Stanford University

AA228/CS238: Decision Making under Uncertainty

Autumn 2019

Prof. Mykel J. Kochenderfer • Durand 255 • email: *mykel@stanford.edu*

MIDTERM 3 **Due date: November 22**

You have until 5pm on November 22 to complete this take-home midterm. **Because this is an exam, there are not late hours permitted. Please plan your time accordingly.** You may use any resources available to you (e.g., book, notes, internet, software). You may not consult with others inside or outside of the class on these specific questions; doing so would be a violation of the Stanford Honor Code. Please upload a **PDF** of your solutions to `gradescope.com`. Your solutions can be written in the programming language of your choice and you may use any packages you want. Each question builds upon the previous ones. No questions about the take-home midterm or associated concepts are allowed in office hours until after November 22 and all Piazza posts will default to private and should be private until November 22.

Optimization of qualification exam administration

A department of Decision Science at a fictional university is studying how to best optimize the administration of Ph.D. qualification exams (referred to as quals), and they asked us to use the tools from our class to help them. Quals consists of a set of subject exams, all taken during the same week. To pass quals and be admitted into the Ph.D. program, the candidate must demonstrate that they are properly qualified (meaning they have mastered the material) in all subject exams. If the faculty panel decides the candidate is not qualified, they may require that the candidate retake one or more exams during the next quals session (typically 6 months later). Candidates are permitted to take exams in at most a fixed number of sessions. If the student needs to retake one or more exams, they can spend time studying the relevant material to improve the chance they master the material. An exam is not a perfect indicator of whether the candidate is qualified in a particular subject area; sometimes students who have mastered the material fail the exam (e.g., due to nervousness) and sometimes students who have not mastered the material pass (e.g., because the questions do not provide full coverage of the subject material).

Since this is a sequential decision problem with both state and outcome uncertainty, we model this problem as a POMDP with the following parameters (default values are provided):

- $h = 2$, maximum number of exam attempts
- $n = 3$, number of exams per session
- $\rho = 0.3$, model parameter for transition from unqualified to qualified (more details later)
- $r_{\text{ff}} = -1$, reward for false fail (permanently failing a qualified candidate on their last allowed exam attempt, preventing them from ever entering the Ph.D program)
- $r_{\text{fp}} = -1$, reward for false pass (passing the unqualified candidate to enter the Ph.D. program)
- $p_{\text{ff}} = 0.15$, probability of a false fail on a particular subject exam (student is qualified but fails)
- $p_{\text{fp}} = 0.2$, probability of a false pass on a particular subject exam (student is not qualified but passes)
- $p_q = 0.9$, prior probability that a student is qualified in a particular subject prior to the start of exams

If the candidate is qualified in a subject, they stay qualified. If the candidate is not qualified in a subject, they can become qualified with some probability related to the number of exams they have to study for (and take in the next session). The probability that they become qualified is ρ divided by the number of exams they have to study for. If they are not forced to retake a subject exam, they will remain unqualified if they are currently unqualified. In this simple model, the dynamics of the state variables are independent of each other given the action.

Here is an example scenario. Johnny has to take exams in three areas: data mining, optimization, and risk analysis. At timestep 1, he starts out being qualified in data mining, but not optimization and risk analysis. He studies hard and at timestep 2 he becomes qualified in data mining and optimization but still

is not qualified in risk analysis. At this timestep (timestep 2), he takes the three exams and passes data mining but fails optimization (due to nervousness) and risk analysis (because of lack of understanding the concepts). He is asked to take all the exams again by the examiners. He studies really hard, and at timestep 3, he transitions to becoming qualified in all three areas. He takes the exam and passes all three. Faculty say no more examining to be done, and welcome him as a Ph.D. student.

Question 1. Show code for defining a problem structure (e.g., *struct* in Julia or Matlab, *class* in Python) with the parameters defined above. [1pt] Show the creation of an instance of this structure with the default values populated. [1pt]

Solution:

```
struct Problem
    h   # max number of exam session tries
    n   # number of exams per session
    ρ   # model parameter for transitioning from unqualified to qualified
    rff # reward for false fail
    rfp # reward for false pass
    pff # probability of false fail
    pfp # probability of false pass
    pq  # prior probability of being qualified
end
p = Problem(2, 3, 0.3, -1, -1, 0.15, 0.2, 0.9)
```

Question 2. Define functions for creating the state space, action space, and observation space. The functions should take the problem definition as input. Elements in all three of these sets should be vectors of length n, and each component corresponds to a particular subject exam.

1. For states, the components are either 0 (not qualified) or 1 (qualified).
2. For actions, the components are either 0 (do not retake) or 1 (retake).
3. For observations, the components are either -1 (not observed, since exam not taken in that session), 0 (failed subject exam), or 1 (passed subject exam).

Show these functions. [1pt] Hint: you'll want `length(observations(p))` to be 27.
If working with Julia, you may find the following function helpful:

```
space(x, n) = vec(collect(Iterators.product([x for i = 1:n]...)))
```

If working with Python, you may find the following function helpful:

```
def space(x,n): return [i for i in itertools.product(x, repeat = n)]
```

Solution:

```
space(x, n) = vec(collect(Iterators.product([x for i = 1:n]...)))

states(p::Problem) = space([0,1], p.n) # 0 = not qualified, 1 = qualified
actions(p::Problem) = space([0,1], p.n) # 0 = don't retake, 1 = retake
observations(p::Problem) = space([-1,0,1], p.n); # -1 = not observed, 0 = fail, 1 = pass
```

Question 3. Write a transition function that takes as input the problem p, state s, action a, and next state s' and returns $T(s' \mid s, a)$. [2pt]
Hint: you'll want `transition(p, [0,1,0], [1,1,0], [1,1,0])` to be 0.15.

Solution: We can multiply because the action effects are independent across subject areas.

```
function transition(p::Problem, s, a, s′)
    r = 1.0
    learnprob = p.ρ/sum(a)
    for i = 1:p.n
        if s[i] == 1 && s′[i] == 0
```

```
            # unlearned a subject (never allowed)
            return 0.0
        elseif s[i] == 0 && s′[i] == 0 && a[i] == 1
            # didn't master a subject
            r *= 1.0 - learnprob
        elseif s[i] == 0 && s′[i] == 1 && a[i] == 1
            # mastered a subject
            r *= learnprob
        elseif a[i] == 0
            # not testing a subject (not allowed to change)
            r *= (s[i] == s′[i])
        end
    end
    return r
end
```

Question 4. Time can be 1, 2, or 3. At time step 1, the state relates to the candidate's mastery prior to studying and the action corresponds to which subject exams to take (e.g., action $[1, 1, 1]$ would indicate that the candidate is to take all three subject exams while an action $[0, 0, 1]$ would indicate that the candidate will only take the third subject exam and the exam committee already recognizes the student has mastered the first two subjects). Time step 2 involves a transition to a new state (after studying), and an observation of whether the student passed or failed the various subject exams. Based on this observation, the faculty can take a new action at that time step. If the faculty take action $[0, 0, 0]$, then the candidate has passed quals and no additional exams are needed. Time step 3 involves transitioning to a new state (based on the outcome of the studying) and a new observation of which exams the candidate passed. The final action (at this time step) indicates whether the candidate has passed quals. If any component of that action is 1, then the student has failed and can no longer make any attempts.

Show your code for a reward function that takes as input the problem p, the state s, the action a, and time t. [2pt]

Hint: you'll want `reward(p, [1,0,0], [0,0,0], 3)` to be -1.

Solution:

```
function reward(p::Problem, s, a, t)
    if t == p.h + 1 && prod(s) == 1 && sum(a) > 0
        # false fail
        # student is qualified, we tell them they need to retake, but they don't have more tries
        return p.rff
    elseif prod(s) != 1 && sum(a) == 0
        # false pass
        # student is not qualified any we tell them they don't need to retake
        return p.rfp
    end
    return 0.0
end
```

Question 5. Show a function that takes as input your problem p, action a, next state s', and observation o. It should output $O(o \mid s', a)$. [2pt]

Hint: `sum(observation(p, [1,1,1], [1,1,1], o) for o in observations(p))` should be 1 or close to 1.

Solution: We return 0 for impossible transitions.

```
function observation(p::Problem, a, s′, o)
    r = 1.0
    for i = 1:p.n
        if a[i] == 1 && o[i] == -1
            return 0.0
        end
        if a[i] == 0 && o[i] != -1
            return 0.0
```

```
        end
        if a[i] == 1 # if we're observing this test
            if s′[i] == 0
                if o[i] == 0
                    r *= 1 - p.pfp # true fail P(not pass|not qualified)
                else
                    r *= p.pfp     # false pass P(pass|not qualified)
                end
            else # s′ == 1
                if o[i] == 0
                    r *= p.pff     # false fail P(not pass|qualified)
                else
                    r *= 1 - p.pff # true pass P(pass|qualified)
                end
            end
        end
    end
    return r
end
```

Question 6. Show a function that takes as input the problem p and outputs the initial belief vector. [2pt] Display the numerical values within this belief vector. [1pt]

Hint: you'll want: `length(initbelief(p))` to be 8.

Solution:

```
function initbelief(p::Problem)
    S = states(p)
    b = zeros(length(S))
    for (i, s) in enumerate(S)
        numqual = sum(s)
        b[i] = p.pq^numqual * (1-p.pq)^(p.n - numqual)
    end
    return b
end
print(initbelief(p))
```

```
[0.001, 0.009, 0.009, 0.081, 0.009, 0.081, 0.081, 0.729]
```

Question 7. Show a belief update function that takes as input the problem p, the current belief b, the action a and resulting observation o. It should output the updated belief vector. [2pt]

- Print the belief vector that results after taking action $[1, 1, 1]$ at the first time step and observing $[1, 1, 1]$. [1pt]
  Hint: the last component should be near 0.933 (accurate to the stated precision).
- Print the belief vector that results after taking action $[1, 1, 1]$ at the first time step and observing $[1, 1, 0]$. [1pt]

Solution:

```
# eq. 6.11 in DMU
function update(p::Problem, b, a, o)
    b′ = similar(b)
    S = states(p)
    for (i, s′) in enumerate(S)
        b′[i] = observation(p, a, s′, o)
        b′[i] *= sum(transition(p, s, a, s′)*b[j] for (j, s) in enumerate(S))
    end
    return b′ ./ sum(b′)
end
print(update(p, initbelief(p), [1,1,1], [1,1,1]))
```

```
[1.18e-5, 0.000505, 0.000505, 0.0217, 0.000505, 0.0217, 0.0217, 0.933]

print(update(p, initbelief(p), [1,1,1], [1,1,0]))

[0.000179, 0.00767, 0.00767, 0.33, 0.000339, 0.0145, 0.0145, 0.625]
```

Question 8. From our initial belief state, what is the expected immediate reward for each of the eight possible actions? Please show the action and the reward. [2pt]

Solution:

```
b = initbelief(p)
for a in actions(p)
    v =  b'*[reward(p, s, a, 1) for s in states(p)]
    println("$a => $v")
end
```

```
(0, 0, 0) => -0.271
(1, 0, 0) => 0.0
(0, 1, 0) => 0.0
(1, 1, 0) => 0.0
(0, 0, 1) => 0.0
(1, 0, 1) => 0.0
(0, 1, 1) => 0.0
(1, 1, 1) => 0.0
```

Question 9. Suppose $t = 3$. Show a function that is the reward associated with entering state s and observing o given that we fail the student if o indicates they did not pass all their subjects they had to take (and pass them otherwise)? [2pt]

Hint: we want `lastreward(p, [1,1,1], [1,1,-1])` to be 0.

Solution:

```
function lastreward(p, s, o)
    if prod(o) == 0 # failed at least one
        return reward(p, s, ones(p.n), 3) # any nonzero action component leads to fail
    else
        return reward(p, s, zeros(p.n), 3)
    end
end
```

Question 10. Suppose $t = 2$. Show a function that computes the expected utility after entering state s and observing o given that we have them retake all exams if they fail any of them. [2pt] Show the result when $s = [1, 1, 1]$ and $o = [1, 1, 0]$. [1pt]

Solution:

```
function retakeall_utility(p, s, o)
    if prod(o) != 0 # passed all taken
        return reward(p, s, zeros(p.n), 2)
    end
    a = ones(p.n)
    S = states(p)
    O = observations(p)
    u = reward(p, s, a, 2)
    for s′ in S
        x = 0.0
        for o in O
            x += observation(p, a, s′, o) * lastreward(p, s′, o)
        end
        x *= transition(p, s, a, s′)
```

```
        u += x
    end
    return u
end

print(retakeall_utility(p, [1,1,1], [1,1,0]))
```

```
-0.386
```

Question 11. Show a function [2pt] that computes the alpha vector associated with the following policy:

- At $t = 1$, have the student take all exams.
- At $t = 2$, have the student take all exams if they failed any (including ones they may have passed).
- At $t = 3$, take action $[0, 0, 0]$ if they pass all their exams; otherwise, take $[1, 1, 1]$.

Print the alpha vector. [1pt] Using this alpha vector, compute the expected utility from the initial belief state. [1pt]

Solution:

```
function retakeall_utility(p, s)
    O = observations(p)
    a = ones(p.n)
    u = reward(p, s, a, 1)
    for s′ in states(p)
        x = 0.0
        for (k, o) in enumerate(O)
            x += observation(p, a, s′, o) * retakeall_utility(p, s′, o)
        end
        u += transition(p, s, a, s′) * x
    end
    return u
end
retakeall_alpha(p) = [retakeall_utility(p, s) for s in states(p)]
print(retakeall_alpha(p))
```

```
[-0.0488, -0.126, -0.126, -0.275, -0.126, -0.275, -0.275, -0.149]
```

```
b = initbelief(p)
print(retakeall_alpha(p)'*b)
```

```
-0.179
```

Question 12. Suppose we only have the student retake the exams that they fail at each timestep. Show your code [2pt] and compute the alpha vector associated with this policy [1pt]. Compute the expected utility of this policy when starting from our initial belief state. [1pt]

Solution:

```
function retakefailed_utility(p, s, o)
    if prod(o) != 0 # passed all taken exams
        return reward(p, s, zeros(p.n), 2)
    end
    a = [oi == 0 ? 1 : 0 for oi in o]
    S = states(p)
    O = observations(p)
    u = reward(p, s, a, 2)
    for s′ in S
        x = 0.0
        for o in O
```

```
            x += observation(p, a, s′, o) * lastreward(p, s′, o)
        end
        x *= transition(p, s, a, s′)
        u += x
    end
    return u
end

function retakefailed_utility(p, s)
    O = observations(p)
    a = ones(p.n)
    u = reward(p, s, a, 1)
    for s′ in states(p)
        x = 0.0
        for o in O
            x += observation(p, a, s′, o) * retakefailed_utility(p, s′, o)
        end
        u += transition(p, s, a, s′) * x
    end
    return u
end
retakefailed_alpha(p) = [retakefailed_utility(p, s) for s in states(p)]
print(retakefailed_alpha(p))
```

```
[-0.125, -0.229, -0.229, -0.312, -0.229, -0.312, -0.312, -0.066]
```

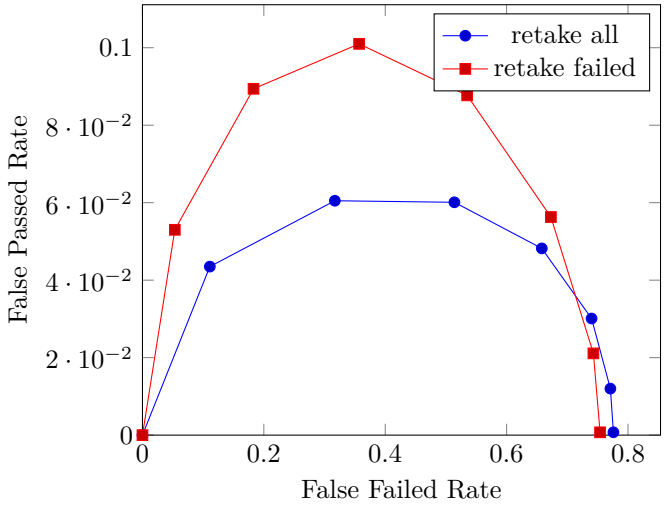
```
b = initbelief(p)
print(retakefailed_alpha(p)'*b)
```

```
-0.13
```

Question 13. Extra credit. Suppose we set $p_{\text{ff}} = p_{\text{fp}} = \theta$. As we vary θ (which governs individual subject exams), we will get different false passed rates and false failed rates for the quals. Show your code for computing these rates for the two policies (i.e., retake all and retake failed) [1pt]. Make a plot that illusrates the tradeoff between the false passed rate and the false failed rate for the two policies [1pt].

Solution:

```
nsamples = 8
ffretakeall = zeros(nsamples)
fpretakeall = zeros(nsamples)
ffretakefailed = zeros(nsamples)
fpretakefailed = zeros(nsamples)
b = initbelief(p)
for (i, θ) in enumerate(range(0, stop=1, length=nsamples))
    p = Problem(2, 3, 0.3, 1, 0, θ, θ, 0.9)
    ffretakeall[i] = retakeall_alpha(p)'*b
    ffretakefailed[i] = retakefailed_alpha(p)'*b
    p = Problem(2, 3, 0.3, 0, 1, θ, θ, 0.9)
    fpretakeall[i] = retakeall_alpha(p)'*b
    fpretakefailed[i] = retakefailed_alpha(p)'*b
end
a = Axis([
    Plots.Linear(ffretakeall, fpretakeall, legendentry="retake all"),
    Plots.Linear(ffretakefailed, fpretakefailed, legendentry="retake failed"),
], xlabel="False Failed Rate", ylabel="False Passed Rate", xmin=0, ymin=0)
plot(a)
```



Question 14. Optional. Write a little rhyme about what you learned in this class. A selection of clever ones will be compiled and shared with the class. By default they will be shared anonymously, but if you would like your name associated with it, please indicate. [0pt]