# HW #1 CS159

Due on 9 April 2021 at 9PM

## Instructions

Please LaTeX your solutions using the attached template. Fill in each section with your answers and please do not change the order of sections and subsections. You need to submit both a PDF and your code. We provided a code templated which can be found here:

https://github.com/1five9/1five9.github.io/raw/master/hw/hw_1.zip

In the `hw_1.zip` folder, we provided an anaconda environment that contains all packages that you need for this homework. Anaconda is a distribution that aims to simplify package management and deployment. More info and details about installation can be found here. After installation, you can create a conda environment from the attached `environment.yaml` file the using the following command:

```
conda env create -f py3_CS159.yaml --name py3_CS159
```

Then, you can activate such environment using the following command:

```
conda activate py3_CS159
```

**Note:** If you just updated your Mac OS to Big Sur and you are having issues running conda from the terminal here the fix: https://github.com/conda/conda/issues/10361#issuecomment-744019125. After following the instruction, please update your xcode environment installation.

## 1 Problem 1

In this problem, you will code the value iteration and policy iteration algorithms for the parking example[1] from Lecture # 1 (Fig. 1). As we have discussed in class, in this example a driver is looking for cheap parking on its way to the theater. At each parking spot $i$, the driver can park paying a cost of $N - i$ or move to the next parking spot $i + 1$ which may be free with probability $p$. Finally, if the driver has reached the parking spot $i = N$ and the parking spot is full, then the driver has to park at the garage and pay a cost of $C_g$.

The set of states

$$\mathcal{S} = \{0_f, 0_o, \dots, i_f, i_o, \dots, (N-1)_f, (N-1)_o, G, T\}, \tag{1}$$

where $i_f$ and $i_o$ are the states of the system representing the $i$-th parking spot being free or occupied, $G$ is the garage state and $T$ the theater state. At each state, the driver has two actions: `move`

[1] The parking example was presented in "Neuro-Dynamic Programming", Bertsekas, D. P., and Tsitsiklis, J. N., Athena Scientific, Belmont, MA 1996.

forward and park. The action move forward is used to go to the next parking spot which may be free with probability $p$ and the action park is used to park if the current parking spot is free, otherwise the system moves to the next parking spot. The transition probabilities are given as follows:

$$
\begin{aligned}
&p((i+1)_f|i_f, \texttt{move forward}) = p\\
&p((i+1)_o|i_f, \texttt{move forward}) = 1-p\\
&p((i+1)_f|i_o, \texttt{move forward}) = p\\
&p((i+1)_o|i_o, \texttt{move forward}) = 1-p\\
&p(G|(N-1)_f, \texttt{move forward}) = 1\\
&p(G|(N-1)_o, \texttt{move forward}) = 1\\
&p(T|i_f, \texttt{park}) = 1\\
&p((i+1)_f|i_o, \texttt{park}) = p\\
&p((i+1)_o|i_o, \texttt{park}) = 1-p\\
&p(G|(N-1)_f, \texttt{park}) = 1\\
&p(G|(N-1)_o, \texttt{park}) = 1
\end{aligned}
\tag{2}
$$

and the cost vector is described in Figure 1.

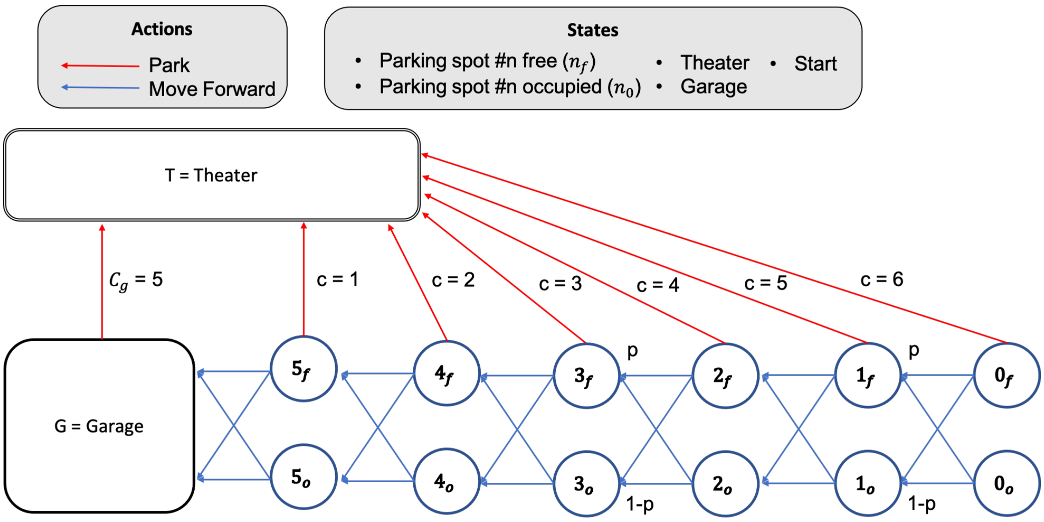


Figure 1: Parking example with 6 parking spots.

In the attached folder code/problem_1 you have the following files:

- main.py: This file loads the problem parameters and it initializes the MDP object.
- MDP.py: This files contains the MDP definition. You will modify this file.

## 1.1 Transition Matrices and Cost Function (3 points)

In this subsection, you will modify the method `buildTransitionMatrices` in the file `MDP.py`. Based on the transition probabilities from Equation (2) fill the entries of the matrices `P_move_forward` and `P_park`, and the vector `C_park` from the method `buildTransitionMatrices`.

**Please turn in** the values of the matrices `P_move_forward` and `P_park`, and the vector `C_park` for $N = 5$ and $p = 0.05$.

## 1.2 Closed-loop system Matrices (2 points)

Consider the control policy $\pi_{\text{tr}}$ that is uniquely defined by the threshold index tr:

$$\pi_{\text{tr}}(s) = \begin{cases} \texttt{park} & \text{If } s = i_f \text{ and } i \geq \text{tr} \\ \texttt{move forward} & \text{Otherwise.} \end{cases} \tag{3}$$

Basically, the above policy selects the action `park` when the driver reaches $i$-th free parking spot and $i \geq$ tr.

Uncomment line 18 of the file `main.py` and modify the method `computePolicy` to compute the matrix $P_{\pi_{\text{tr}}}$ and the cost vector $C_{\pi_{\text{tr}}}$, which describe the closed-loop system under the policy $\pi_{\text{tr}}$.

**Please turn in** the values of the the matrix $P_{\pi_{\text{tr}}}$ and the cost vector $C_{\pi_{\text{tr}}}$ for $N = 5$ and $p = 0.05$.

## 1.3 Policy Evaluation (2 points)

In this subsection, you will code the *Iterative Strategy* for policy evaluation described in Slide #48 of Lecture #1. Uncomment line 21 in file `main.py` and modify for loop in the method `policyEvaluation`.

**Please turn in** a snipped of the code and the value function vector for $N = 5$ and $p = 0.05$.

## 1.4 Value Iteration (2 points)

In this subsection, you will code the value iteration algorithm.

- Uncomment line 24 in file `main.py`
- Update the method `bellmanRecursion` to return the scalars `Vn_s` and `An_s` representing the value function and action at the state `s`. In order to compute these quantities you should use the Bellman recursion based on the value function vector `V`, the cost vector `C_a_s` and transition matrix `P_a_s`.
- Update the method `valueIteration` to update the vectors `Vnext` and `Anext` using the Bellman recursion and the latest value function vector `Vcurrent`. You need to code a for loop that evaluates the Bellman recursion for all states $s \in \mathcal{S}$. Notice that $|\mathcal{S}| = 2N + 2$, as there are $2N$ states for $N$ parking spots, the garage state and the theater state.

**Please turn in** the number of iterations required to reach convergence and the value function vector for $N = 5$ and $p = 0.05$. Also include a code snipped of the method `valueIteration`.

### 1.5 Policy Iteration (2 points)

In this subsection, you will code the policy iteration algorithm.

- Uncomment line 27 in file `main.py`
- Update the method `policyImprovement` which given a value function vector `V` updates the value function vector `Vn` and the action vector `An` using the Bellman recursion.
- Update the method `policyIteration` to compute the optimal value function. (Hint: you need to use the methods `computePolicy`, `policyEvaluation` and `policyImprovement`)

**Please turn in** the number of iterations required to reach convergence and the value function vector for $N = 5$ and $p = 0.05$. Also include a code snipped of the method `policyIteration`.

### 1.6 Testing on a bigger example (4 points)

In this section you will be testing your code on a bigger example. In the file `main.py` set $N = 200$, $Cg = 100$ and $printLevel = 0$. Then run the main file.

**Please turn in** the number of iterations required to reach convergence and the value of the value function vector for the first 2 states, both for the value iteration and policy iteration algorithms. Also include the optimal value of `iThreshold` at convergence.

## 2 Problem 2

In this problem, you will solve the following finite time optimal control problem:

$$
\begin{aligned}
J^*_{0\to N}(x_0) = \min_{\boldsymbol{u}_{0\to N}} \quad & \sum_{k=0}^{N-1} h(x_k, u_k) + q(x_N) \\
\text{subject to} \quad & x_{k+1} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k, \forall k \in \{0, \dots, N-1\}, \\
& \begin{bmatrix} -15 \\ -15 \end{bmatrix} \le x_k \le \begin{bmatrix} 15 \\ 15 \end{bmatrix}, \forall k \in \{0, \dots, N-1\}, \\
& -5 \le u_k \le 5, \forall k \in \{0, \dots, N-1\}, \\
& x_N \in \mathcal{X}_F, x_0 = [-15, 15]^\top.
\end{aligned}
$$

You will recast the above finite optimal control problem as a Quadratic Program (QP) using the strategy presented in class. In particular, we will recast the above problem as the following QP:

$$
\begin{aligned}
J^*_0(x(0)) = \min_{U_0, X_0} \quad & [U_0^\top, X_0^\top, x(0)] \begin{bmatrix} \overline{Q} & 0 \\ 0 & \overline{R} \end{bmatrix} \begin{bmatrix} U_0 \\ X_0 \\ x(0) \end{bmatrix} \\
\text{subject to} \quad & G_{0,\text{in}} \begin{bmatrix} X_0 \\ U_0 \end{bmatrix} \le E_{0,\text{in}} x(0) + w_{0,\text{in}} \\
& G_{0,\text{eq}} \begin{bmatrix} X_0 \\ U_0 \end{bmatrix} = E_{0,\text{eq}} x(0).
\end{aligned} \tag{4}
$$

where the matrices $\overline{Q}$, $\overline{R}$, $G_{0,\text{in}}$, $E_{0,\text{in}}$, $w_{0,\text{in}}$, $G_{0,\text{eq}}$ and $E_{0,\text{eq}}$ are defined as in the slide set from Lecture # 2 (Section: "Quadratic Program without Substitution").

In the attached folder `code/problem_2` you have the following files:

- `main.py`: This file loads the problem parameters and it initializes the FTOCP object.
- `ftocp.py`: This files contains the FTOCP definition. You will modify this file.

## 2.1 Cost Reformulation (1 points)

In this subsection, you will construct the cost matrices $\overline{Q}$ and $\overline{R}$ from Problem (4). Edit the method `buildCost` in the file `ftocp.py` to update the variables `barQ` and `barR` so that

$$\texttt{barQ} = \overline{Q} \text{ and } \texttt{barR} = \overline{R}.$$

**<u>Please turn in</u>** the matrices `barQ` and `barR`.

## 2.2 Inequality Constraint Reformulation (3 points)

In this subsection, you will construct the cost matrices $G_{0,\text{in}}$, $E_{0,\text{in}}$ and $w_{0,\text{in}}$ from Problem (4). Edit the method `buildIneqConstr` in the file `ftocp.py` to update the variables `G_in`, `E_in` and `w_in` so that

$$\texttt{G\_in} = G_{0,\text{in}},\ \texttt{E\_in} = E_{0,\text{in}} \text{ and } \texttt{w\_in} = w_{0,\text{in}}.$$

**<u>Please turn in</u>** the matrices `G_in`, `E_in` and `w_in`, and a code snipped for the method `buildIneqConstr`.

## 2.3 Equality Constraint Reformulation (3 points)

In this subsection, you will construct the cost matrices $G_{0,\text{eq}}$ and $E_{0,\text{eq}}$ from Problem (4). Edit the method `buildEqConstr` in the file `ftocp.py` to update the variables `G_eq` and `E_eq` so that

$$\texttt{G\_eq} = G_{0,\text{eq}} \text{ and } \texttt{E\_eq} = E_{0,\text{eq}}.$$

**<u>Please turn in</u>** the matrices `G_eq` and `E_eq`, and a code snipped for the method `buildEqConstr`.

## 2.4 Solve the FTOCP (3 points)

In this subsection, you will solve the FTOCP. Uncomment lines #38-48 in the file `main.py` and run the script which should print to screen the optimal solution.

**<u>Please turn in</u>** the optimal state and input trajectories (copy and paste the exact solution) and a plot of state trajectory on the $x_1$-$x_2$ plane.

## 3 Problem 3

In this problem, you will solve the following finite time optimal control problem:

$$
\begin{aligned}
J^*_{0\to N}(x_0) = \min_{\boldsymbol{u}_{0\to N}} \quad & \sum_{k=0}^{N-1} h(x_k, u_k) + q(x_N) \\
\text{such that} \quad & x_{k+1} = \begin{bmatrix} 1 & 0.5^k \\ 0 & 1 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u_k + \begin{bmatrix} 0 \\ 0.1^k \end{bmatrix}, \forall k \in \{0, \dots, N-1\}, \\
& \begin{bmatrix} -15 \\ -15 \end{bmatrix} \leq x_k \leq \begin{bmatrix} 15 \\ 15 \end{bmatrix}, \forall k \in \{0, \dots, N-1\}, \\
& -5 \leq u_k \leq 5, \forall k \in \{0, \dots, N-1\}, \\
& x_N \in \mathcal{X}_F, x_0 = [-15, 15]^\top.
\end{aligned}
\tag{5}
$$

Notice that in the above problem the system dynamics are affine time-varying.

In the attached folder `code/problem_3` you have the following files:

- `main.py`: This file loads the problem parameters and it initializes the FTOCP object.
- `ftocp.py`: This files contains the FTOCP definition. You will modify this file.

### 3.1 QP formulation (4 points)

In this subsection, we will recast Problem (5) as a QP. Rewrite Problem (5) as a QP where the optimization vector is $[X_0, U_0]$ (*Hint: you may use the matrices from the QP* (4) *and define one new vector* `C_eq`*.* )

**<u>Please turn in</u>** your QP formulation.

### 3.2 Solve the QP (6 points)

In this subsection, we will solve Problem (5). We have implemented the time-varying dynamics in lines #35–46, please take a look before starting the implementing the QP. (*Hint: You should use the code that you developed for the previous problem with some small changes*).

- Modify the method `buildCost` in the `ftocp.py` file.
- Modify the method `buildIneqConstr` in the `ftocp.py` file.
- Modify the method `buildEqConstr` in the `ftocp.py` file.
- Modify the method `solve` in the `ftocp.py` file.

Run the file `ftocp.py` to solve Problem (5) and plot the closed-loop trajectory.

**<u>Please turn in</u>** a snipped of code for each of the four methods that you have modified, the optimal state and input trajectories (copy and paste the exact solution) and a plot of state trajectory on the $x_1$-$x_2$ plane.