EEE598 Fall 2021

HW3

Jacob Sindorf

Homework 3

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.

10 points total

i = 0:m

I. Inventory control problem with infinite horizon discounted cost

Consider the same inventory management problem of HW1:

A company needs to manage its inventory levels of a certain product in a warehouse. Assume that the control problem operates at discrete stages $k = 0, 1, 2, \ldots$. For instance, each stage may represent a month, or a year (depends on the application).

At stage k, let $S_k \in \{0, \ldots, M\}$ be the current inventory level at the warehouse, and M be the maximum amount of products that can be stocked at the warehouse.

The company refills its inventory only when its inventory level reaches 0.

During stage k, D_k items are purchased from customers (demand). It follows a probability distribution $P(\cdot)$, independent and identically distributed (i.i.d.) over stages, so that $P(d)$ is the probability that $D_k = d$ items are purchased by costumers in stage k.

The cost for the company to purchase each unit is c. The revenue to the company for each unit sold is r. However, due to finite inventory, some of the requests may not be met. In this case, each unit of unsatisfied requests incur a penalty of p. Finally, the cost of maintaining the inventory is m per unit per stage.

Assumptions on the sequence of events:

1 - The stock level is S_k at the beginning of stage k

2 - Then, the maintenance cost is incurred

3 - Then, U_k new units are purchased, if necessary (based on the policy outlined earlier)

4 - Then, D_k units of demand occur. We assume that D_k is uniform in $\{0, \ldots, M\}$, so that $P(D_k = d) = 1/(M + 1),\ \forall d = 0, 1, \ldots, M$.

5 - After the demand is processed, stage k terminates and the new one begins.

Do the following with Matlab, for a scenario with $M = 20$, $c = 1$, $r = 2$, $p = 1$, $m = 0.5$ (same as HW1):

1) **[4 points]** Implement an algorithm to compute the infinite horizon discounted expected profit:

$$V(i) \triangleq \mathbb{E}\left[\left(\sum_{k=0}^{\infty} \gamma^k p_k\right) | S_0 = i\right],$$

where p_k is the profit at the kth stage (function of current state S_k, action U_k, and demand D_k), $\gamma = 0.9$ is the discount factor. Either policy iteration or value iteration are fine, it is up to you.

(used in this report)

Plot the optimal policy as a function of the state, and the optimal value function (two distinct plots).

Code Logic

- policy iteration
  - initialize $\mu^{(0)}$ as lazy policy $\mu(0) = M$, $\mu(i \neq 0) = 0$

- Policy Evaluation

$$V_\mu(i) = \bar{C}(i, \mu(i)) + \gamma \sum_j P_{j|i,\mu(i)} V_\mu(j) \; ; \forall i$$

- $\bar{C}(i, \mu(i)) = \frac{1}{M+1} \sum_{d=0}^{M} \text{cost}(i, \mu(i), d)$ (from HW1)

  Stored in a struct

  Clinevals(i).c($\mu(i)$)

  $i \downarrow \; \xrightarrow{u}$

- Prob. matrix in struct. (from HW1)

  pmat($\mu(i)$).m(i, j)

  → pulls appropriate value given action and state

$$\Rightarrow V_\mu = (I - \gamma \mathbf{P}_\mu)^{-1} \bar{\mathbf{c}}_\mu$$

build $\mathbf{P}_\mu$ based on $i, \mu(i)$ (take row i from $\mu(i)$)

build $\bar{\mathbf{c}}_\mu$ based on $i, \mu(i)$ (take value)

Solve for V_μ

- Policy Improvement

$$\left[V_{\mu n}(i), \mu_n(i)\right] = \max_{u \in U(i)} \bar{c}(i, \mu) + \gamma \sum_j P_{j|i,u} V_\mu(j)$$

- Check

$\mu_{new} = \mu_{old}$?

yes, optimal solution found

No

$\mu_{old} = \mu_{new}$, loop again

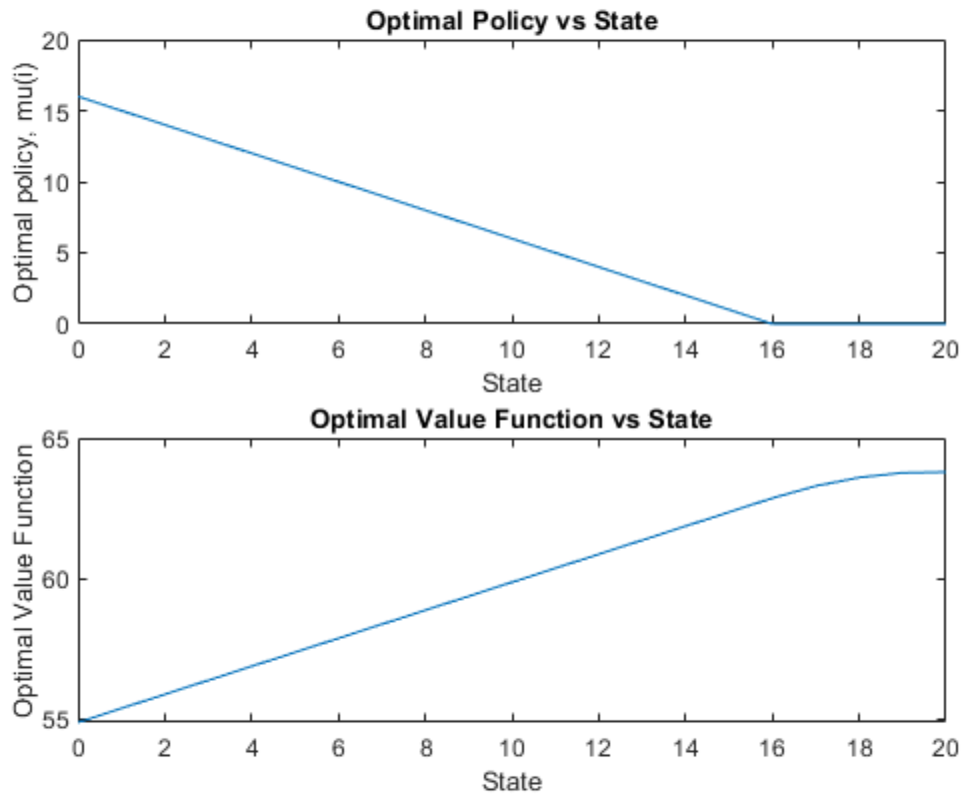
Code for part 1

```
%Constants
M = 20;
c = 1; r = 2; p = 1; m = .5;
gamma = .9;


%Initial Policy = Lazy Policy
mu = zeros(1,M+1);
mu(1) = M;
%Get all possible Pj|i,u and cline values
pmat = pmatcalc(M);
clinevals = clinecalc(M);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Policy Evaluation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Policy Evaluation
%Vmu(i) = cline(i,mu(i)) + gamma*(sumj (Pj|i,mu(i) * Vmu(j))
%Simplify to Vmu = (I - gamma*Pmu)^-1 * Clinemu
iter = 0;      %iteration count
converge = 0; %once optimal is found, break the policy eval/improve
while converge == 0
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Build Pmu and clinemu
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Need to build Pmu/clinemu based on optimal policy.
    %Per i and mu(i), take the entire row corresponding to it from
 pmat
    %take the value for clinevals

    for i = 1:M+1
        Pmu(i,:) = pmat(mu(i)+1).m(i,:);  %to idex the actions, have
 to add one
        %0-20 or 1-21.
        clinemu(i) = clinevals(i).c(mu(i)+1);
    end

    Vmu = inv(eye(M+1) - gamma.*Pmu);
    Vmu = Vmu*clinemu';

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Policy Improvement
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %iloop
    for i = 0:M
        %u loop
        %get max value over all actions
        for u = 0:(M-i) %can't do every action with every state
            %j loop
            %get sum Pj|i,u Vmu(j)
            for j = 1:M+1
```

```
                jloop(j) = pmat(u+1).m(i+1,j)*Vmu(j);
            %end jloop
            end
            uloop(u+1) = clinevals(i+1).c(u+1) + gamma*sum(jloop);
            %end u loop
        end
        [Vmunew(i+1),munew(i+1)] = max(uloop);

        %end i loop
    end
    munew = munew - 1; %Take care of 1 index

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %Policy check
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    if munew == mu
        converge = 1; %break while loop

    else
        %didn't converge, go again
        mu = munew;
        iter = iter + 1;

    end

end
fprintf('optimal policy: \n');
mu

states = [0:1:M];
subplot(2,1,1)
plot(states,mu);
xlabel('State')
ylabel('Optimal policy, mu(i)')
title('Optimal Policy vs State')

subplot(2,1,2)
plot(states,Vmunew);
xlabel('State')
ylabel('Optimal Value Function')
title('Optimal Value Function vs State')
```

```
optimal policy:

mu =

  Columns 1 through 13

    16     15     14     13     12     11     10      9      8      7      6
 5      4

  Columns 14 through 21
```

```
3   2   1   0   0   0   0   0
```



Published with MATLAB® R2020b

Used to build P_μ (used in multiple parts)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate Pmatrix based on action
%store in a struct
%FROM HW1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [pmat] = pmatcalc(M)
delta = .00001;
pmat = struct('m',{}); %structre to hold possible Prob matricies
%matrix based on action, so go from 0 to M for action (1 to M+1 index)

for a = 1:M+1 %(0 to M), go through all possble actions to make M prob
 matricies

    Probmat = zeros(M+1-(a-1),M+1); %initalize based on action
%      %note, i goes from 0 to M, but action cannot exceed next state.
%      %thus limit i based on action as we technically look at i+a to
 get j
%      %so this excludes impossible actions
%      %(ex: M=3, a=1, i~=3 as i + a > M, so exlude row M (3),
%      %giving an (3+1-1,3+1) or (3,4) matrix
%
    for i = 1:size(Probmat,1)
        for j = 1:M+1
            if j > i
               if (j-1) > (i+a-2)%matlab is 1 index so scale
                   Probmat(i,j) = 0;
               else %j <= (i-1) + (a-1)
                   Probmat(i,j) = (1/(M+1));
               end

            else %j<=i
                 if j ~= 1
                    Probmat(i,j) = (1/(M+1));
                else %j == 1
                    Probmat(i,j) = ((M+1-(i-1)-(a-1))/(M+1));
                 end
            end
        end
    end
      if abs(sum(Probmat(i,:)) - 1) > delta %make sure row sums to 1
            i
            fprintf('error'); %if not show where it messed up
      end
    pmat(a).m = Probmat;
end

end
```

used to build $\bar{c}_\mu$ (used in multiple parts)

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate expected cost cline(i,u)
%store in a struct
%FROM HW1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [clinevals] = clinecalc(M)
clinevals = struct('c',{});
%constants
c = 1; r = 2; p = 1; m = .5;

for j=0:M
    clinetemp = zeros(1,(M+1-j));
    for a = 0:M-j
        costpk = zeros(1,a+1);

        for d = 0:M    %find cost based on demand d, state j, and
 action a
            if d > (j+a)
                costpk(1,d+1) = -(j)*m  -c*a + (j+a)*r - p*(d - (j
+a));

            else
                costpk(1,d+1) = -(j)*m - c*a + d*r;
            end
        end
        costpksum = sum(costpk); %sum each row and multiply by prob
        cline = (1/(M+1))*costpksum;
        clinetemp(1,a+1) = cline;
    end
    clinevals(j+1).c = clinetemp;
end

end
```

Published with MATLAB® R2020b

2) **[3 point]** For the optimal policy you have just found, implement a *direct approximation algorithm* to compute an approximate value function. One way to generate features is to use polynomials. So, for instance, $\phi(i) = [1, i, i^2, \ldots, i^{F-1}]^\top$ may be used as the feature vector for the inventory level i, where F is the number of features. For our problem, we are going to use only two features, that is, the following feature vector

$$\phi(i) = [1, i]^\top.$$

Recall that the approximated value function is expressed in state i as

$$\tilde{V}(i) = \phi(i)^\top \cdot \mathbf{r}^*$$

where in direct approximation $\mathbf{r}^*$ is found by minimizing the squared error between the true value function and its approximation, and is obtained as

$$\mathbf{r}^* = (\Phi \cdot \Pi \cdot \Phi^\top)^{-1} \cdot \Phi \cdot \Pi \cdot \underline{\mathbf{V}}$$

where $\mathbf{V}$ is the true value function vector (under the optimal policy); Φ is the feature matrix; Π is a diagonal matrix with diagonal elements $[\Pi]_{i,i} = \pi(i)$ (the steady-state distribution of state i).

The algorithm should return both $\mathbf{r}^*$ and the approximated value function vector $\tilde{V}$.

Code Logic

a) $\phi(i) = [1, i]^T \ \forall i = 0 \ldots M$ compute $\phi(i)$

b) build P_μ + $\bar{C}_\mu$ using $i, \mu(i)$; $V = (I - \gamma P_\mu)^{-1} \bar{C}_\mu$ compute V

c) $[\Pi]_{i,i} = \mathrm{diag}(\pi(i)) \Rightarrow \pi^T(i) = \mathbf{1}^T [w \cdot w^T]^{-1}$ where $w = [I - P_\mu ; \mathbf{1}]$ compute Π

d) $\mathbf{r}^* = (\Phi \cdot \Pi \cdot \Phi^\top)^{-1} \cdot \Phi \cdot \Pi \cdot \mathbf{V}$

e) $\tilde{V}(i) = \phi(i)^\top \cdot \mathbf{r}^*$

Code for part 2

```
%Constants
M = 20;
c = 1; r = 2; p = 1; m = .5;
gamma = .9;
%Get all possible Pj|i,u and cline values
pmat = pmatcalc(M);
clinevals = clinecalc(M);

%Optimal policy from question 1
mustar = [16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0  0  0  0
  0];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Get Phi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
phi = ones(2,M+1);
for i = 0:M
    phi(2,i+1) = i;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Build Pmu and clinemu
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Need to build Pmu/clinemu based on optimal policy.
%Per i and mu(i), take the entire row corresponding to it from pmat
%take the value for clinevals

for i = 1:M+1
    Pmu(i,:) = pmat(mustar(i)+1).m(i,:);  %to idex the actions, have
 to add one
        %0-20 or 1-21.
    clinemu(i) = clinevals(i).c(mustar(i)+1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Get V
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

V = inv(eye(M+1) - gamma.*Pmu)*clinemu';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Get PI
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
onerow = ones(M+1,1);
w = [(eye(M+1) - Pmu) onerow];
pivals = onerow' *inv(w*w');
PI = diag(pivals);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Get rstar
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
rstar = inv(phi*PI*phi')*phi*PI*V;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Get Vtil
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for i = 1:M+1
    Vtil(i) = phi(:,i)'*rstar;
end

fprintf('r and Vtilda values: \n');
rstar
Vtil
```

```
r and Vtilda values:

rstar =

   54.8571
    0.5000


Vtil =

  Columns 1 through 7

   54.8571   55.3571   55.8571   56.3571   56.8571   57.3571   57.8571

  Columns 8 through 14

   58.3571   58.8571   59.3571   59.8571   60.3571   60.8571   61.3571

  Columns 15 through 21

   61.8571   62.3571   62.8571   63.3571   63.8571   64.3571   64.8571
```

Published with MATLAB® R2020b

3) **[3 point]** Note that, to compute $\mathbf{r}^*$, we need to have the steady-state distribution (Π) and the value function ($\mathbf{V}$), which defeats the purpose of approximation methods! To solve this issue, implement a Monte-Carlo method to estimate $\mathbf{r}^*$, without the need to know Π nor $\mathbf{V}$, and without the need to operate with large dimensional vectors. Specifically, implement the online algorithm specified in the class notes (to do so, you will need to simulate the dynamics of the systems, i.e., a sequence of states, controls, demands, and profits, as already done in HW1):

0 – Initialize $S_0 = 0$, $k = 0$; $\mathbf{y}_{-1} = \mathbf{0}$, $\mathbf{B}_0 = \mathbf{0}$, $\mathbf{w}_0 = \mathbf{0}$. Then, for $k \geq 0$ do:
1 – Collect the state sample S_k, generate the action U_k (based on the optimal policy), the demand D_k randomly, and profit p_k (function of S_k, U_k, D_k);
2 – Compute the feature vector $\boldsymbol{\phi}(S_k)$
3 – Update

$$\mathbf{y}_k = \gamma \mathbf{y}_{k-1} + \boldsymbol{\phi}(S_k).$$

$$\mathbf{B}_{k+1} = \left(1 - \frac{1}{k+1}\right) \mathbf{B}_k + \frac{1}{k+1} \boldsymbol{\phi}(S_k) \boldsymbol{\phi}(S_k)^\top$$

$$\mathbf{w}_{k+1} = \left(1 - \frac{1}{k+1}\right) \mathbf{w}_k + \frac{1}{k+1} \mathbf{y}_k C_k$$

Estimate $\mathbf{r}^*$ as

$$\mathbf{r}_{k+1}^{(est)} = \mathbf{B}_{k+1}^{-1} \mathbf{w}_{k+1}$$

4 – Update the state and proceed to the next stage $k+1$

Do this process for $K = 10^5$ timesteps. At each k, compute the error metric

$$\text{error}(k) = \frac{\|\mathbf{r}_{k+1}^{(est)} - \mathbf{r}^*\|_2^2}{\|\mathbf{r}^*\|_2^2}$$

(for a vector $\mathbf{x}$, $\|\mathbf{x}\|_2^2 = \sum_i \mathbf{x}_i^2$, i.e, the sum of the square of all its elements).

Plot $\text{error}(k)$ versus k, and discuss what you observe.

Plot the true value function $V(i)$, the approximated value function $\tilde{V}(i) = \boldsymbol{\phi}(i)^\top \cdot \mathbf{r}^*$, and the one approximated via Monte-Carlo method, $\boldsymbol{\phi}(i)^\top \cdot \mathbf{r}_K^{(est)}$ (at termination of the Monte-Carlo simulation), vs i. Discuss what you observe. In particular, discuss the quality of the approximation in states 17-20. How can you explain what you see? *Hint:* think about the error metric that is used to find $\mathbf{r}^*$:

$$\mathbf{r}^* = \arg\min_{\mathbf{r}} \sum_{i \in \mathcal{S}} \pi(i) (\mathbf{V}(i) - \boldsymbol{\phi}(i)^\top \mathbf{r})^2$$

Code Logic

0 – simulate states, costs, demands (for $k = 10^5$)

with $S_0 = 0$, using μ^*

initialize $y_{-1} = 0$ $B_0 = 0$ $w_0 = 0$

$r_0 = 0$ $(B_0^{-1} w_0 = 0)$

for $k \geq 0$

1 – get S_k, U_k, D_k, C_k from calculated sequence

2 – compute $\phi(S_k)$ $\phi(i) = [1, i]^\top$. (with $i = S_k$)

3 – update

$[- \ -]$ 1×2 $\begin{bmatrix} - \\ - \end{bmatrix}$ 2×1

$$\mathbf{y}_k = \gamma \mathbf{y}_{k-1} + \phi(S_k).$$

$$\mathbf{B}_{k+1} = \left(1 - \frac{1}{k+1}\right) \mathbf{B}_k + \frac{1}{k+1} \phi(S_k) \phi(S_k)^\top$$

$$\mathbf{w}_{k+1} = \left(1 - \frac{1}{k+1}\right) \mathbf{w}_k + \frac{1}{k+1} \mathbf{y}_k C_k$$

4 – get new r_{k+1}^{est} $\mathbf{r}_{k+1}^{(est)} = \mathbf{B}_{k+1}^{-1} \mathbf{w}_{k+1}$

$$\text{error}(k) = \frac{\|\mathbf{r}_{k+1}^{(est)} - \mathbf{r}^*\|_2^2}{\|\mathbf{r}^*\|_2^2}$$ – get error

(for a vector $\mathbf{x}$, $\|\mathbf{x}\|_2^2 = \sum_i \mathbf{x}_i^2$, i.e, the sum of the square of all its elements).

5 – Now $B_k = B_{k+1}$ old = new (update terms)

$W_k = W_{k+1}$

$y_{k-1} = y_k$

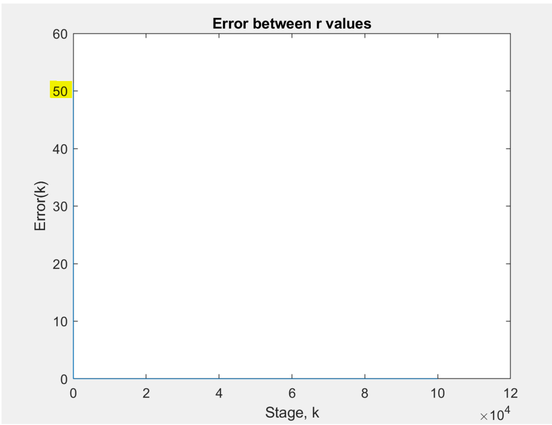
$r_k = r_{k+1}$

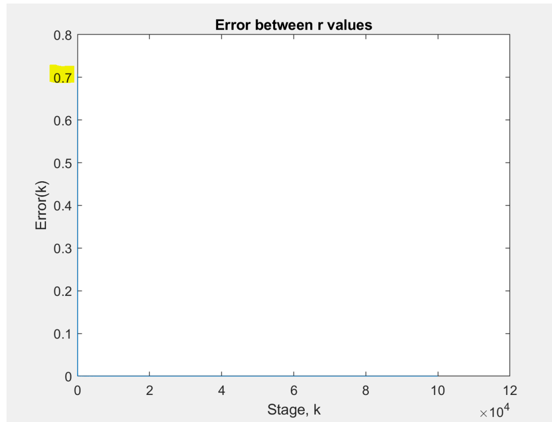
6 – loop $k = 10^5$ times

$$\tilde{V}_{MC}(i) = \phi(i)^\top \cdot \mathbf{r}_K^{(est)}$$

Error Discussion

We can see that the range of error is very high within the first very few timesteps. That being within the first 10-20 timesteps, the error is much much higher than the rest of the runs. Sometimes the error was as high as 50, other times it was around .7 at max. This is due to the randomness in sequence generation.





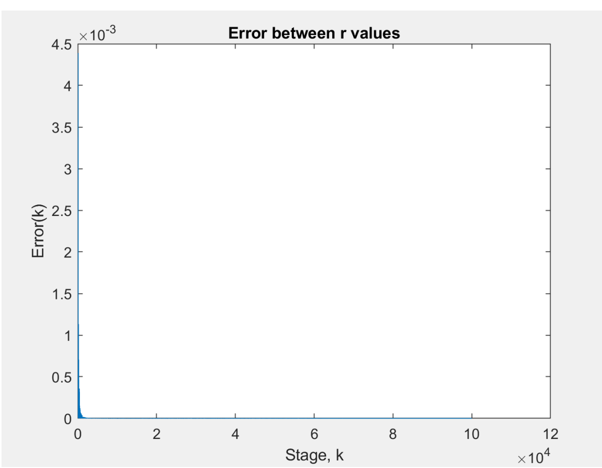
Results show That usually within the first four steps, the error calculation usually has issues. With values of inf and NaN due to singular inverse of Bk. So to fix this I just set the first four to zero.

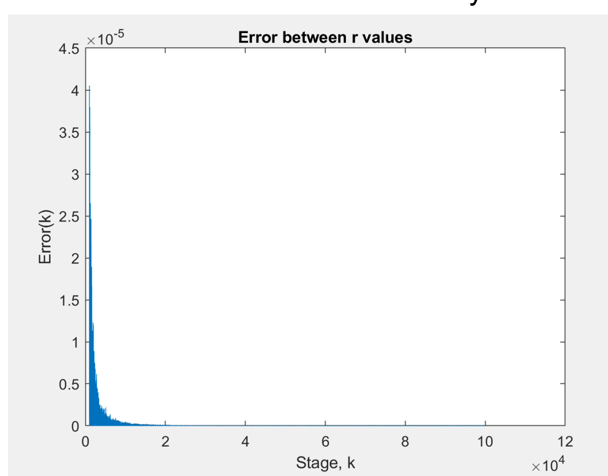
	2	3	4
0	Inf	1	Inf

	2	3	4
0	Inf	NaN	0.0209

	2	3	4
0	0	0	0

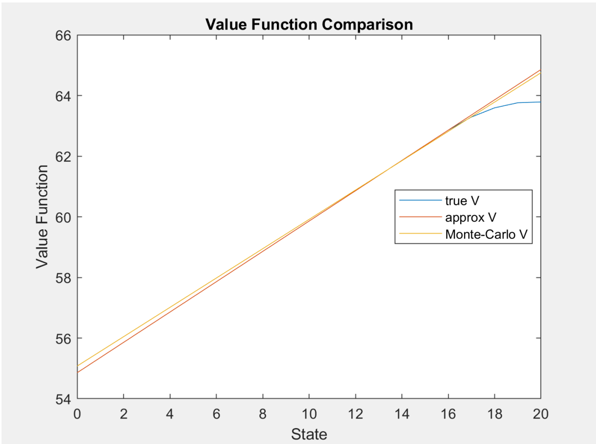
Error is easier to see when we skip the beginning. This graph shows errors after 100 timesteps. It is already to the -3 power in size and still decreases to a very very small number.
The graph next to it is after 1e3 timesteps and still continues to decrease to basically zero.

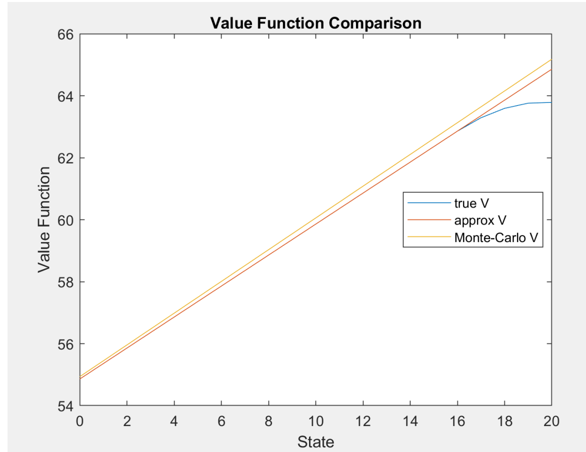




The error has issues in the beginning but quickly converges to around zero.

V Discussion



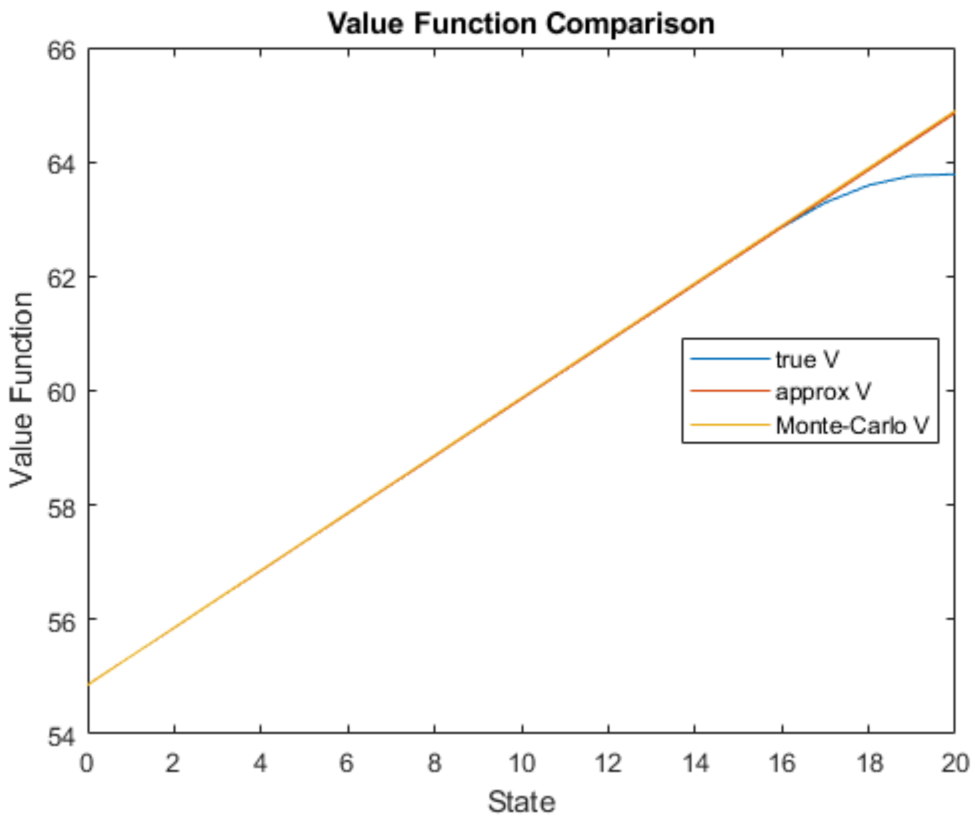


Due to randomness, the Monte Carlo simulation sometimes was about the same as the approximate V, and other times it was either just higher or lower than the approximate. Both however follow the same trend of a linear line from around 55 to 65. Both also diverge from the true V around 17, as this value slopes to a constant. Essentially, we are approximating V with a linear approximation, which is why it keeps a linear trend. It would be difficult for it to follow the quick convergence onto a number as the true value does around 17-20.

Code for part 3

```
%Constants
M = 20;
c = 1; r = 2; p = 1; m = .5;
gamma = .9;

%Stages to simulate
K = 1e5;
%Optimal policy from question 1
mustar = [16 15 14 13 12 11 10  9  8  7  6  5  4  3  2  1  0  0  0  0
  0];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate sequences
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
[dseq, Useq, Sseq, cseq] = generateseq(K,M,mustar);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Initialize
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
yk = [0;0]; Bk = zeros(2); wk = [0;0];
rest = [0;0];

%k loop K>0 (here 1) so start at 2
for k = 2:K+1
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Compute phi
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
phi = [1,Sseq(k)]';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Update
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
yknew = gamma*yk + phi;
Bknew = (1 - (1/(k+1))).*Bk + (1/(k+1)).*(phi*phi');
wknew = (1 - (1/(k+1))).*wk + (1/(k+1)).*yknew*cseq(k);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%r est and error
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
restnew = inv(Bknew)*wknew;

%a lot of issues are happening around k = 2,3
%so we will exclude these in the final plot
%due to randomness in sequence generation it changes each run
%sometimes NaN, sometimes inf. So skip k=1:3
if k > 4
    rdiff = (restnew - rest);
    error(k) = (rdiff'*rdiff)/(rest'*rest);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%old = new
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Bk = Bknew;wk = wknew; rest = restnew;yk = yknew;
%loop
end

%plot error vs k
%This is commented out, uncomment to plot
%Plots and description shown in report.
% Ks = [1:1:K+1];
% plot(Ks(1e3:end),error(1e3:end));
% xlabel('Stage, k')
% ylabel('Error(k)')
% title('Error between r values')

%Compare V vals
phivals = ones(2,M+1);
for i = 0:M
    phivals(2,i+1) = i;
end
for i = 1:M+1
    VMC(i) = phivals(:,i)'*rest;
end

load('Vmunew.mat')%true val fcn from p1
load('Vtil.mat')%approx val fcn from p2
states = [0:1:M];

%NOTE: due to randomness, MC changes each time
%discuss values.....
plot(states,Vmunew,states,Vtil,states,VMC);
xlabel('State')
ylabel('Value Function')
title('Value Function Comparison')
legend('true V','approx V','Monte-Carlo V','Location','best');

Warning: Matrix is singular to working precision.
```



Published with MATLAB® R2020b

Function to generate sequences: demand, state, action, cost

```
%%%%%%Non lazy values%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%FROM HW1
%Generates and returns a sequence of demand, actions(based on opt
 policy)
% sequences, and costs
%Inputs: K (stages), M (max inventory level), Muinf (optimal policy)


function [dseq, Useqstar, Sseqstar, cseqstar] = generateseq(K,M,Muinf)

%Constants
c = 1; r = 2; p = 1; m = .5;
So = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate sequence of demands
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
dseq = zeros(1,K+1);  %initialize vector for demand sequence
for i = 1:K+1
    dseq(i) = randi([0 M]); %random gumbel
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate sequence of states
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generate state, action = Mustar(state), using same sequence of
 demands
%generated earlier
Sseqstar = zeros(1,K+1);  %initialize vector for state sequence
Useqstar = zeros(1,K+1);
%initial state = 0,and demand
Sseqstar(1) = So;
Useqstar(1) = Muinf(Sseqstar(1) + 1);
for i = 2:K+1
    %becomes value less than 0, gets a penatly, new state must be 0
    if dseq(i-1) > (Sseqstar(i-1) + Useqstar(i-1))
        Sseqstar(i) = 0;
        Useqstar(i) = Muinf(Sseqstar(i) + 1);
    else %new state is action + current - demand
        Sseqstar(i) = (Sseqstar(i-1) + Useqstar(i-1)) - dseq(i-1);
        Useqstar(i) = Muinf(Sseqstar(i) + 1);
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Generate sequence of costs
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%must follow maintenance on current state, buy based on action
% sold demand based on current state + action if too much demand
%otherwise just sell based on demand
cseqstar = zeros(1,K+1);  %initialize vector for cost sequence
```

```
for i = 1:K+1
    if dseq(i) > (Sseqstar(i) + Useqstar(i))
        cseqstar(i) = -Sseqstar(i)*m  -c*Useqstar(i) + ...
            (Sseqstar(i)+Useqstar(i))*r - ...
            p*(dseq(i) - (Sseqstar(i)+Useqstar(i)));

    else
        cseqstar(i) = -Sseqstar(i)*m - c*Useqstar(i) +...
            dseq(i)*r;
    end
end


end
```

```
Not enough input arguments.

Error in generateseq (line 17)
dseq = zeros(1,K+1);  %initialize vector for demand sequence
```

Published with MATLAB® R2020b