr_type` to 0, 1, or 2 to change the type of GMM constraint. Which one works best ?
4. Report here your best fit for the training and testing datasets. How does it compare with SEDS ?

5. Export your DS by setting the variable `usingSEDS` = $false$, and study the closed loop simulation of your DS using the file `DS_control.m`
6. Based on your results, compare MPC based Dynamical system with normal MPC. Give one advantage and one disadvantage when approximating the initial MPC trajectory planner.

3 Dataset 3: Teaching a motion using dynamical system

The last type of dataset comes from real-world recordings of the robot's joint positions and velocities, that were demonstrated by a human moving the robot's joints in a passive manner. We didn't do any post-processing on the data gathered during the experiment.

Figure 1: Experiement setup, with the panda robot at the attractor, and the demonstration dataset overlayed in red

Your task here is to reuse the code from Dataset 2 to learn this new dataset using SEDS or LPVDS, and understand how the imperfections of the real world affect the quality of the dataset and the fit of the algorithms. Figure out some simple corrections to apply to the dataset that would improve the fit.

TASK 1 Fit a DS to the demonstration dataset:

1. Start by again creating a new file based on the same template `learning_3D_DS.m` and modify it to load the dataset `demonstration_dataset.mat`
2. Observe the dataset: do you expect SEDS and LPVDS to achieve similar results or for one to be better than the other? Verify your assumption by running SEDS and LPVDS using the same best parameters as found for the MPC solution.
3. Try updating the parameters to improve the fit. What is the best result you can get ?

4. Find at least two reasons why the fit on this dataset is worse than the others.
   Hint : find where on the dataset the velocity is not well defined, ambiguous, or of

poor quality, resulting in a bad demonstration trajectory.
Hint 1 : Remember that on a real dataset, the velocity starts at zero at the beginning of the trajectory.

TASK 2 [Optional]: Implement some simple pre-processing on the dataset to correct these issues. Does the fit improve, and, if so, by how much ?

1. Plot each trajectory one-by-one (line 34 to 38 in `learning_3D_DS.m`) to detect bad demonstration trajectories, and remove them from the 3D array `trajectories` (6-by-`nPoints`-by-`nTraj`) .
2. Smooth each trajectory using a low pass filter. We recommend the `sgolayfilt` matlab function which is a non causal filter that works very well for this application.
3. Add any other step you judge necessary to improve this dataset.
4. How do these pre-processing steps improve the fit ? Report here your best results.

References

[1] Aude Billard, Sina Mirrazavi, and Nadia Figueroa. *Learning for Adaptive and Reactive Robot Control: A Dynamical Systems Approach*. MIT press, 2022.