Stochastic Models and Optimization: Problem Set 4

Roger Garriga Calleja, José Fernando Moreno Gutiérrez, David Rosenfeld, Katrina Walker

March 18, 2017

Q1

We are given a linear-quadratic problem with perfect state information but with a forecast. We first set up the primitives of the system:
x_k: the state in period k
u_k: the decision variable in period k
w_k: the disturbances in period k
y_k: an accurate prediction that w_k will be selected according to a particular probability distribution $P_{k|y_k}$

We set up the dynamics of the problem with a linear system, as follows:

$$x_{k+1} = A_k x_k + B_k u_k + w_k$$

Where A and B are $n \times n$ matrices.
We also have a quadratic cost:

$$g_N(x_N) = x_N' Q_N x_N$$

$$g_k(x_k) = x_k' Q_k x_k + u_k' R_k u_k$$

Where Q_k and R_k are $n \times n$ positive definite matrices.

Our problem is thus to minimise:

$$E[\sum_{k=0}^{N-1} (x_k' Q_k x_k + u_k' R_k u_k) + x_N' Q_N x_N]$$

Subject to:

$$x_{k+1} = A_k x_k + B_k u_k + w_k$$

We can now set up our DP-algorithm:

$$J_N(x_N, y_N) = x_N' Q_N x_N$$

$$J_k(x_k, y_k) = \min_{u_k \in R^n} E_{w_k}[x_k' Q_k x_k + u_k' R_k u_k + J_{k+1}(x_{k+1}, y_{k+1})]$$

We can see that $J_N(x_N)$ is of the form $J(x_k, y_k) = x_k' K_k x_k + x_k' b_k(y_k) + c(y_k)$, with$Q_n = K_n$, $x_N' b_N(y_N) = 0$ and $c(y_k) = 0$.
We assume that this is also true at stage k+1, so that:

$$J(x_{k+1}, y_{k+1}) = x_{k+1}' K_{k+1} x_{k+1} + x_{k+1}' b_{k+1}(y_{k+1}) + c(y_{k+1})$$

Where $b_{k+1}(y_{k+1})$ is an n-dimensional vector and $c(y_{k+1})$ is a scalar. Using this, we can compute $J_k(x_k, y_k)$:

$$
\begin{aligned}
J_k(x_k, y_k) &= \min_{u_k \in R^n} E_{w_k}[x_k' Q_k x_k + u_k' R_k u_k + J_{k+1}(x_{k+1}, y_{k+1})] = \\
&= \min_{u_k \in R^n} E_{w_k}[x_k' Q_k x_k + u_k' R_k u_k + x_{k+1}' K_{k+1} x_{k+1} + x_{k+1}' b_{k+1}(y_{k+1}) + c(y_{k+1}) \\
&= \min_{u_k \in R^n} E_{w_k}[x_k' Q_k x_k + u_k' R_k u_k + (A_k x_k + B_k u_k + w_k)' K_{k+1}(A_k x_k + B_k u_k + w_k) + \\
&+ (A_k x_k + B_k u_k + w_k)' b_{k+1}(y_{k+1}) + c(y_{k+1})] = x_k'(Q_k + A_k' K_{k+1} A_k) x_k + E[w_k' K_{k+1} w_k] + \\
&+ \min_{u_k \in R^n} [u_k'(R_k + B_k' K_{k+1} B_k) u_k + 2x_k' A_k K_{k+1} B_k u_k + 2u_k B_k K_{k+1} E[w_k | y_k] + \\
&+ u_k' B_k' b_{k+1}(y_{k+1})] + x_k' A_k' K_{k+1} E[w_k | y_k] + x_k' A_k' b_{k+1}(y_{k+1}) + E[w_k | y_k]' b_{k+1}(y_{k+1})
\end{aligned}
$$

We then find our optimal decision by taking the derivative of $J_k(x_k, y_k)$ with respect to u_k and set it equal to zero in order to solve for our optimal solution $u*$:

$$2(R_k + B_k' K_{k+1} B_k) u_k^* + 2B_k' K_{k+1} A_k x_k + 2B_k K_{k+1} B_k + 2B_k K_{k+1} E[w_k | y_k] + B_k' b_{k+1}(y_{k+1}) = 0$$

$$u_k^* = (R_k + B_k' K_{k+1} B_k)^{-1} B_k' K_{k+1}(A_k x_k + E[w_k | y_k]) + \alpha_k$$

Where $\alpha_k = B_k' b_{k+1}(y_{k+1})$

Q2

(i)

Controllability is related to the stability of the system. It allows that the eigenvalues of $A + BL$ could have magnitude less than 1, and then the closed loop system is stable. It also is an important factor for determining the convergence rate of the system. Observability is an important condition for making K merely positive definite. Thus, controllability and observability allow K to satisfy the Algebraic Riccati Equation and guarantee uniqueness of the optimal control.

(ii, iii, iv, v)

Python Code:

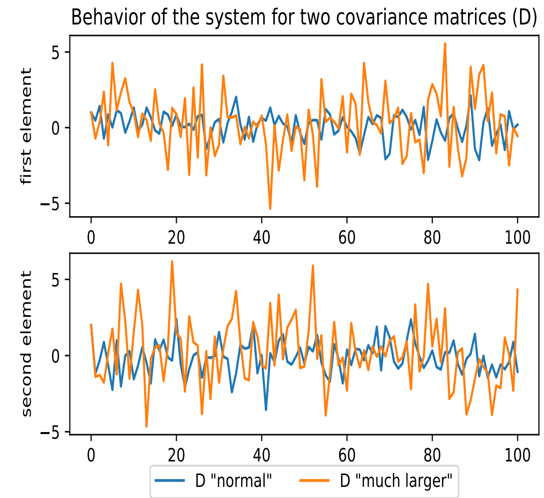
```python
# Import libraries
import numpy as np #Load numpy library for matrices operations
import matplotlib.pyplot as plt # library for graphics
from matplotlib.backends.backend_pdf import PdfPages # save graphics as pdf
import control #Load control for solving Riccati Equation
np.random.seed(1234)

# System components that are not going to be modified
N = 100 # horizon (time periods)
n = 2

A = np.array([[2, 0], [1, 0]])
B = np.array([[0, 2], [1, 1]])
C = np.array([[0, 3]])

Q = np.matmul(np.transpose(C), C)
#np.linalg.eigvals(Q)

# Behaviour of the system
# ii ------------------------------------------------------------
# Fix R and x[0], D_1 = normal - D_2 = "much larger"
R = np.diag(np.repeat(1, [n], axis = 0))
x_1 = np.empty([N + 1, n])
x_1[0] = [1, 2]
D_1 = np.diag(np.repeat(1, [n], axis = 0))
w_1 = np.random.multivariate_normal(mean = np.repeat(0, [n], axis = 0), cov = D_1, size = N)
x_2 = np.empty([N + 1, n])
x_2[0] = x_1[0]
D_2 = np.diag(np.repeat(5, [n], axis = 0))
w_2 = np.random.multivariate_normal(mean = np.repeat(0, [n], axis = 0), cov = D_2, size = N)
#np.linalg.eigvals(K)
K = np.empty([N + 1, n, n])
K[N] = Q
for i in range(100,0,-1):
    K[i-1] = np.matmul(np.matmul(np.transpose(A), K[i] - np.matmul(np.matmul(K[i], B), np.linalg.inv(np.matmul(np.matmul(np.transpose(B), K[i]), B) + R), np.matmul(np.transpose(B), K[i]))), A) + Q
```

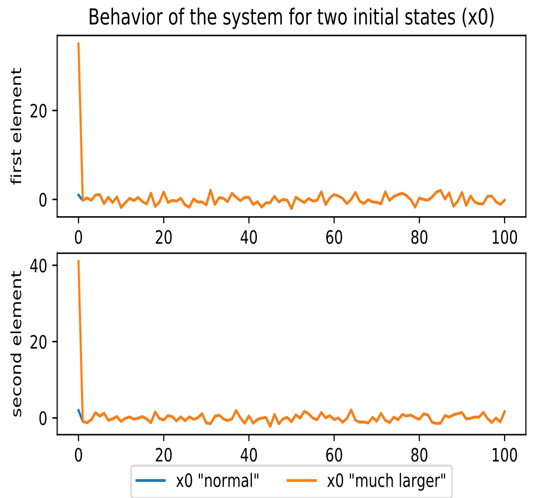
```
L = np.empty([N, n, n])
for i in range(0, N):
    L[i] = - np.matmul(np.linalg.inv(np.matmul(np.matmul(np.transpose(B), K[i+1]), B) + R), np.matmul(np.matmul(np.transpose(B), K[i+1]), A))

for i in range(0, N):
    # Values normal
    x_1[i+1] = np.matmul(A + np.matmul(B, L[i]), x_1[i]) + w_1[i]
    # Values much larger
    x_2[i+1] = np.matmul(A + np.matmul(B, L[i]), x_2[i]) + w_2[i]

# plot
plot_fig2 = plt.figure(2)
plt.subplot(211)
plt.title('Behavior of the system for two covariance matrices (D)')
plt.ylabel('first element')
plt.plot(x_1.T[0])
plt.plot(x_2.T[0])
plt.subplot(212)
plt.ylabel('second element')
plt.plot(x_1.T[1], label = 'D "normal"')
plt.plot(x_2.T[1], label = 'D "much larger"')
plt.legend(loc='lower center', bbox_to_anchor=(0.5, -0.4), ncol=2)

# iii ---------------------------------------------------------
# Fix R and D, x_1[0] = normal - x_2[0] = "much larger"
R = np.diag(np.repeat(1, [n], axis = 0))
x_1 = np.empty([N + 1, n])
x_1[0] = [1, 2]
D = np.diag(np.repeat(1, [n], axis = 0))
w = np.random.multivariate_normal(mean = np.repeat(0, [n], axis = 0), cov = D, size = N)
x_2 = np.empty([N + 1, n])
x_2[0] = [35, 41]
K = np.empty([N + 1, n, n])
K[N] = Q
for i in range(100,0,-1):
    K[i-1] = np.matmul(np.matmul(np.transpose(A), K[i] - np.matmul(np.matmul(K[i], B), np.linalg.inv(np.matmul(np.matmul(np.transpose(B), K[i]), B) + R), np.matmul(np.transpose(B), K[i])))), A) + Q

L = np.empty([N, n, n])
for i in range(0, N):
    L[i] = - np.matmul(np.linalg.inv(np.matmul(np.matmul(np.transpose(B), K[i+1]), B) + R), np.matmul(np.matmul(np.transpose(B), K[i+1]), A))

for i in range(0, N):
    # Values normal
    x_1[i+1] = np.matmul(A + np.matmul(B, L[i]), x_1[i]) + w[i]
    # Values much larger
    x_2[i+1] = np.matmul(A + np.matmul(B, L[i]), x_2[i]) + w[i]

# plot
```

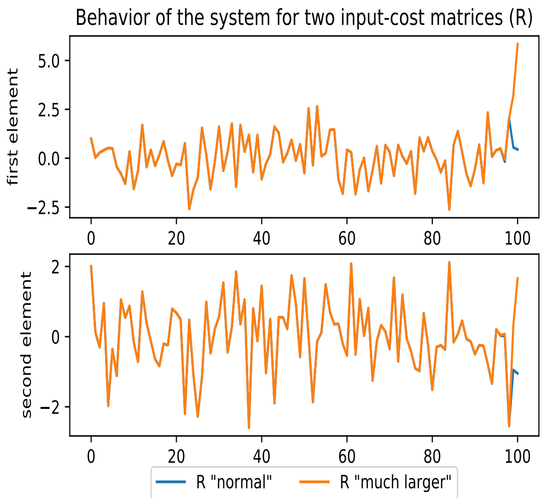
```python
plot_fig3 = plt.figure(3)
plt.subplot(211)
plt.title('Behavior of the system for two initial states (x0)')
plt.ylabel('first element')
plt.plot(x_1.T[0])
plt.plot(x_2.T[0])
plt.subplot(212)
plt.ylabel('second element')
plt.plot(x_1.T[1], label = 'x0 "normal"')
plt.plot(x_2.T[1], label = 'x0 "much larger"')
plt.legend(loc='lower center', bbox_to_anchor=(0.5, -0.4), ncol=2)

# iv ---------------------------------------------------------------
# Fix x[0] and D, R_1[0] = normal - R_2[0] = "much larger"
R_1 = np.diag(np.repeat(1, [n], axis = 0))
R_2 = np.diag(np.repeat(100, [n], axis = 0))
x_1 = np.empty([N + 1, n])
x_1[0] = [1, 2]
D = np.diag(np.repeat(1, [n], axis = 0))
w = np.random.multivariate_normal(mean = np.repeat(0, [n], axis = 0), cov = D,
    size = N)
x_2 = np.empty([N + 1, n])
x_2[0] = x_1[0]

K_1 = np.empty([N + 1, n, n])
K_1[N] = Q
for i in range(100,0,-1):
    K_1[i-1] = np.matmul(np.matmul(np.transpose(A), K_1[i] - np.matmul(np.matmul
        (K_1[i], B), np.linalg.inv(np.matmul(np.matmul(np.transpose(B), K_1[i]),
        B) + R_1), np.matmul(np.transpose(B), K_1[i])))), A) + Q

L_1 = np.empty([N, n, n])
for i in range(0, N):
    L_1[i] = - np.matmul(np.linalg.inv(np.matmul(np.matmul(np.transpose(B), K_1[
        i+1]), B) + R_1), np.matmul(np.matmul(np.transpose(B), K_1[i+1]), A))

K_2 = np.empty([N + 1, n, n])
K_2[N] = Q
for i in range(100,0,-1):
    K_2[i-1] = np.matmul(np.matmul(np.transpose(A), K_2[i] - np.matmul(np.matmul
        (K_2[i], B), np.linalg.inv(np.matmul(np.matmul(np.transpose(B), K_2[i]),
        B) + R_2), np.matmul(np.transpose(B), K_2[i])))), A) + Q

L_2 = np.empty([N, n, n])
for i in range(0, N):
    L_2[i] = - np.matmul(np.linalg.inv(np.matmul(np.matmul(np.transpose(B), K_2[
        i+1]), B) + R_2), np.matmul(np.matmul(np.transpose(B), K_2[i+1]), A))

for i in range(0, N):
    # Values normal
    x_1[i+1] = np.matmul(A + np.matmul(B, L_1[i]), x_1[i]) + w[i]
    # Values much larger
    x_2[i+1] = np.matmul(A + np.matmul(B, L_2[i]), x_2[i]) + w[i]

```

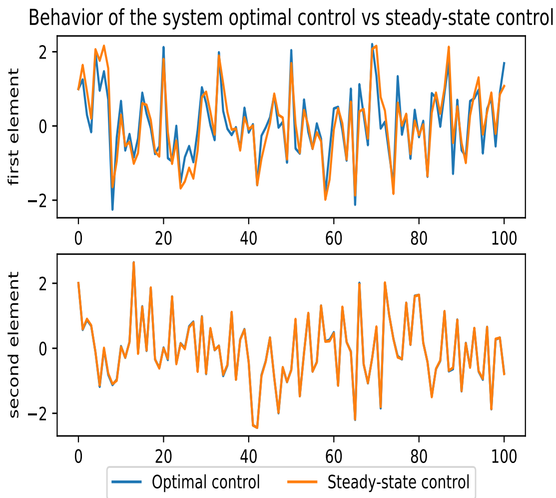
```
# plot
plot_fig4 = plt.figure(4)
plt.subplot(211)
plt.title('Behavior of the system for two input-cost matrices (R)')
plt.ylabel('first element')
plt.plot(x_1.T[0])
plt.plot(x_2.T[0])
plt.subplot(212)
plt.ylabel('second element')
plt.plot(x_1.T[1], label = 'R "normal"')
plt.plot(x_2.T[1], label = 'R "much larger"')
plt.legend(loc='lower center', bbox_to_anchor=(0.5, -0.4), ncol=2)

# v -----------------------------------------------------------------
# Fix x[0] and D, R_1[0] = normal - R_2[0] = "much larger"
R = np.diag(np.repeat(1, [n], axis = 0))
x_1 = np.empty([N + 1, n])
x_1[0] = [1, 2]
D = np.diag(np.repeat(1, [n], axis = 0))
w = np.random.multivariate_normal(mean = np.repeat(0, [n], axis = 0), cov = D,
    size = N)
x_2 = np.empty([N + 1, n])
x_2[0] = x_1[0]

K_1 = np.empty([N + 1, n, n])
K_1[N] = Q
for i in range(100,0,-1):
    K_1[i-1] = np.matmul(np.matmul(np.transpose(A), K_1[i] - np.matmul(np.matmul
        (K_1[i], B), np.linalg.inv(np.matmul(np.matmul(np.transpose(B), K_1[i]),
        B) + R), np.matmul(np.transpose(B), K_1[i])))), A) + Q

L_1 = np.empty([N, n, n])
for i in range(0, N):
    L_1[i] = - np.matmul(np.linalg.inv(np.matmul(np.matmul(np.transpose(B), K_1[
        i+1]), B) + R), np.matmul(np.matmul(np.transpose(B), K_1[i+1]), A))

K_2, G, E = control.dare(A = A, B = B, Q = Q, R = R)

L_2 = - np.matmul(np.linalg.inv(np.matmul(np.matmul(np.transpose(B), K_2), B) +
    R), np.matmul(np.matmul(np.transpose(B), K_2), A))

for i in range(0, N):
    # Values normal
    x_1[i+1] = np.matmul(A + np.matmul(B, L_1[i]), x_1[i]) + w[i]
    # Values much larger
    x_2[i+1] = np.matmul(A + np.matmul(B, L_2), x_2[i]) + w[i]

# plot
plot_fig5 = plt.figure(5)
plt.subplot(211)
plt.title('Behavior of the system optimal control vs steady-state control')
plt.ylabel('first element')
plt.plot(x_1.T[0])
plt.plot(x_2.T[0])
```

```
plt.subplot(212)
plt.ylabel('second element')
plt.plot(x_1.T[1], label = 'Optimal control')
plt.plot(x_2.T[1], label = 'Steady-state control')
plt.legend(loc='lower center', bbox_to_anchor=(0.5, -0.4), ncol=2)

pp = PdfPages('/home/chpmoreno/Dropbox/Documents/BGSE/Second_Term/SMO/Problemsets/PS4/figures.pdf')
pp.savefig(plot_fig2)
pp.savefig(plot_fig3)
pp.savefig(plot_fig4)
pp.savefig(plot_fig5)
pp.close()
```









Q3

Asset selling w/offer estimation

Primitives

- x_k current offer.
- $\{w_k\}$ iid offers with an unknown underlying distribution F_1 or F_2.
- constraints: selling (u_1) or not selling (u_2)): $\begin{cases} \{u^1, u^2\} \text{ if } x_k \neq \mathrm{T} \\ 0, \text{otherwise} \end{cases}$
- rewards: $g_n(x_N) = \begin{cases} x_N, \text{ if } x_N \neq \mathrm{T} \\ 0, \text{otherwise} \end{cases}$
  $g_k(x_k, u_k, w_k) = \begin{cases} (1+r)^{N-k} x_k, \text{ if } x_k \neq \mathrm{T} \text{ and if } u_k = u^1 \\ 0, \text{otherwise} \end{cases}$
- q = prior belief that F_1 is true
- $q_{k+1} = \frac{\mathbb{P}\{y_k = y^1 | w_0, \ldots, w_k\}}{\mathbb{P}(w_1 = w_1)} = \frac{q_k F_1(w_k)}{q_k F_1(w_k) + (1-q) F_2(w_k)}$

Now, we can apply the DP algorithm to find an optimal asset selling policy

$$J_{N-1}(x_{N-1}) = \begin{cases} \max\{(1+r)x_k, (q_{N-1}\mathbb{E}_{F_1}[J_N(w_{N-1})] + (1-q_{N-1})\mathbb{E}_{F_2}[J_N(w_{N-1})]) \text{ if } x_{N-1} \neq T \\ 0, \text{otherwise} \end{cases}$$

$$J_k(x_k) = \begin{cases} \max(q_k\mathbb{E}_{F_1}[J_{k+1}(w_k)] + (1-q_k)\mathbb{E}_{F_2}[J_{k+1}(w_k)]), (1+r)^{N-k}x_k \text{ if } x_k \neq T \\ 0, \text{otherwise} \end{cases}$$

Thus, the threshold for selling an asset will be: $(1+r)^{N-k}x_k \geqslant q_k\mathbb{E}_{F_1}[J_{k+1}(w_k)] + (1-q_k)\mathbb{E}_{F_2}[J_{k+1}(w_k)]$

So, the optimal asset selling policy will be: $\mu^*(x_k) = \begin{cases} u^1, x_{k+1} \geqslant \frac{q_k\mathbb{E}_{F_1}[J_{k+1}(w_k)] + (1-q_k)\mathbb{E}_{F_2}[J_{k+1}(w_k)]}{(1+r)^{N-k}} \\ u^2, \text{otherwise} \end{cases}$

Q4

This problem is basically the same as the inventory management considering the demand as a random variable following an unknown distribution. It is a case with imperfect state information, in which the distribution of demand will be either F_1 or F_2. The probability that the demand follows F_1 is updated at each period k after observing the realization of the demand. That will effect the way the expectation of the demand is computed.

Primitives:
x_k: items in the inventory at period k.
u_k: quantity ordered at period k.
w_k: demand during period k. w_k are iid with probability distribution either F_1 or F_2.
q_k: probability that w_k follows distribution F_1.
$q_0 = q$: a priori probability that demand follows the distribution F_1.

Dynamics:
$x_{k+1} = x_k + u_k - w_k$
$q_{k+1} = \frac{q_k f_1(w_k)}{q_k f_1(w_k) + (1-q_k) f_2(w_k)}$, where $f_i(w)$ is the pdf of the distribution F_i.

Cost:
$g_N(x_N) = 0$.
$g_k(x_k, u_k, w_k) = cu_k + h\max\{0, w_k - x_k - u_k\} + p\max\{0, x_k + u_k - w_k\}$, where c, h, p are positive and $p > c$.

DP algorithm:

$J_N(x_N) = 0$
$J_k(x_k) = \min\limits_{u_k \geqslant 0} \mathbb{E}\left[cu_k + h\max\{0, w_k - x_k - u_k\} + p\max\{0, x_k + u_k - w_k\} + J_{k+1}(x_{k+1})\right]$

In order to solve it we can introduce the variable $y_k = x_k + u_k$, and the we have
$J_k(y_k) = \min\limits_{u_k \geqslant x_k} G_k(y_k) - cx_k$, where

$$G_k(y_k) = cy + h\mathbb{E}[\max\{0, w_k - y_k\}] + p\mathbb{E}[\max\{0, y_k - w_k\}] + \mathbb{E}[J_{k+1}(y_k - w_k)].$$

Now, since w_k is drawn from F_1 with probability q_k and from F_2 with probability F_2 we can apply the law of total probabilities, leading to

$$\begin{aligned} G(y_k) = cy_k &+ q_k(h\mathbb{E}_{w_k|w\sim F_1}[\max\{0, w_k - y_k\}] + p\mathbb{E}_{w_k|w\sim F_1}[\max\{0, y_k - w_k\}] + \mathbb{E}_{w_k|w\sim F_1}[J_{k+1}(y_k - w_k)])+ \\ &+(1-q_k)(h\mathbb{E}_{w_k|w\sim F_2}[\max\{0, w_k - y_k\}] + p\mathbb{E}_{w_k|w\sim F_2}[\max\{0, y_k - w_k\}] + \mathbb{E}_{w_k|w\sim F_2}[J_{k+1}(y_k - w_k)]). \end{aligned}$$

We saw in class that $cy_k + h\mathbb{E}_{w_k|w\sim F_i}[\max\{0, w_k - y_k\}] + p\mathbb{E}_{w_k|w\sim F_i}[\max\{0, y_k - w_k\}] + \mathbb{E}_{w_k|w\sim F_i}[J_{k+1}(y_k - w_k)]$ is convex, since we have a sum of convex, our $G(y_k)$ will also be convex. So, there exists a S_k that will represent the optimal stock we seek at period k. However, S_k could be smaller than x_k, so it would not be reachable (in which case we would not by stock). Then, the policy will be

$$\mu_k^*(x_k) = \begin{cases} S_k - x_k & \text{if } S_k > x_k \\ 0 & \text{otherwise.} \end{cases}$$

Q5

(a) To find the policy that will minimize the cost function in the worst case. We can write a DP-like algorithm to solve this problem as

$J_N(x_N) = g_N(x_N)$

$J_k(x_k) = \min_{\mu_k \in U_k} \max_{w_k \in W_k(x_k, \mu_k(x_k))} [g_k(x_k, \mu_k(x_k), w_k) + J_{k+1}(x_{k+1})]$, where $x_{k+1} = f_k(x_k, \mu_k(x_k), w_k)$ $\forall k = 0, \ldots, N-1$.

(b) In this problem the state x_k needs to belong to a certain set X_k at each state k, so we have to choose a policy that will assure that this will happen. We write the dynamics in a general form like $x_{k+1} = f(x_k, \mu_k(x_k)) + h_k(w_k)$.

We consider a cost of the policy $g(\mu_k(x_k))$ and the cost of being outside X_k infinite to ensure we stay whenever possible there.

DP algorithm:

$$J_N(x_N) = \begin{cases} 0 & \text{if } x_N \in X_N \\ \infty & \text{otherwise} \end{cases}$$

$$J_{k+1} = \min_{\mu_k(x_k) \in U_k} \max_{w_k \in W_k(x_k, \mu_k(x_k))} \begin{cases} g_k(\mu(x_k)) + J_{k+1}(x_{k+1}) & \text{if } x_k \in X_k \\ \infty & \text{otherwise} \end{cases},$$

where $x_{k+1} = f_k(x_k, \mu_k(x_k)) + h_k(w_k)$. Then, we will need to compute recursively the $\bar{X}_k$ that allow to continue in the sets $X_{k+1}, \ldots, X_N$. In order to do so, x_k will need to be in a certain set Y_{k+1} such that for any $w_k \in W_k$, $f(x_k, \mu_k(x_k)) + h(w_k)$ belong to $\bar{X}_{k+1}$. Initializing $\bar{X}_N = X_N$, $\forall k = 1, \ldots, N-1$, the recursion would be the next:

$$Y_{k+1} = \{z \in \mathcal{X} : z + h_k(w_k) \in \bar{X}_{k+1}, \forall w_k \in W_k(x_k, \mu_k(x_k))\}$$

$$\bar{X}_k = \{x \in X_k : \exists \mu_k(x) \in U_k \text{ such that } f_k(x, \mu_k(x_k)) \in Y_{k+1}\}$$