Assignment 02: Comparison between different methods for taking account of underactuation in DDP

Gianluigi Grandesso* - gianluigi.grandesso@unitn.it

November 2021

1 Description

The goals of this assignment are:

- making practice using Differential Dynamic Programming (DDP) for generating reference trajectories and feedback control gains
- comparing two methods for considering underactuation in DDP applied to the case of a double pendulum without motor on the second joint (Pendubot)

2 Submission procedure

You are encouraged to work on the assignments in groups of 2 people. **If you have a good reason to work alone, then you can do it, but this has to be previously validated by one of the instructors**. Groups of more than 2 people are not allowed. The mark of each assignment contributes to 10% of you final mark for the class (i.e. 3 points out of 30).

When you are done with the assignment, please submit a single compressed file (e.g., zip). **The file name should contain the surnames of the group members**, and it must contain:

- A pdf file with the answers to the questions, the **names and ID number** of the group members; you are encouraged to include plots and/or numerical values obtained through simulations to support your answers. **This pdf does not need to be long. One or two pages of text should be enough to answer the questions. You can then add other pages for plots and tables.**
- The complete orc folder containing all the python code that you have developed.

If you are working in a group (i.e., 2 people) only one of you has to submit.

Submitting the pdf file without the code is not allowed and would result in zero points. Your code should be consistent with your answers (i.e. it should be possible to produce the results that motivated your answers using the code that you submitted). If your code does not even run, then your mark will be zero, so make sure to submit a correct code.

*Optimization-based Robot Control, Industrial Engineering Department, University of Trento.

3 DDP

DDP is a direct optimal control method for finite-horizon trajectory optimization that can deal with discrete nonlinear systems by using the dynamic programming principle, also known as Bellman principle of optimality, and a local quadratic approximation of the Q function to finally get an optimal open-loop control trajectory $\bar{u}$ and locally optimal linear feedback gains K that add together to form the final control u:

$$u = \bar{u} + K(\bar{x} - x) \tag{1}$$

where x is the current state of the system and $\bar{x}$ is the state reference trajectory.

This algorithm proceeds by alternating backward and forward passes: first the gradients and hessians of the Q and Value functions are computed to get the optimal variation of the control and then the system is simulated to compute the cost and set the new reference control and state trajectories, respectively. The algorithm stops when the convergence criterion is satisfied, meaning that the cost decrease computed in the forward pass remains under a certain threshold.

The general structure of the optimal control problem solved by DDP is as follows:

$$\begin{aligned} \min_{u_t} \quad & \sum_{t=1}^{N-1} l_t(x_t, u_t) + l_N(x_N) \\ \text{s.t.} \quad & x_{t+1} = f(x_t, u_t) \\ & x_0 = x^{init} \end{aligned} \tag{2}$$

where $l_t(x_t, u_t)$ is the running cost, $l_N(x_N)$ is the final cost and $f(x_t, u_t)$ is the discretized system dynamics.

For simplicity we consider a linearization of the system dynamics around the current reference trajectory. Thus, starting from an initial guess for the control trajectory U, the algorithm alternates between the linearization, the solution of the LQR problem to get the optimal variation of U and the line search to ensure convergence.

Minimizing the local quadratic model of Q we get the optimal control variation w^*:

$$w^* = \arg\min_w Q(z, w) = -Q_{uu}^{-1} \left(Q_u + Q_{ux} z\right) = \bar{w} + Kz \tag{3}$$

where z is the state variation from the current reference trajectory, $\bar{w} \triangleq -Q_{uu}^{-1} Q_u$, $K \triangleq -Q_{uu}^{-1} Q_{ux}$, Q_u is the gradient of Q w.r.t. u and Q_{uu} and Q_{ux} are the diagonal term regarding u and the off-diagonal term of the hessian of Q, respectively. Considering the definition of Q at timestep i $Q(z, w) \triangleq \min_w \left[l_i(z, w) + V\left(f(z, w), i + 1\right)\right]$, the terms Q_u, Q_{uu} and Q_{ux} are expressed as follows:

$$\begin{aligned} Q_u &= l_x + f_x^T V_x' \\ Q_{uu} &= l_{uu} + f_u^T V_{xx}' f_u + V_x' f_{uu} \\ Q_{ux} &= l_{ux} + f_x^T V_{xx}' f_x + V_x' f_{ux} \end{aligned} \tag{4}$$

where $V' \triangleq V(\cdot, i+1)$, $V_x \triangleq Q_x - Q_{ux} Q_{uu}^{-1} Q_u$, $V_{xx} \triangleq Q_{xx} - Q_{ux} Q_{uu}^{-1} Q_{ux}$ and the subscripts indicate w.r.t. which variables the derivation is performed.

If Q_{uu} is not invertible we can regularize it by adding a scalar $\mu > 0$ to its diagonal terms:

$$\overline{Q}_{uu} = Q_{uu} + \mu I \tag{5}$$

The pseudocode of the algorithm is illustrated here below:

Algorithm 1 DDP

1: Given $\overline{U} \leftarrow U^0$, compute $\overline{X}$ by simulating $x_{t+1} = f(x_t, u_t)$
2: **Backward Pass**
3: Set $V_x(N) = \nabla_x l_N$, $V_{xx} = \nabla_{xx} l_N$
4: **for** $i = N - 1...0$
5: Compute $Q_x, Q_u, Q_{xx}, Q_{uu}, Q_{ux}$
6: $w^* = -Q_{uu}^{-1}(Q_u + Q_{ux} z) = \bar{w} + Kz$
7: Compute $V_x(i), V_{xx}(i)$
8: **Forward Pass**
9: Set line search parameter $\alpha = 1$
10: Simulate system with $u = \overline{U} + \alpha\bar{w} + K\left(\overline{X} - x\right)$
11: **if** cost has not decreased enough:
12: Decrease α and go to 11
13: $\overline{U} \leftarrow u$, $\overline{X} \leftarrow x$
14: **if** not converged:
15: Perform **Backward Pass**

4 Underactuation

The aim of this assignment is to use DDP to control an Pendubot, namely a double pendulum with only the first joint actuated, and the goal is to perform a swing-up maneuver. Since in `example_robot_data` there is only the model of a fully actuated double pendulum, we have to load that model and then properly modify the code in order to take into account the underactuation.

4.1 Selection matrix

The first method to accomplish that relies on the use of a diagonal selection matrix $S \in \mathbb{R}^{n \times n}$, where n is the state dimension, such that:

$$\begin{cases} S[i,i] = 1 & if\ i\ corresponds\ to\ an\ actuated\ joint \\ S[i,i] = 0 & if\ i\ corresponds\ to\ a\ joint\ not\ actuated \end{cases} \tag{6}$$

By premultiplying the control torque vector in the system dynamics equations, this selection matrix is used to force the torques associated with the joints not actuated to be zero.

Notice that this modification of the system dynamics will affect also the computation of f_u. Since for simplicity we linearize the system dynamics around the current reference trajectory, the second order derivatives f_{uu} and f_{ux} are null.

4.2 Additional penalty

The second method consists in adding a term in the running cost with a large weight (`underact`) that penalizes the torque provided by a fictitious motor on the joint (`i`) actually not actuated.

$$running\_cost \mathrel{+}= \texttt{underact}||u_i|| \tag{7}$$

In doing so, please notice that the gradient and hessian of the running cost w.r.t. u must be modified accordingly.

5 Tests

The template code for this part of the assignment is located in:

`code/orc/02_assignment/ddp_doublependulum_template.py`

This file already contains all the code for formulating and solving with DDP the optimal control problem of a fully actuated double pendulum that has to perform a swing-up maneuver. The only parts that need to be implemented are two methods illustrated in the previous section. More precisely, the first method requires the definition of the selection matrix and the modification of the systems dynamics as well as of the function computing the gradient of the dynamics w.r.t. u (in the `DDPSolverDoublePendulum` class). For the second method instead, it is required to add the penalty shown in Eq. (7) in the running cost function (defined in the `DDPSolverLinearDyn` class) and modify accordingly its gradient and hessian w.r.t. u.

Two flags (`SELECTION_MATRIX` and `ACTUATION_PENALTY`) in the configuration file are used to select one of the two methods for taking account of the underactuation. Please do not change the parameters of DDP and the ones expressed in the configuration file.

Try to answer the following questions:

1. Implement the additional penalty method setting `underact` to `1e2`. Can DDP compute a state reference trajectory able to reach the final desired state? Then, simulating the system, is the controller able to make the Pendubot successfully perform the swing-up maneuver? Report the plots showing the reference and simulated position and control trajectories as well as the values of their cost (`Cost` and `Cost Sim.` printed in the terminal).

2. Implement the selection matrix method and answer to the same questions asked in the previous point.

3. What differences can you notice between the two methods in terms of reference motion and its cost? And in terms of simulated motion and its cost? Explain the reasons behind these discrepancies. [1]

4. Try to increase `underact` to `1e5`. Why do the discrepancies observed before disappear? [2]

5. If you had to use such controllers in a real robot, which one would you choose? Discuss your choice.

[1] The simulation cost will never perfectly match the reference cost because of the different timestep used in simulation, for more details see how the simulation cost is computed in `start_simu()`

[2] Except for a negligible difference in the cost of the reference motions.