EEE 598 Fall 2021
HW2

Jacob Sindorf

Homework 2

Assistant Professor Nicolò Michelusi

Office: GWC 330

In your submission, please include:

- printout of Matlab scripts (pdf) that you created and .m files
- printout of figures
- discussion and steps as requested
- all printouts (solutions, coding parts, figures and discussions) should be included in a SINGLE pdf file.

Please, be clear, concise and organized, and make sure your hand-writing is readable. Make sure that your Matlab code is clearly organized, and provide sufficient descriptions to help me follow your reasoning.

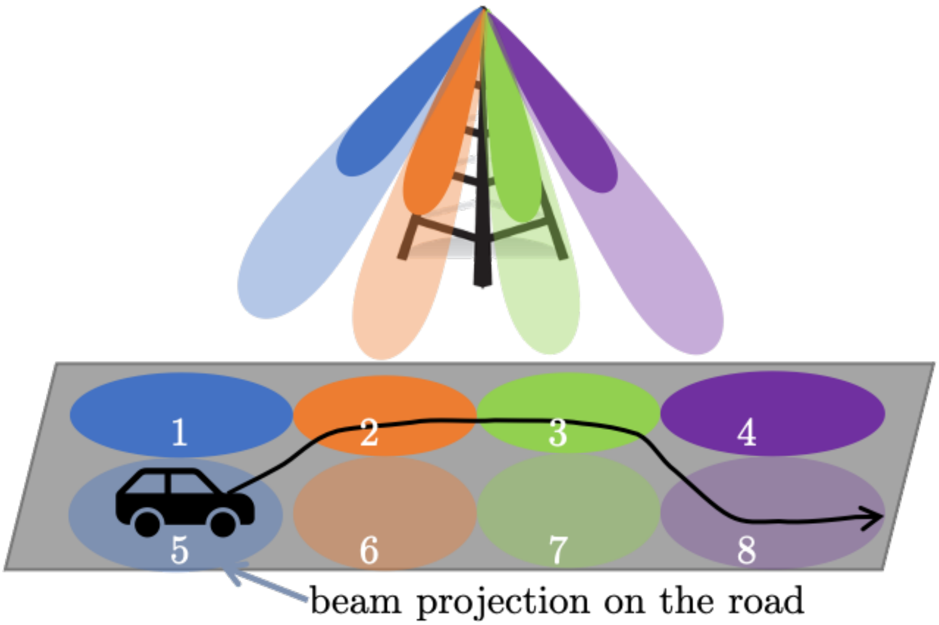
NOTE: unless you have already started coding this homework, you are required to solve all the coding parts using Matlab. In future HWs, only Matlab is allowed.

I. POMDP

Consider the following problem:

- A car travels along a road served by a base station on the road side, and communicates using millimeter wave technology.

- The BS uses directional beams to communicate with the car. For instance, if the car is located under beam number 5, then the BS should use beam #5 to transmit data to the car. If the correct beam is used by the BS to transmit data, B bits are successfully transmitted from the BS to the car. If the wrong beam is used, transmission is unreliable and no data goes through.



- Let S_k be the sector the car is currently located in at timeslot k and assume that there are only two sectors, so that $S_k \in \mathcal{S} \equiv \{1,2\}$; due to its mobility, S_k evolves over time. Assume that S_k follows a Markov process. Let $q = \mathbb{P}(S_{k+1} = 2|S_k = 1) = \mathbb{P}(S_{k+1} = 1|S_k = 2)$ be the probability of exiting the current sector and entering the other one in one timeslot, so that

$$\mathbb{P}(S_{k+1} = 1|S_k = 1) = \mathbb{P}(S_{k+1} = 2|S_k = 2) = 1 - q.$$

- The BS can select either data transmission actions 1 (transmit on sector 1) and 2 (transmit on sector 2), or a *beam training* action 0.

- If action 1 is selected, no feedback signal is collected ($Y_k = 0$), and B bits are delivered successfully if and only if $S_k = 1$ (i.e, the car is located in the sector that the BS is transmitting to); if $S_k = 2$, the transmission fails and no data is delivered.
- If action 2 is selected, no feedback signal is collected ($Y_k = 0$), and B bits are delivered successfully if and only if $S_k = 2$ (i.e, the car is located in the

sector that the BS is transmitting to); if $S_k = 1$, the transmission fails and no data is delivered.

- If action 0 is selected in slot k, no bits are delivered, but a feedback signal $Y_k \in \{1, 2\}$ is generated, indicating which sector the car is located in. However, this feedback signal may be erroneous. Let $\epsilon = \mathbb{P}(Y_k = 1|S_k = 2, U_k = 0) = \mathbb{P}(Y_k = 2|S_k = 1, U_k = 0)$ be the probability that the beam training action $U_k = 0$ generates an erroneous feedback signal.

The goal is to maximize the average amount of bits per stage transmitted by the BS to the car (approximated as a finite horizon problem with N large),

$$\lim_{N\to\infty} \frac{1}{N} \mathbb{E}[\sum_{k=0}^{N-1} B_k],$$

where B_k is the amount of bits successfully delivered in stage k.

a. b. c. d. e.

1) Characterize the state space, actions, observations, state transition and observation probabilities, and reward f. metric $r(i, u)$ as a function of state and action pairs.

a) State Space

$S_k \in \{1, 2\}$ sector vehicle is located

b) Action Space

$U_k \in \{0, 1, 2\}$

0: beam traning

1: sector 1 data transm.

2: sector 2 data transm.

c) Observations

$$Y_k = \begin{cases} 0 & u \in \{1, 2\} \\ \{1, 2\} & u = 0 \end{cases}$$

No feedback

location of car

d) State Transitions and Dynamics

$$\mathbb{P}(s_{k+1}=j \mid s_k=i, U_k=u)$$

Vehicle moves freely so independent of action

$$\mathbb{P}(s_{k+1}=1 \mid s_k=1) = 1-q$$

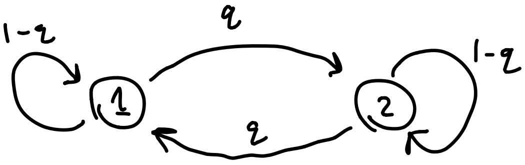
$$\mathbb{P}(s_{k+1}=2 \mid s_k=1) = q$$

$$\mathbb{P}(s_{k+1}=2 \mid s_k=2) = 1-q$$

$$\mathbb{P}(s_{k+1}=1 \mid s_k=2) = q$$

General

$$\mathbb{P}(s_{k+1}=j \mid s_k=i) \begin{cases} 1-q & j=i \\ q & j\neq i \end{cases}$$



e) Observation Model

$$\mathbb{P}(s_{k+1}=j, Y_k=y \mid s_k=i, V_k=u)$$

$$\Rightarrow \underbrace{\mathbb{P}(Y_k=y \mid , s_k=i, U_k=u)}_{\text{• need to define this the obs. model}} \cdot \underbrace{\mathbb{P}(s_{k+1}=j \mid s_k=i)}_{\text{transition model (state dynamics)}}$$

- pres. $Y_k=y$ does not depend on next state

- look at ε

<u>$u=1$</u>

$$\mathbb{P}(Y_k=y \mid s_k=i, U_k=1) = \begin{cases} 1, & y=0 \\ 0, & y\neq 0 \end{cases}$$

<u>$u=2$</u>

$$\mathbb{P}(Y_k=y \mid s_k=i, U_k=2) = \begin{cases} 1, & y=0 \\ 0, & y\neq 0 \end{cases}$$

<u>$u=0$</u>

$$\mathbb{P}(Y_k=y \mid s_k=i, U_k=0) = \begin{cases} 0, & y=0 \\ 1-\varepsilon, & y=i \\ \varepsilon, & y\neq i \end{cases}$$

f) Reward

maximize data delivery $\Rightarrow$ reward

$r(s,u)$ go action & state

$$r(s,0) = 0 \quad (u=0), \forall s \in \{1,2\}$$

$$r(s,1) = \begin{cases} B & \text{if} \quad s=1 \quad \text{bits transferred} \\ 0 & \text{if} \quad s=2 \quad \text{failed} \end{cases}$$

$$r(s,2) = \begin{cases} 0 & \text{if} \quad s=1 \\ B & \text{if} \quad s=2 \end{cases}$$

<u>General</u>

$$r(i,u) = \begin{cases} B & i=u \\ 0 & i \neq u \\ 0 & u=0 \end{cases}$$

2) Characterize the belief update function $B(\beta(1), \beta(2), u, y)$, i.e. how the belief β is updated after selecting action u and observing y.

Belief Update

$k=0$: initial belief $= \beta_0$

At time k, given β_k, the controller selects a generic action, $U_k = u$ and then observes $Y_k = y$

How to compute the new belief β_{k+1}?

$$\beta_{k+1} = \mathbb{B}\left(\beta_k, U_k, Y_k\right)$$

$$\beta_{k+1} = \begin{bmatrix} \beta_{k+1}(1) \\ \beta_{k+1}(2) \end{bmatrix}$$

$$\beta_{k+1}(j) = \mathbb{P}\left(\underbrace{s_{k+1}=j}_{A} \;\middle|\; \underbrace{s_k \sim \beta_k, U_k=u, Y_k=y}_{B}\right)$$

Belief Update eqn

$$(*) \quad \beta_{k+1}(j) = \frac{\sum_{i=1}^{2} \beta_k(i)\, \mathbb{P}\left(Y_k=y \mid s_k=i, U_k=u\right) \cdot \mathbb{P}\left(s_{k+1}=j \mid s_k=i\right)}{\sum_{i=1}^{2} \beta_k(i)\, \mathbb{P}\left(Y_k=y \mid s_k=i, U_k=u\right)}$$

- using cond. prob.

$$\mathbb{P}(A \mid B) = \frac{\mathbb{P}(A, B)}{\mathbb{P}(B)}$$

Specialize for all possible actions

$$\boxed{\begin{matrix} \underline{u=1} \\ \underline{u=2} \end{matrix}}$$

with prob. $=1$, $Y_k=0$ $(y=0)$

$=1$, $Y_k=0$ $(y=0)$, so these actions look the same

$$\beta_{k+1}(j) = \frac{\sum_{i=1}^{2} \beta_k(i) \overset{1}{\cancel{\mathbb{P}(Y_k=y \mid s_k=i, U_k=u)}} \cdot \mathbb{P}(s_{k+1}=j \mid s_k=i)}{1 \; \underbrace{\cancel{\sum_{i=1}^{2} \beta_k(i)}}_{\text{belief sums to 1!}} \overset{1}{\cancel{\mathbb{P}(Y_k=y \mid s_k=i, U_k=u)}}}$$

$\boxed{\text{* do sum over } i}$

$$\beta_{k+1}(j) = \sum_{i=1}^{2} \beta_k(i) \cdot \mathbb{P}(s_{k+1}=j \mid s_k=i)$$

$$= \beta_k(1)\mathbb{P}(s_{k+1}=j \mid s_k=1) + \beta_k(2)\mathbb{P}(s_{k+1}=j \mid s_k=2)$$

* $\boxed{\underline{j=1}}$

$$\beta_{k+1}(1) = \beta_k(1)\overset{1-q}{\mathbb{P}(s_{k+1}=1 \mid s_k=1)} + \beta_k(2)\overset{q}{\mathbb{P}(s_{k+1}=1 \mid s_k=2)}$$

$$\beta_{k+1}(1) = \beta_k(1)(1-q) + \beta_k(2)(q)$$

* $\boxed{\underline{j=2}}$

$$\beta_{k+1}(2) = \beta_k(1)\overset{q}{\mathbb{P}(s_{k+1}=2 \mid s_k=1)} + \beta_k(2)\overset{1-q}{\mathbb{P}(s_{k+1}=2 \mid s_k=2)}$$

$$\beta_{k+1}(2) = \beta_k(1)(q) + \beta_k(2)(1-q)$$

$U=2$ similar to $u=1$

$$\beta_{k+1}(1) = \beta_k(1)(1-q) + \beta_k(2)(q) \qquad (j=1)$$

$u \in \{1,2\}$
$y = 0$

$$\beta_{k+1}(2) = \beta_k(1)(q) + \beta_k(2)(1-q) \qquad (j=2)$$

belief updates for actions $u \in \{1, 2\}$

for $B(\beta_k, u, 0)$ $\quad y = 0$

$\boxed{\underline{u=0}}$ $\quad Y_k = S_k$ with prob $1-\varepsilon$

$Y_k \neq S_k \qquad \varepsilon$

Case $\boxed{\underline{y=1}}$

$$\beta_{k+1}(j) = \frac{\sum_{i=1}^{2} \beta_k(i) \mathbb{P}(Y_k=1 \mid s_k=i, U_k=0) \cdot \mathbb{P}(s_{k+1}=j \mid s_k=i)}{\underbrace{\sum_{i=1}^{2} \beta_k(i) \mathbb{P}(Y_k=1 \mid s_k=i, U_k=0)}_{\beta_k(1)(1-\varepsilon)+\beta_k(2)(\varepsilon)}}$$

* look at i, & do sums

$i=1$, $Y_k=1 \mid s_k=1, U_k=0 \quad = 1-\varepsilon$

$i=2 \quad Y_k=1 \mid s_k=2, U_k=0 \quad = \varepsilon$

$$= \frac{\beta_k(2)(1-\varepsilon)\mathbb{P}(s_{k+1}=j \mid s_k=1) + \beta_k(2)(\varepsilon)\mathbb{P}(s_{k+1}=j \mid s_k=2)}{\beta_k(1)(1-\varepsilon)+\beta_k(2)(\varepsilon)}$$

* now look at j

$\boxed{\underline{j=1}}$

$$\frac{\beta_k(2)(1-\varepsilon)\mathbb{P}(\overset{1-q}{\cancel{s_{k+1}=1|}}\, s_k=1) + \beta_k(2)(\varepsilon)\mathbb{P}(\overset{q}{\cancel{s_{k+1}=1|}}\, s_k=2)}{\beta_k(1)(1-\varepsilon)+\beta_k(2)(\varepsilon)}$$

$$\beta_{k+1}(1) = \frac{\beta_k(2)(1-\varepsilon)\overset{1-q}{(1-q)} + \beta_k(2)(\varepsilon)\overset{q}{(q)}}{\beta_k(1)(1-\varepsilon)+\beta_k(2)(\varepsilon)}$$

$\boxed{j=2}$

$$\frac{\beta_k(2)(1-\varepsilon)\,\mathbb{P}(\overset{q}{\cancel{s_{k+1}=2}}|\, s_k=1) + \beta_k(2)(\varepsilon)\,\mathbb{P}(\overset{1-q}{\cancel{s_{k+1}=2}}|\, s_k=2)}{\beta_k(1)(1-\varepsilon)+\beta_k(2)(\varepsilon)}$$

$$\beta_{k+1}(2) = \frac{\beta_k(2)(1-\varepsilon)(q) + \beta_k(2)(\varepsilon)(1-q)}{\beta_k(1)(1-\varepsilon)+\beta_k(2)(\varepsilon)}$$

$$\beta_{k+1} = B(\beta_k, 0, 1)$$

<u>repeat</u> for $u=0$, $y=2$, set $\beta_{k+1} = B(\beta_k, 0, 2)$

<u>Case</u> $\boxed{y=2}$

$$\beta_{k+1}(j) = \frac{\overbrace{\sum_{i=1}^{2}\beta_k(i)\,\mathbb{P}(Y_k=2|\, s_k=i, U_k=0)\cdot\mathbb{P}(s_{k+1}=j\,|s_k=i)}^{\beta_k(1)(\varepsilon)\mathbb{P}(\,) + \beta_k(2)(1-\varepsilon)\mathbb{P}(\,)}}{\underbrace{\sum_{i=1}^{2}\beta_k(i)\,\mathbb{P}(Y_k=2|\, s_k=i, U_k=0)}_{\beta_k(1)(\varepsilon)+\beta_k(2)(1-\varepsilon)}}$$

✱ <u>look at i, & do sums</u>

$i=1$, $Y_k=2|\, s_k=1, u_k=0 \quad = \varepsilon$

$i=2$ $\quad Y_k=2|\, s_k=2, u_k=0 \quad = 1-\varepsilon$

$$= \frac{\beta_k(1)(\varepsilon)\,\mathbb{P}(s_{k+1}=j \mid s_k=1) + \beta_k(2)(1-\varepsilon)\,\mathbb{P}(s_{k+1}=j \mid s_k=2)}{\beta_k(1)(\varepsilon) + \beta_k(2)(1-\varepsilon)}$$

* now look at j

$\boxed{j=1}$

$$\frac{\beta_k(1)(\varepsilon)\,\mathbb{P}(\overset{1-q}{\cancel{s_{k+1}=1 \mid}}\, s_k=1) + \beta_k(2)(1-\varepsilon)\,\mathbb{P}(\overset{q}{\cancel{s_{k+1}=1 \mid}}\, s_k=2)}{\beta_k(1)(\varepsilon) + \beta_k(2)(1-\varepsilon)}$$

$$\beta_{k+1}(1) = \frac{\beta_k(1)(\varepsilon)\overset{1-q}{(1-q)} + \beta_k(2)(1-\varepsilon)\overset{q}{(q)}}{\beta_k(1)(\varepsilon) + \beta_k(2)(1-\varepsilon)}$$

$\boxed{j=2}$

$$\frac{\beta_k(1)(\varepsilon)\,\mathbb{P}(\overset{q}{\cancel{s_{k+1}=2 \mid}}\, s_k=1) + \beta_k(2)(1-\varepsilon)\,\mathbb{P}(\overset{1-q}{\cancel{s_{k+1}=2 \mid}}\, s_k=2)}{\beta_k(1)(\varepsilon) + \beta_k(2)(1-\varepsilon)}$$

$$\beta_{k+1}(2) = \frac{\beta_k(1)(\varepsilon)(q) + \beta_k(2)(1-\varepsilon)(1-q)}{\beta_k(1)(\varepsilon) + \beta_k(2)(1-\varepsilon)}$$

$$\beta_{k+1} = \beta(\beta_k, 0, 2)$$

$u=0,\ y=1$

$$\beta_{k+1}(1) = \frac{\beta_k(1)(1-\varepsilon)(1-q) + \beta_k(2)(\varepsilon)(q)}{\beta_k(1)(1-\varepsilon) + \beta_k(2)(\varepsilon)} \qquad (j=1)$$

$$\beta_{k+1}(2) = \frac{\beta_k(1)(1-\varepsilon)(q) + \beta_k(2)(\varepsilon)(1-q)}{\beta_k(1)(1-\varepsilon) + \beta_k(2)(\varepsilon)} \qquad (j=2)$$

$$\beta_{k+1} = \beta(\beta_k, 0, 1)$$

$1-\varepsilon$

$u=0,\ y=2$

$$\beta_{k+1}(1) = \frac{\beta_k(1)(\varepsilon)(1-q) + \beta_k(2)(1-\varepsilon)(q)}{\beta_k(1)(\varepsilon) + \beta_k(2)(1-\varepsilon)} \qquad (j=1)$$

$$\beta_{k+1}(2) = \frac{\beta_k(1)(\varepsilon)(q) + \beta_k(2)(1-\varepsilon)(1-q)}{\beta_k(1)(\varepsilon) + \beta_k(2)(1-\varepsilon)} \qquad \underbrace{(j=2)}$$

$$\beta_{k+1} = \overbrace{\beta(\beta_k, \overset{u}{0}, \overset{y}{2})}$$

3) Develop a PBVI algorithm to find a set of hyperplanes $\tilde{A}_0$ ~~an approximately optimal policy~~ via PBVI (you will need this set later on to do the one-step lookahead during policy execution). To do so, approximate the infinite horizon problem with a finite horizon problem with $N = 100$ stages and no terminal cost:

$$\max_{policy} \mathbb{E}[\sum_{k=0}^{N-1} B_k].$$

Use the following parameters: $q = \epsilon = 0.1$, $Q = 10$, $B = 1$ and the following belief space

$$\tilde{\mathbb{B}} = \left\{ \left[\frac{\ell - 1}{Q-1}, \frac{Q-\ell}{Q-1} \right]^\top : \ell = 1 : Q \right\}$$

Note: The code for 3-5 calls functions for:

- belief update
- Probability $\begin{cases} \text{observation} \\ \text{joint} \end{cases}$
- reward
- one step look ahead
- state / observation generation

ALL functions included at the end of this report as the main code for each part used them in dume way. Please refer to the end for functions.

Part 3 code logic: get A_0

$$\tilde{\mathbb{B}} = \left\{ \left[\frac{\ell - 1}{Q-1}, \frac{Q-\ell}{Q-1} \right]^\top : \ell = 1 : Q \right\}$$

– gets $\beta^{\ell}(1), \beta^{\ell}(2)$ for $1:Q$

• Initialize $\tilde{A}$ with $[\alpha^{1} \dots \alpha^{Q}]$ $\quad \alpha = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

part I

$$\left[\ \tilde{V}_{N-k}(\beta^{(\ell)})\ ,\ u^{*}\ \right]$$

$$= \underbrace{\max_{u \in \mathcal{U}} \underbrace{\sum_i \underline{\beta^{(\ell)}(i)} \Big[c_k(i,u) + \underbrace{\sum_y \mathbb{P}(Y_k = y | S_k = i, U_k = u) \underbrace{\tilde{V}_{N-k-1}(B(\beta^{(\ell)}, u, y))}_{(\cdot)}}_{\text{yloop}} \Big]}_{\text{i loop}}}_{\text{u loop}}$$

yloop:

$$\tilde{V}_{N-k-1}(B(\beta^{(\ell)}, u, y)) = \max_{\alpha \in \mathcal{A}_{k+1}} \langle B(\beta^{(\ell)}, u, y), \alpha \rangle$$

where inner prod.

$$\left. \begin{array}{c} \beta^{\ell}(1) A(1,1) + \beta^{\ell}(2) A(2,1) \\ \vdots \\ \beta^{\ell}(1) A(1,Q) + \beta^{\ell}(2) A(2,Q) \end{array} \right\} \text{get max}$$

get $P_{y|\underbrace{i,u}_{\text{fixed}}}$ (obs. prob.)

yloop = Prob · max(V_{N-k-1})

sum(yloop)

iloop:

iloop = $\beta^{\ell}(i)\left[r(i,u) + \text{sum(yloop)}\right]$

uloop:

max(iloop) for all u.

This becomes u^{*}

Part II

new $d_k^{(\ell)} = \Big[\overbrace{c_k(i,u^*) + \sum_{y,j} \underbrace{\mathbb{P}(Y_k=y, s_{k+1}=j \mid s_k=i, u_k=u^*) \cdot d_{k+1}^{(y)}}_{y,j \text{ loop}}}^{i \text{ loop}} \Big]_{i\in S}$

Where

$$d_{k+1}^{y} = \underset{d\in\tilde{A}_{k+1}}{\arg\max} \langle B(\beta^{\ell}, u^*, y), d\rangle \quad \forall y \in \mathcal{Y}$$

gets $A_{k+1}(:, d_{k+1}^{y})$

⇒ $d^0 \ d^1 \ a^2 \quad \left[\vdots \ \vdots \ \vdots\right]$

y,j loop

sum over y & j

get $P_{y,j|i,u}$ (joint prob)

i loop:

$r(i,u^*)$ + sum(j,y loop)

gets d_k^{ℓ}

⇒ Fill out $A_k(:,\ell)$ with d_k^{ℓ}

Part III

loop everything above (Parts I, II) over ℓ 1 : Q.

This gets A_k

loop everything above (parts I, II, III) over k N-1 : 0

⟹ this gets A_0

Main Code to Solve P3, get A_0

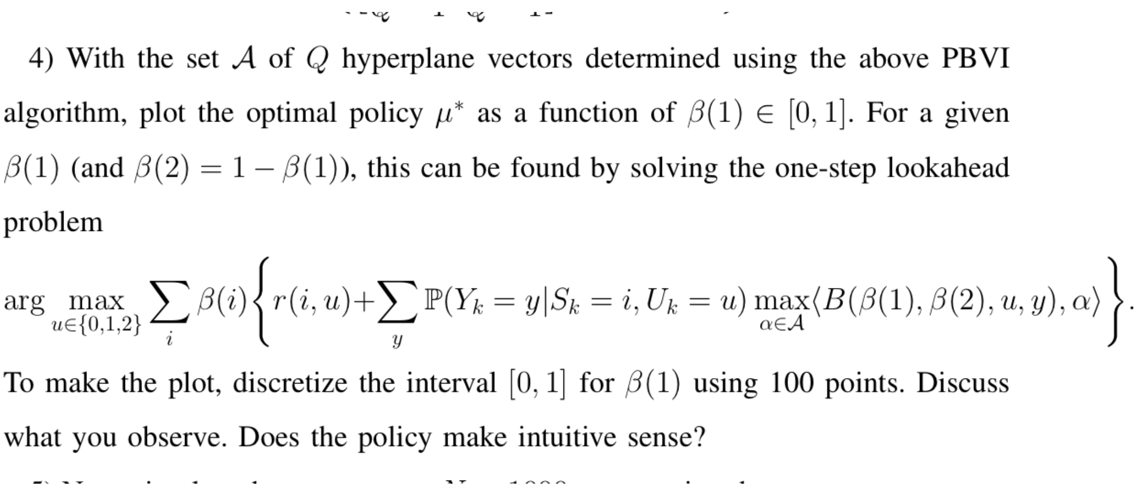
```
%Constants
q = .1; eps = .1;
Q = 10; bits = 1;%bits = B, renamed to avoid confusion

%Initialize AN
A = zeros(2,Q);  %2 states, so 2 rows, Q (1...Q) alphas
%Only need A0 so no need to save previous

%Create IB for l = 1...Q (so a 2,Q size matrix)
IB = zeros(2,Q);
for l = 1:Q
    IB(:,l) = [ (l - 1)/(Q - 1); (Q - l)/(Q - 1)];

end

%Belief update ex
%[bnew] = beliefupd([1;1],2,0);


%Loop k
for k = 99:-1:0   %N-1 to 0, k is never used so this could be 0to99
    %%Loop l        %for l = 1:Q
     (1)
    for l = 1:Q
        %Pick initial belief from IB
        %IB is (beta(i),l)
        beta = IB(:,l);   %Vector with bet(1),bet(2)
        %rest executes after u loop

        %%Loop u
        for u = 0:2
            %executes after i loop

            %%Loop i
            for i = 1:2
                %Executes after y loop

                %%Loop y
                for y = 0:2
                %%%%%%Get V(N-K-1)

                    %%Belief update:
                    betanew = beliefupd(beta,u,y);
                    %%Do inner product (sum IB(i)A(i))
                    %%Looks like IBnl(1,fix)*A(1,1) +
 IBnl(2,fix)*A(2,1)
                    %%Too         IBnl(1,fix)*A(1,Q) +
 IBnl(2,fix)*A(2Q1)
                    innerprod = zeros(1,Q);  %initialize to hold
 values
                    %Get inner product across 1:l of A
                    for l2 = 1:Q
```

```
                       innerprod(l2) = betanew(1)*A(1,l2) +
betanew(2)*A(2,l2);
                   end
                   %Take max
                   Vnk1(y+1) = max(innerprod);
                   %%Gets Max of V(N-K-1) for speific u,y
               (a)

                   %%Yloop value(y) = (Py|i,u)*V(N-K-1)
                   %Get obs prob
                   Probo = obsprob(y,i,u);
                   yloop(y+1) = Vnk1(y+1)*Probo;
                   %Gets Yloop for y=0,1,2

               %%END Y loop
               end
               %i loop execution
               % iloop(i) =  beta(l)(i) * [ck(i,u) + sum(Yloop)]
               %get reward
               riu = reward(i,u);
               iloop(i) = beta(i)*(riu + sum(yloop));


               %%END i loop
           end
           %u loop execution
           uloop(u+1) = sum(iloop);

           %%END u loop
       end
       %get opt action
       [Vtemp(l), ustar(l)] = max(uloop);        %
    (b)
       ustar(l) = ustar(l) - 1; %take care of 1 index
       %%Second y loop in l loop

       for y2 = 0:2
       %argmax for alpha in A(K+1) <IB,alpha> (inner product)
           %Looks like IBnl(1,fix)*A(1,1) + IBnl(2,fix)*A(2,1)
           %Too        IBnl(1,fix)*A(1,Q) + IBnl(2,fix)*A(2Q1)
           %take argmax [val alpha(y)(k+1)] = max(inner prod)
           %store value in lloop(y)
           %belief update
           betanew2 = beliefupd(beta,ustar(l),y2);
           innerprod = zeros(1,Q);  %initialize to hold values
           %Get inner product across 1:l of A
           for l2 = 1:Q
               innerprod(l2) = betanew2(1)*A(1,l2) +
betanew2(2)*A(2,l2);
           end
           %Take max
           [Vtemp2(y2+1),ly(y2+1)] = max(innerprod);
       %END second y loop
```

```
        end

        %%Second i loop
        for i2 = 1:2
            %%third y loop
            for y3 = 0:2

            %%j loop
                for j = 1:2

                %jloop(j) = Py,j|i,ustar)*lloop(y)

                Probjoint = jointprob(y3,i2,ustar(l),j);

                jloop(j) = Probjoint*A(j,ly(y3+1));

                %%END j loop
                end
            yloop3(y3+1) = sum(jloop);
            %END third y loop
            end
            riustar = reward(i2,ustar(l));
            alpha(i2,l) = riustar + sum(yloop3);
    %(c)
            %%END second i loop
        end
        %have alpha l k

        Anew(1,l) = alpha(1,l);
        Anew(2,l) = alpha(2,l);
        %Repeat for  all l
        %Gets Ak(l)
       (2)

        %%END l loop
    end
    %Next K, get Ak-1 set
    A = Anew;
    %END K loop

end
%Have A0
A0 = A


A0 =

  Columns 1 through 7

   56.4041   56.4041   57.7354   57.7354   59.3591   59.6541   60.6874
   60.8672   60.8672   60.6874   60.6874   59.6541   59.3591   57.7354

  Columns 8 through 10
```

```
    60.6874   60.8672   60.8672
    57.7354   56.4041   56.4041
```

Published with MATLAB® R2020b

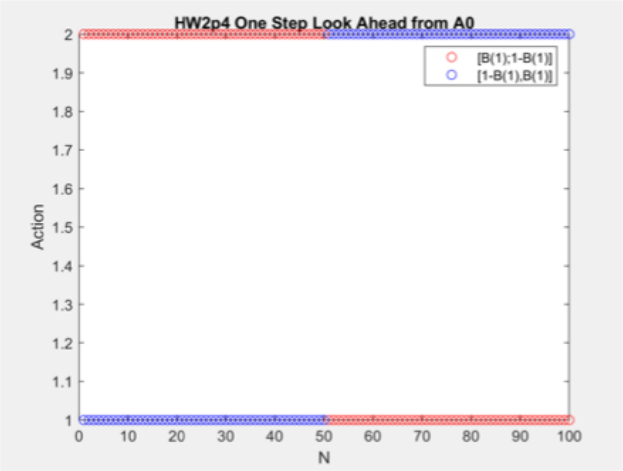


4) With the set $\mathcal{A}$ of Q hyperplane vectors determined using the above PBVI algorithm, plot the optimal policy μ^* as a function of $\beta(1) \in [0,1]$. For a given $\beta(1)$ (and $\beta(2) = 1 - \beta(1)$), this can be found by solving the one-step lookahead problem

$$\arg \max_{u \in \{0,1,2\}} \sum_i \beta(i) \left\{ r(i,u) + \sum_y \mathbb{P}(Y_k = y | S_k = i, U_k = u) \max_{\alpha \in \mathcal{A}} \langle B(\beta(1), \beta(2), u, y), \alpha \rangle \right\}.$$

To make the plot, discretize the interval $[0,1]$ for $\beta(1)$ using 100 points. Discuss what you observe. Does the policy make intuitive sense?

Discussion P4



Here the policy makes sense as Bet(1) and Beta(2) are related as Beta2 =1 - Beta(1). As shown above, you can even see that switching the two values also switches the graph where it flips from 2 to 1 halfway through. As we go from 0 to 1 by 100 points, that would be the point when both betas are around equal. Then the Beta(1) becomes the larger of the numbers.

One step look ahead Code logic (p.4)

~ used in a fcn onesteplookahead.m

problem

$$\left[\arg\max_{u\in\{0,1,2\}} \underbrace{\sum_i \beta(i)\left\{r(i,u)+\overbrace{\sum_y \mathbb{P}(Y_k=y|S_k=i,U_k=u)\underbrace{\max_{\alpha\in\mathcal{A}}\langle B(\beta(1),\beta(2),u,y),\alpha\rangle}_{\ell \text{ loop}}}^{y \text{ loop}}\right\}.}_{i \text{ loop}}\right] \; u \text{ loop}$$

ℓ loop: Belief update and set $\beta(1)_{new}, \beta(2)_{new}$

get inner product for all $\alpha \in A_0$ (1:Q)

$$\left.\begin{array}{c} \beta_n(1)A_0(1,1) + \beta_n(2)A_0(2,2) \\ \vdots \\ \beta_n(1)A_0(1,Q) + \beta_n(2)A_0(2,Q) \end{array}\right\} \text{get max}$$

yloop : sum over y

get $\quad \text{yloop} = \underbrace{P_{y|i,u}}_{\text{Fixed}} \cdot \max\underbrace{(\text{innerproducts})}_{\text{from } \ell \text{ loop}}$

iloop : sum over i

set reward(i,u) Fixed

$\text{iloop} = \beta(i) \cdot [r(i,u) + \text{sum}(\text{yloop})]$

uloop: get max over u values

change u vals and take argmax.

$$\arg\max_{u \in \mathcal{U}} \text{sum}[\text{iloop}]$$

⇒ a new u value can be computed based on different β values

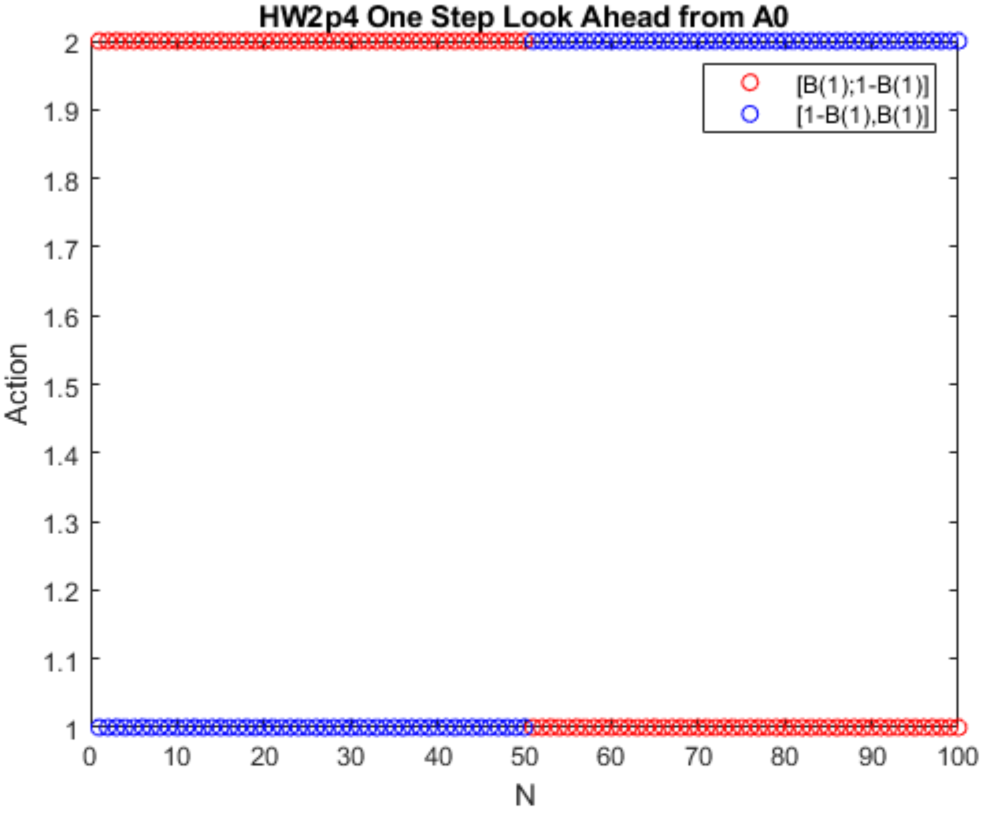
⇒ run above 100 times to get 100 u values for $\beta \in [0,1]$

($\beta(1)$ is 100 even points between 0,1)

($\beta(2) = 1-\beta(1)$)

Main code p4

```
%Import A0 from main_p3
%load('A0.mat')%preloaded to make it easier
A0act = [   56.4041   56.4041   57.7354   57.7354   59.3591   59.6541
 60.6874   60.6874   60.8672   60.8672;
            60.8672   60.8672   60.6874   60.6874   59.6541   59.3591
 57.7354   57.7354   56.4041   56.4041];
%Define beta
beta1 = linspace(0,1,100);
beta = [beta1; 1-beta1];
uvals = zeros(1,100); %initialize
%loop over betas (100 times)
for n = 1:100
%1 step look ahead
%give beta(:,n) and A0
%get and store uval in uvals(n)
    uvals(n) = onesteplookahead(beta(:,n),A0act);
end

%plot
plot(uvals,'ro');
title('HW2p4 One Step Look Ahead from A0');
xlabel('N');
ylabel('Action');
hold on;

%Try with Beta(1) = 1-B(1), B(2) = B(1)
betainv = [1-beta; beta];
for n = 1:100
    uvalsinv(n) = onesteplookahead(betainv(:,n),A0act);
end
plot(uvalsinv,'bo');
legend('[B(1);1-B(1)]','[1-B(1),B(1)]');
```



Published with MATLAB® R2020b

Functions

- included here and at end of report with other Fcns.

One step look ahead

```
function [uval]  = onesteplookahead(beta,A0)
%Constants
q = .1; eps = .1;
Q = 10; bits = 1;%bits = B, renamed to avoid confusion

    %%u loop
    for u = 0:2
        %%i loop
        for i = 1:2
            %%y loop
            for y = 0:2
            %%l loop
            %belief update
            betanew = beliefupd(beta,u,y);
            %get innerproduct
            %Each one looks like

                %Looks like IBnl(1,fix)*A0(1,1) + IBnl(2,fix)*A0(2,1)
                %Too        IBnl(1,fix)*A0(1,Q) + IBnl(2,fix)*A0(2,Q1)
                innerprod = zeros(1,Q);   %initialize to hold values
                %Get inner product across 1:l of A
                for l = 1:Q
                    innerprod(l) = betanew(1)*A0(1,l) +
 betanew(2)*A0(2,l);
                end
            %end l loop

            %get max of inner products

            %get obsprob
            Probo = obsprob(y,i,u);
            yloop(y+1) = Probo*max(innerprod);
            %end y loop
            end
            %sum y loop  (sum(yloop))
            %get reward(i,u)
            iloop(i) = beta(i)*(reward(i,u)*sum(yloop));

            %end i loop
        end
        uloop(u+1) = sum(iloop);

        %%end u loop
    end
    %get max from u loop and return it as uval
    [val, uval] = max(uloop);
    uval = uval - 1;
end
```

---

betanew = beliefupd(beta,u,y);

Published with MATLAB® R2020b

5) Now, simulate the system over $N = 1000$ stages using the same parameters as above, starting with an initial state distribution $P_0(1) = 1/2$ and $P_0(2) = 1/2$ (the initial probability that the car is in sector 1 or 2, respectively). Compute the average amount of bits per stage delivered to the car during your simulation ($\frac{1}{N}\sum_{k=0}^{N-1} B_k$) under the following schemes:

a) The policy where the BS makes a random guess in every slot, and transmits with 50% probability in sector 1 and 50% probability in sector 2

b) The policy where, in even slots (k even), the BS selects the beam training action 0 and collects the feedback Y_k; in odd slots (k odd), the BS transmits on the sector Y_{k-1} identified by the feedback signal of the previous slot.

c) The PBVI policy found earlier. Use only the vectors in the set $\tilde{\mathcal{A}}_0$ to compute the action, i.e., a stationary policy. The initial belief is $\beta_0 = [1/2, 1/2]$.

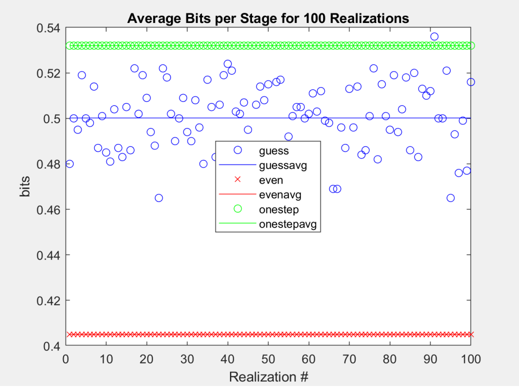
Repeat this process over 100 independent realizations of the state sequences, for all three policies. Plot each realization of $\frac{1}{N}\sum_{k=0}^{N-1} B_k$ for all three policies on the same scatter plot, as well as the average of these 100 realizations.

Comment on what you observe: How many bits do each policy transmit on average? Which one is the best? Explain what you observe and why.

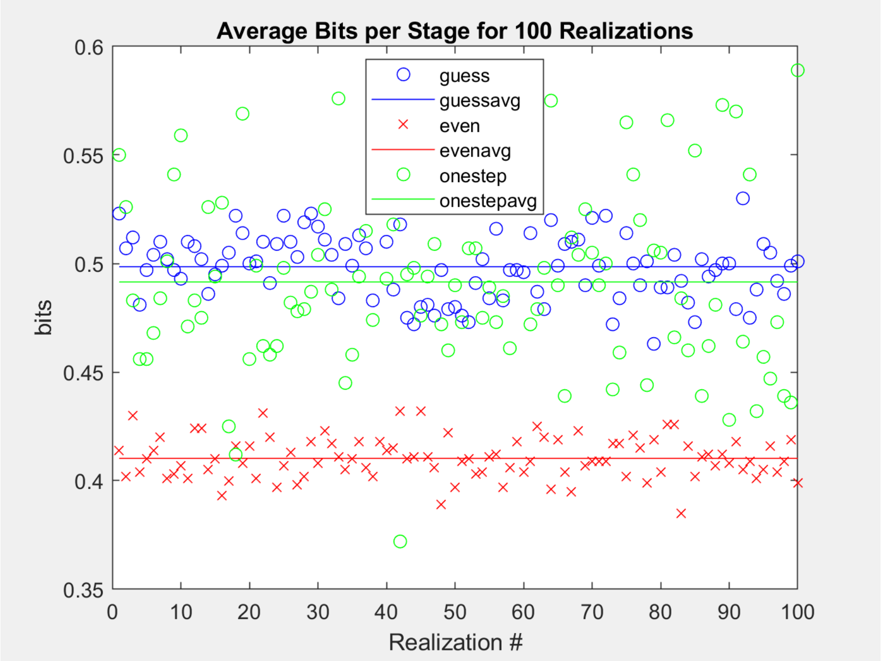
Important: to have a fair comparison of the three policies, you need to generate the sequence of states $\{S_k : k = 0, \ldots, N-1\}$ and of beam-training feedback signals $\{Y_k : k = 0, \ldots, N-1\}$ *beforehand* (for instance, using the MC generator of HW1) and use the *same* sequences to evaluate the performance of all three schemes; schemes b and c will use the feedback signal only when the beam training action is called.

Discussion P5

Originally I had only generated 1 sequence of states and observations before the 100 realizations. This caused an expected graph of the even and one-step look ahead to have the same value each run as there is no randomness involved to change the outcome. Depending on the state and observation sequence, the one-step look ahead sometimes performed better than random and sometimes performed a little worse. The even strategy always performed the west



This however wouldn't make sense as we need some randomness in the system. That will also capture the true average of the one-step look ahead as it varied in performance.



This graph captures the system more appropriately. The random guess seems to always settle around .5 reward which makes sense. This is due to having just two states to choose from making a 50/50 guess usually accurate as it always has a chance to receive a reward with relatively good odds. The even policy uses the observation to make a better guess however by taking an action of 0 to check observation, it guarantees no reward 50% of the time. So at best it could match the guess policy. Now it also has to choose based on observations leaving it worse than a random guess. Lastly is the one-step look ahead which performs around the same as the guess but with a much wider range affecting the average. I believe the results are mainly due to the restrictive state space. If more states are considered it would force the random guess to be wrong more often, and would improve the one-step look ahead. The guess could even be worse than the even policy with more states.

Generate States & Observations for Q5

See function state_obs_gen

$q = \varepsilon = .1$ $\quad P_0(1)=\frac{1}{2}, P_0(2)=\frac{1}{2}$; $P_0 = \begin{pmatrix} .5 \\ .5 \end{pmatrix}$

$$\left.\begin{array}{l} \mathbb{P}(s_{k+1}=1 \mid s_k=1) = 1-q \\ \mathbb{P}(s_{k+1}=2 \mid s_k=1) = q \\ \\ \mathbb{P}(s_{k+1}=2 \mid s_k=2) = 1-q \\ \mathbb{P}(s_{k+1}=1 \mid s_k=2) = q \end{array}\right\} \Rightarrow \mathbf{P}_{i,j} = \begin{array}{c|cc} & 1 & 2 \\ \hline 1 & 1-q & q \\ 2 & q & 1-q \end{array}$$

$$\mathbf{P} = \begin{bmatrix} .9 & .1 \\ .1 & .9 \end{bmatrix}$$

use MC generator from HW1 for 1000.
gets S_k

Get Sequence of observations based on states, assume action = 0.

$$\mathbb{P}(Y_k = 1 \mid s_k=1, U_k=0) = \mathbb{P}(Y_k=2 \mid s_k=2, u_k=0) = 1-q = .9$$

$$\mathbb{P}(Y_k=2 \mid S_k=1, U=0) = \mathbb{P}(Y_k=1 \mid s_k=2, u_k=0) = \varepsilon = .1$$

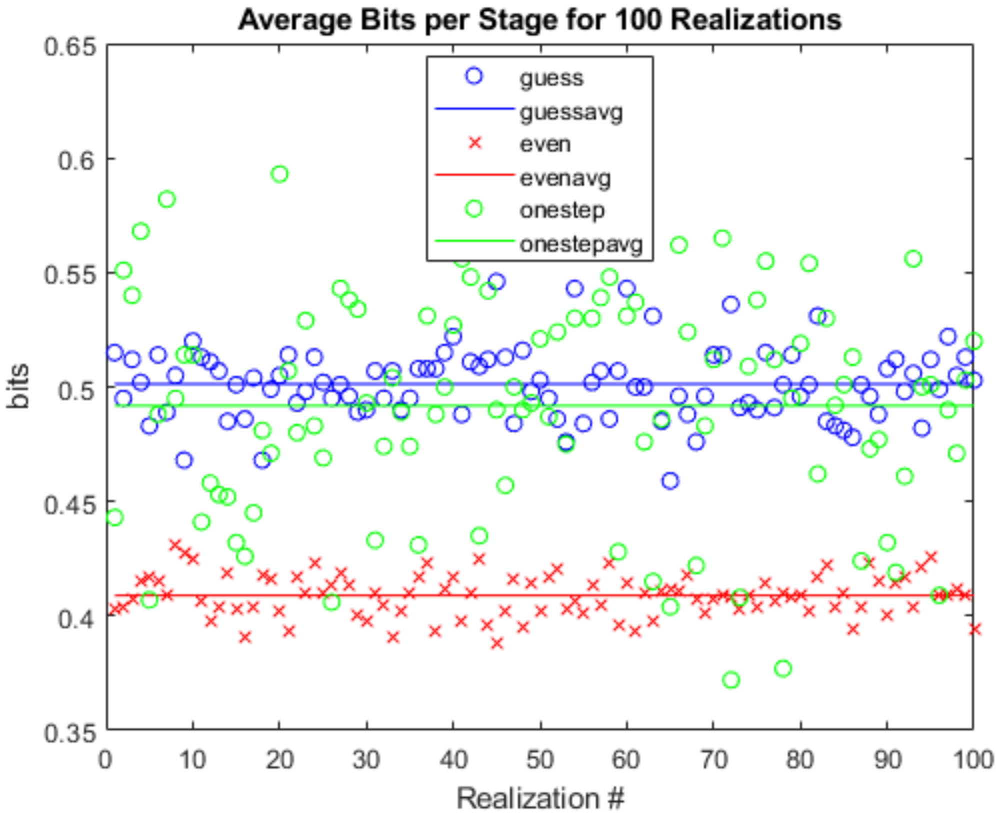
Generate random #
~ x = rand

$\underline{S_k=1}$ $\begin{cases} x < .9, & Y_k=1 \\ x \geq .9, & Y_k=2 \end{cases}$ $\quad \underline{S_k=2}$ $\begin{cases} x < .9, & Y_k=2 \\ x \geq .9, & Y_k=1 \end{cases}$

~rand is normally dist, so ~90% of time, it will select $S_k = Y_k$

Main code p5

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                   import A0
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
load('A0.mat') %loads in as A0
%just in case
A0act = [   56.4041   56.4041   57.7354   57.7354   59.3591   59.6541
 60.6874   60.6874   60.8672   60.8672;
            60.8672   60.8672   60.6874   60.6874   59.6541   59.3591
 57.7354   57.7354   56.4041   56.4041];


N = 1000;

for iter = 1:100
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                   Generate state and obs
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    [S,Y] = state_obs_gen();
    %loop 100 times
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                   Generate 50/50 policy
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for n = 1:N
       val = rand ;
       if rand > .5
           uguess(n) = 1;
       else
           uguess(n) = 2;
       end
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                   generate even odd policy
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %assume n = 1 is 0 (only odd to be zero)
    n=0;
    for n = 0:N-2
        if mod(n,2) == 0  %even time step
            ueven(n+1) = 0; %select action 0
            ueven(n+2) = Y(n+1); %collects Y, new action is Y
        end
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %               generate onesteplook policy
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %initialize beta
    beta = [.5;.5];
    for n = 1:N
        %perform one step look ahead to get u
        uonestep(n) = onesteplookahead(beta,A0);
        %update belief
        beta = beliefupd(beta,uonestep(n),Y(n));
        %get new action

    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %                     Generate rewards
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    for i = 1:N
        guessrewards(i) = reward(S(i),uguess(i));
        evenrewards(i) = reward(S(i),ueven(i));
        onesteprewards(i) = reward(S(i),uonestep(i));
    end
    %get average rewards
    guessbits(iter) = (1/N)*sum(guessrewards);
    evenbits(iter) = (1/N)*sum(evenrewards);
    onestepbits(iter) = (1/N)*sum(onesteprewards);
end
guessavg = mean(guessbits).*ones(1,100);
evenavg = mean(evenbits).*ones(1,100);
onestepavg = mean(onestepbits).*ones(1,100);




%plot


plot(guessbits,'bo');hold on; plot(guessavg,'b'); hold on;
plot(evenbits,'rx');hold on; plot(evenavg,'r');hold on;
plot(onestepbits,'go');hold on; plot(onestepavg,'g');
title('Average Bits per Stage for 100 Realizations');
xlabel('Realization #');
ylabel('bits');
legend('guess','guessavg','even','evenavg','onestep','onestepavg','Location','best
```



Published with MATLAB® R2020b

Functions

Belief update

```
%Inputs:
%beta (2,1)  from IB
%u: action
%y: observation
%Ouput: betanew: updated belief (2,1)
function [betanew] = beliefupd(beta,u,y)
%Constants
q = .1; eps = .1;

if u == 0 && y ~= 0 %case where u = 0 and y = 1,2
    if y == 1
       betanew(1) = (beta(1)*(1 - eps)*(1-q) + beta(2)*eps*q)/...
                    (beta(1)*(1-eps) + beta(2)*eps);
       betanew(2) = (beta(1)*(1 - eps)*(q) + beta(2)*eps*(1-q))/...
                    (beta(1)*(1-eps) + beta(2)*eps);
    else
       betanew(1) = (beta(1)*(eps)*(1-q) + beta(2)*(1-eps)*q)/...
                    (beta(1)*(eps) + beta(2)*(1-eps));
       betanew(2) = (beta(1)*(eps)*(q) + beta(2)*(1-eps)*(1-q))/...
                    (beta(1)*(eps) + beta(2)*(1-eps));
    end

elseif u~=0 && y==0 %case where u = 1,2 and y = 0

    betanew(1) = beta(1)*(1-q) + beta(2)*q;
    betanew(2) = beta(1)*q + beta(2)*(1-q);

else %all other cases u = 1,2 and y = 1,2 or u=0 and y=0
     betanew(1) = 0;
     betanew(2) = 0;
     %Prevents NaN, as we mult by 0 anyways
end
end
```

Published with MATLAB® R2020b

Functions

Joint Probability

```
%Function to get joint probability

function [prob] = jointprob(y,i,u,j)
    q = .1; eps = .1;

    if u == 0
        %Get Obs prob
      if y == 0
          prob = 0;
      elseif y == i
          prob = 1-eps;
      else %y~= i
          prob = eps;
      end
      %multiply by state tranisiton
      if j == i
         prob = (1-q)*prob;

      else
         prob = (q)*prob;
      end



    else %1 and 2 case
      %Get Obs prob
      if y == 0
          prob = 1;
      else
          prob = 0;
      end
        %multiply by state tranisiton
      if j == i
         prob = (1-q)*prob;

      else
         prob = (q)*prob;
      end

end
```

Published with MATLAB® R2020b

Functions

Observation Probability

```
%Function to get observation model probability

function [prob] = obsprob(y,i,u)
    eps = .1;

    if u == 0
      if y == 0
          prob = 0;
      elseif y == i
          prob = 1-eps;
      else %y~= i
          prob = eps;
      end

    else %1 and 2 case
      if y == 0
          prob = 1;
      else
          prob = 0;
      end
    end


end
```

Published with MATLAB® R2020b

Functions

Reward

```
function [reward] = reward(i,u)
    bits = 1;
    if u == 0
      reward = 0;

    else %u= 1,2 case
        if i == u
            reward = bits;
        else
            reward = 0;
        end


end
```

Published with MATLAB® R2020b

Functions

Generate States & Observations

```
function [Sans,Yk] = state_obs_gen()

    % Number of states (n)
    % Transition Prob matrix (P)
    % Initial distribution (Po)
    % Number of stages (K)
    n = 2; P = [.9 .1;.1 .9];
    Po = [.5 .5]; K = 1000;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %              find chain of states
    % Uses gumbel dist and max function to generate
    % chain of states
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    Sans = zeros(K,1)';%initialize state sequence matrix

    for k = 1:K
      vals = zeros(n,1)'; %initialize and zero placeholder for max
 vals
      for i = 1:n   %i goes from 1 to n
        if k == 1 %assume 1 = 0 as matlab has 1 index
        %initial distribution used
            Gi= -log(-log(rand)); %new G value, as G = G(i)
            vals(i) = (Gi + log(Po(i))); %get val of states using
 initial dist
        else
            j = k-1; %index of previous state
            Gi= -log(-log(rand)); %new G value, as G = G(i)
            vals(i) = (Gi + log(P(i,Sans(j)))); %get val of states
        end
      end
        [maxval,Si] = max(vals); %get max values index
        Sans(k) = Si;   %store index as a state
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %              find chain Observations
    %rand is norm dist so 90% should be values < .9
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    Yk = zeros(K,1)';
    errors = zeros(K,1)';
    for i = 1:K
        x = rand;
    if Sans(i) == 1 %State is 1
        if x < .9   %epsilon = .1, so .9 chance of Sk = Yk
            Yk(i) = 1;
        else        %Sk =/= Yk
            errors(i) = 1; %sanity check to make sure correct
 probability
            Yk(i) = 2;
        end
    else            %State is 2
        if x < .9   %epsilon = .1, so .9 chance of Sk = Yk
            Yk(i) = 2;
        else        %Sk =/= Yk
            errors(i) = 1;
            Yk(i) = 1;
        end
    end
    end
end
```

Functions

One step look ahead

```
function [uval]  = onesteplookahead(beta,A0)
%Constants
q = .1; eps = .1;
Q = 10; bits = 1;%bits = B, renamed to avoid confusion

    %%u loop
    for u = 0:2
        %%i loop
        for i = 1:2
            %%y loop
            for y = 0:2
            %%l loop
            %belief update
            betanew = beliefupd(beta,u,y);
            %get innerproduct
            %Each one looks like

                %Looks like IBnl(1,fix)*A0(1,1) + IBnl(2,fix)*A0(2,1)
                %Too        IBnl(1,fix)*A0(1,Q) + IBnl(2,fix)*A0(2,Q1)
                innerprod = zeros(1,Q);  %initialize to hold values
                %Get inner product across 1:l of A
                for l = 1:Q
                    innerprod(l) = betanew(1)*A0(1,l) +
 betanew(2)*A0(2,l);
                end
            %end l loop

            %get max of inner products

            %get obsprob
            Probo = obsprob(y,i,u);
            yloop(y+1) = Probo*max(innerprod);
            %end y loop
            end
            %sum y loop  (sum(yloop))
            %get reward(i,u)
            iloop(i) = beta(i)*(reward(i,u)*sum(yloop));

            %end i loop
        end
        uloop(u+1) = sum(iloop);

        %%end u loop
    end
    %get max from u loop and return it as uval
    [val, uval] = max(uloop);
    uval = uval - 1;
end
```

betanew = beliefupd(beta,u,y);

Published with MATLAB® R2020b