Optimistic planning for deterministic systems
Practical work in Matlab

1 System

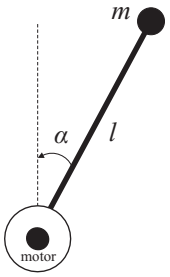

Figure 1: Inverted pendulum schematic.

We consider an inverted pendulum consisting of a weight attached to a disk, which is actuated by a DC motor and rotates in a vertical plane, see Figure 1. The continuous-time dynamics is:

$$\ddot{\alpha} = 1/J \cdot [mgl \sin(\alpha) - b\dot{\alpha} - K^2\dot{\alpha}/R + Ku/R] \tag{1}$$

where $J = 1.91 \cdot 10^{-4}\,\mathrm{kgm}^2$, $m = 0.055\,\mathrm{kg}$, $g = 9.81\,\mathrm{m/s}^2$, $l = 0.042\,\mathrm{m}$, $b = 3 \cdot 10^{-6}\,\mathrm{Nms/rad}$, $K = 0.0536\,\mathrm{Nm/A}$, $R = 9.5\,\Omega$. The angle α varies in the interval $[-\pi, \pi)\,\mathrm{rad}$, with $\alpha = 0$ pointing up, and 'wraps around' so that, e.g. a rotation of $3\pi/2$ corresponds to $\alpha = -\pi/2$. The state vector is $x = [\alpha, \dot{\alpha}]^T$ where α is the angle and $\dot{\alpha}$ is the angular velocity, with the convention that $\alpha = 0$ represents the pointing-up position. The control action u is the voltage, and the motor supports a range of $[-3, 3]\,\mathrm{V}$. The goal is to stabilize the pendulum in the unstable equilibrium $x_{eq} = [0, 0]^T$ (pointing up). Due to the limited voltage, from some initial states the pendulum cannot be pushed up in one go and must be first destabilized (pushed one way and then the other) to gather energy, prior being stabilized. This is called a swing-up and makes the solution highly nonlinear. The inverted pendulum is a commonly used example in the control field, and is particularly interesting for planning because the algorithm must plan over relatively long horizons in order to "understand" the swing-up nature of the solution, without being misled by the short-term rewards of always pushing in one direction towards the zero equilibrium.

We will use a Markov decision process formalism with a quadratic reward function $-(x^T Q_\rho x + R_\rho u^2)$ where $Q_\rho = \mathrm{diag}(1, 0.01)$ and $R_\rho = 0.3$, and a discount factor of 0.9. The system is discretized in time via numerical integration, with a sampling interval $T_s = 0.05\,\mathrm{s}$.

Algorithm

You are provided with start-up code in a ZIP file. Unzip the file in a directory on your computer, point Matlab to that directory, and add the appropriate subdirectories to the Matlab path by running `startup`.

The startup code implements UPD, Uniform Planning for Deterministic Systems. This algorithm works like OPD with one exception: it does not select the leaf to expand based on the largest b-value, instead it always expands a smallest-depth leaf. Thus it explores the tree uniformly, breadth-first. It still satisfies the a posteriori performance bound $v^* - v(\boldsymbol{u}^*) \leq \delta(d^*)$ where d^* is the deepest expanded depth. The algorithm is provided in Matlab function `upd`, and an example of using it in receding horizon for a 3 s long trajectory is given in the script `main`, which also shows how to visualize a graphical representation of the pendulum as it is being controlled, and how to plot the resulting trajectory at the end.

Your first task is to implement OPD starting from this uniform algorithm. To check that OPD is working, run it in closed loop with a budget n of at least 100 and it should bring the pendulum up in one swing-up.

Study

Vary the budgets of UPD and OPD gradually using a few values in a reasonable range (say, 5 values in the range 50-500) so that the experiment is computationally feasible within the allotted time. Try to use the analysis from the lecture in order to roughly predict the depths that will be reached by UPD while planning.

Compare the solutions of the two algorithms according to the following two metrics:

- Discounted return obtained along the trajectory (truncated at the end, of course).
- Depth of the planning tree, averaged over the steps in the trajectory.

Think about how you can (informally) relate the two results with each other using the analysis. Check whether your prediction for the UPD depths is (approximately) verified.

Increase the range of budgets up to a few thousand, and rerun the experiments. Study the execution time of OPD, averaging it over the steps in the trajectory. The analysis assumes that this time will be linear in the budget. Is this verified by the real data? If not, can you identify the reasons?