Decision Making under Uncertainty
ASEN 6519-007
March 31, 2020

Maggie Jacoby

Homework 5

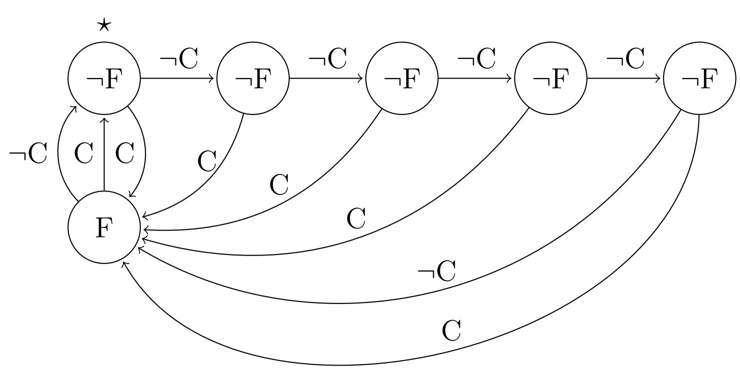
Conceptual Questions

Problem 1: Optimal Policy Graph for Crying Baby POMDP

Given
Consider a modified version of the Crying Baby POMDP, exactly as described in the book, except that the baby has a 20% chance of becoming hungry at the next time step if it is not currently hungry and only a 60% chance of crying when hungry. The alpha vectors for the optimal policy are given. The point at which the alpha vectors for feeding and not feeding intersect is at $P(\texttt{hungry}) = 0.19$.

Draw
An optimal policy graph with appropriate action and observation labels if the initial belief is certainty that the baby is not hungry ($b_0(\texttt{hungry}) = 0$).

Solution



Exploratory code for Question 1

```
using POMDPs
using POMDPModels
using POMDPModelTools
using BeliefUpdaters
using POMDPPolicies
using POMDPSimulators
using QuickPOMDPs
using BeliefUpdaters

#Crying Baby Problem
r_feed = -5.0
r_hungry = -10.0
p_become_hungry = 0.2
p_cry_when_hungry = 0.6
p_cry_when_not_hungry = 0.1
γ = 0.9
m = BabyPOMDP(r_feed, r_hungry,
              p_become_hungry,
              p_cry_when_hungry,
              p_cry_when_not_hungry,
              γ)

updater = DiscreteUpdater(m);
alpha_v = 0.19

cry_dict = Dict(true => "crying", false => "not crying")
act_dict = Dict(true => "feed", false => "don't feed")

function run_scenario(up, m, T)
    b = initialize_belief(up, Deterministic(false));
    o = false;
    for t in 1:T
        display("time: $t")
        p_hungry = b.b[2]
        a = act(p_hungry)
        display("hungry: $(round(p_hungry, digits=2))")
        display("action: $(act_dict[a])")
        display("observation: $(cry_dict[o])")
        println();
        b = update(up, b, a, o)
        o = rand(cry_dict).first
     end
end

function act(b, alpha = alpha_v)
    if b < alpha
        return false
    else
        return true
    end
end

run_scenario(updater, m, 20)
```

Exercises

Problem 2: Cancer POMDP

Given A cancer treatment plan where the states are defined as below, and the transition and observation probabilities are given.

$$
\begin{aligned}
\mathcal{S} &= \{\texttt{healthy}, \texttt{in-situ-cancer}, \texttt{invasive-cancer}, \texttt{death}\} \\
\mathcal{A} &= \{\texttt{wait}, \texttt{test}, \texttt{treat}\} \\
\mathcal{O} &= \{\texttt{positive}, \texttt{negative}\} \\
\gamma &= 0.99 \\
s_0 &= \texttt{healthy}
\end{aligned}
$$

Find

(a) Use Monte Carlo simulations to evaluate a policy that always **wait**s.

(b) Propose a better heuristic strategy based on the observation history or belief and evaluate it with Monte Carlo simulations. See if you can get an average discounted return of 75 or more.

Solution

(a) Given the "always wait" policy, I found the average reward to be **40.5**. This compares to an average reward for an "always test" policy of 32.5.

(b) A better policy I found was dictated by belief, and is described below:

- If belief of *invasive-cancer* is greater 2%, then **treat**.
- If belief of *in-situ-cancer* is greater than 25%, then **treat**.
- If belief of *in-situ-cancer* is greater 1% (but less than 25%), then **test**.
- Otherwise, do nothing (**wait**).

Using this policy I was able to achieve an average reward of **78.2**.

Code for Question 2

```
using POMDPs
using POMDPModels
using POMDPModelTools
using BeliefUpdaters
using POMDPPolicies
using POMDPSimulators
using QuickPOMDPs
using BeliefUpdaters
include("DMU_HW5_Q2_probs.jl")


S = [:healthy, :in_situ, :invasive, :death]
A = [:wait, :test, :treat]
O = [:pos, :neg]
γ = 0.99
s0 = Deterministic(:healthy)
term = Set([:death])

m = DiscreteExplicitPOMDP(S, A, O, T, Z, R, γ, s0, terminals=term)


wait_policy = FunctionPolicy(
    function (o)
        return :wait
    end)


function simulate_policy(m, N, policy)
    up = DiscreteUpdater(m);
    rsum = 0.0
    for i in 1:N
        sim = RolloutSimulator(max_steps=1000)
        rsum += simulate(sim, m, policy, up)
    end
    return rsum/N
end;

p = FunctionPolicy(
    function (b)
        if b.b[3] > 0.02
            return :treat
        elseif b.b[2] > 0.25
            return :treat
        elseif b.b[2] > 0.01
            return :test
        else
            return :wait
        end
end);

N = 100000
avg_reward = simulate_policy(m, N, p)
```

```
## transition and observation dynamics for DMU HW 5 Problem 2


function T(s, a, sp)
    if s == :healthy
        if sp == :in_situ
            return 0.02
        elseif sp == s
            return 0.98
        else
            return 0.0
        end
    elseif s == :in_situ
        if a == :treat
            if sp == :healthy
                return 0.60
            elseif sp == s
                return 0.40
            else
                return 0.0
            end
        else #a == :test || a == :wait
            if sp == :invasive
                return 0.10
            elseif sp == s
                return 0.90
            else
                return 0.0
            end
        end
    elseif s == :invasive
        if a == :treat
            if sp == :healthy
                return 0.20
            elseif sp == :death
                return 0.20
            elseif sp == s
                return 0.60
            else
                return 0.0
            end
        else
            if sp == :death
                return 0.60
            elseif sp == s
                return 0.40
            else
                return 0.0
            end
        end
    else # s == :death
        return 0.25
    end
end
```

```
function Z(a, sp, o)
    if a == :test
        if sp == :healthy
            if o == :pos
                return 0.05
            else
                return 0.95
            end
        elseif sp == :in_situ
            if o == :pos
                return 0.80
            else
                return 0.20
            end
        else
            if o == :pos
                return 1.0
            else
                return 0.0
            end
        end
    elseif a == :treat
        if sp == :in_situ || sp == :invasive
            if o == :pos
                return 1.0
            else
                return 0.0
            end
        else
            if o == :pos
                return 0.0
            else
                return 1.0
            end
        end
    else #a == :wait
        if o == :pos
            return 0.0
        else
            return 1.0
        end
    end
end



function R(s, a)
    if s == :death
        return 0.0
    elseif a == :wait
        return 1.0
    elseif a == :test
        return 0.80
    else
        return 0.10
    end
end
```