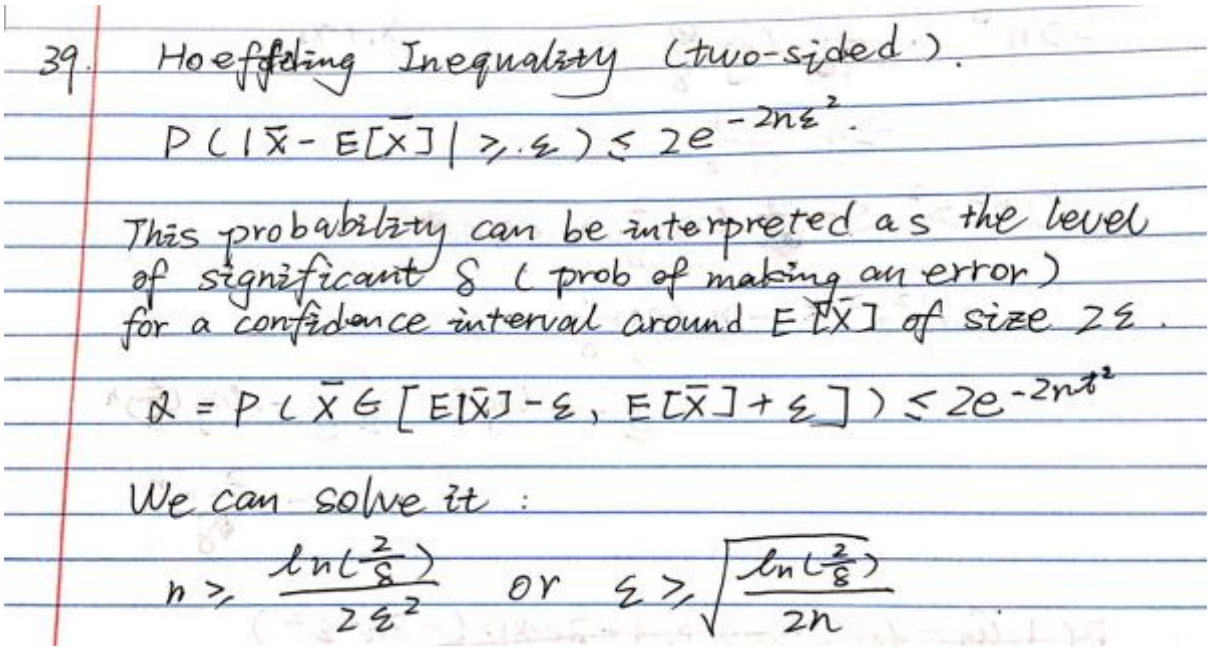
ECE5410 PART 2 HW2

39.



40. Suppose there are 10 iid Gaussian processes, where process i has mean i and variance i2. Illustrate the performance of UCB algorithm. Compare the sample regret with the theoretical prediction of the regret.

A UCB algorithm is implemented in Python. This algorithm is to find the Gaussian process with the largest mean.

Step 0. Initialization.

Construct an array that records outcome of each simulate process, shape of [10,:];

Construct an array that records the means of each simulate process outcome, shape of [10,1], updated in each iteration;

Simulate each process = {1, 2, . . . , 10} once, record the outcomes and update the means.

ÅN=Z=ã~éEÖÉåÉê~í É çëíIê~åÖÉENI NNFF==@=dÉåÉê~í É=ÅNEÍ ÜÉí ~F
Åçëí oÉÅçêÇ=Z=xxÅNxázz=Ñçê=á=áå=ê~åÖÉENMFz
Åçëí oÉÅçêÇ; É~å=Z=ã~éEåéKãÉ~åI Åçëí oÉÅçêÇF=

Step 1. Loop

a. Sampling and evaluation. One process is chosen to be tested based on the candidate solution. After test one process, the outcome is recorded in record array. The means of record are updated.
b. Based on the means, the global optimal is selected.

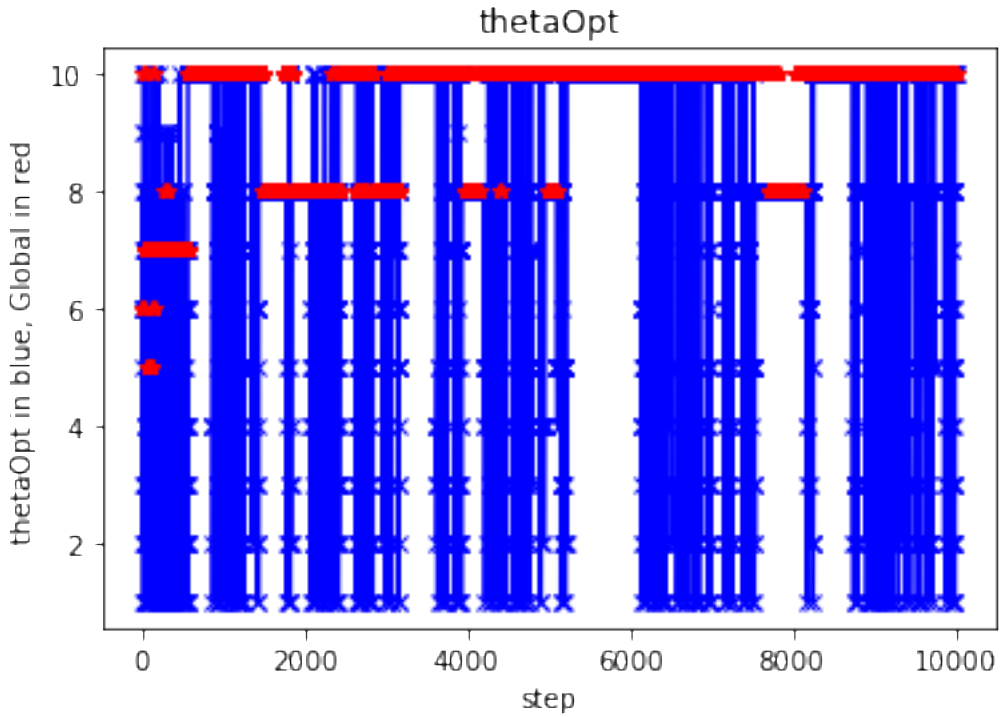
c. Recursions. Beck to a

We can plot the process that each candidate solution picked (blue stars) and the global optimal (red line) at each recursion. 10000 recursions are simulated.

We can see the algorithm has selected all processes, but relatively tended to select processes with large mean, which can be shown in the thetaSelected histogram.

(array([237., 265., 358., 355., 577., 558., 778., 2914.,
2013., 1935.])

The reason is that the algorithm learns the range of where the optimal and sub-optimals may lands in during the simulation. Then candidate solution tries to select the process in this range.



The normalized regret of the simulation is 2.4226, which is close to the theoretical regret of O(log N) = 5.

41. Implement the KL divergence bandit algorithm. Compare this with the UCB algo- rithm in performance.

The performance of two algorithms is simulated with 10 binomial processes.

The KL divergence bandit algorithm can be shown as follow.

Parameters: A non-decreasing function $f : \mathbb{N} \to \mathbb{R}$

Initialization: Pull each arm of $\mathcal{A}$ once

For rounds $t+1$, where $t \geqslant |\mathcal{A}|$,

- compute for each arm $a \in \mathcal{A}$ the quantity

$$B_{a,t}^{+} = \max\left\{q \in [0,1] : \quad N_t(a)\, \mathcal{K}\Big(\beta\big(\widehat{\mu}_{a,N_t(a)}\big),\, \beta(q)\Big) \leqslant f(t)\right\},$$

  where $$\widehat{\mu}_{a,N_t(a)} = \frac{1}{N_t(a)} \sum_{s \leqslant t:\, A_s = a} Y_s\,;$$

- in case of a tie, pick an arm with largest value of $\widehat{\mu}_{a,N_t(a)}$;

- pull any arm $A_{t+1} \in \operatorname*{argmax}_{a \in \mathcal{A}} B_{a,t}^{+}$.

Step 0. Initialization.

Construct an array that records outcome of each simulate process, shape of [10,:];

Construct an array that records the means of each simulate process outcome, shape of [10,1], updated in each iteration;

Simulate each process = {1, 2, . . . , 10} once, record the outcomes and update the means.

ÅN=Z=ã~éEÖÉåÉê~í É çëíIê~åÖÉENI NNFF==@=dÉåÉê~í É=ÅNEï ÜÉï ~F
Åçëí oÉÅçêÇ=Z=xxÅNxázz=Ñçê=á=áå=ê~åÖÉENMFz
Åçëí oÉÅçêÇ} É~å=Z=ã~éEåéKãÉ~åI Åçëí oÉÅçêÇF=

Step 1. Loop

a. Sampling and evaluation. One process is chosen to be tested based on the candidate solution.

   In KL divergence, the best guess with the highest mean of each arm is selected. However, this guess q should be constrained by

$$N_t(a)\, \mathcal{K}\Big(\beta\big(\widehat{\mu}_{a,N_t(a)}\big),\, \beta(q)\Big) \leqslant f(t)$$

   Then pull one process that has the best 'best guess'.

   After test one process, the outcome is recorded in record array. The means of record are updated.

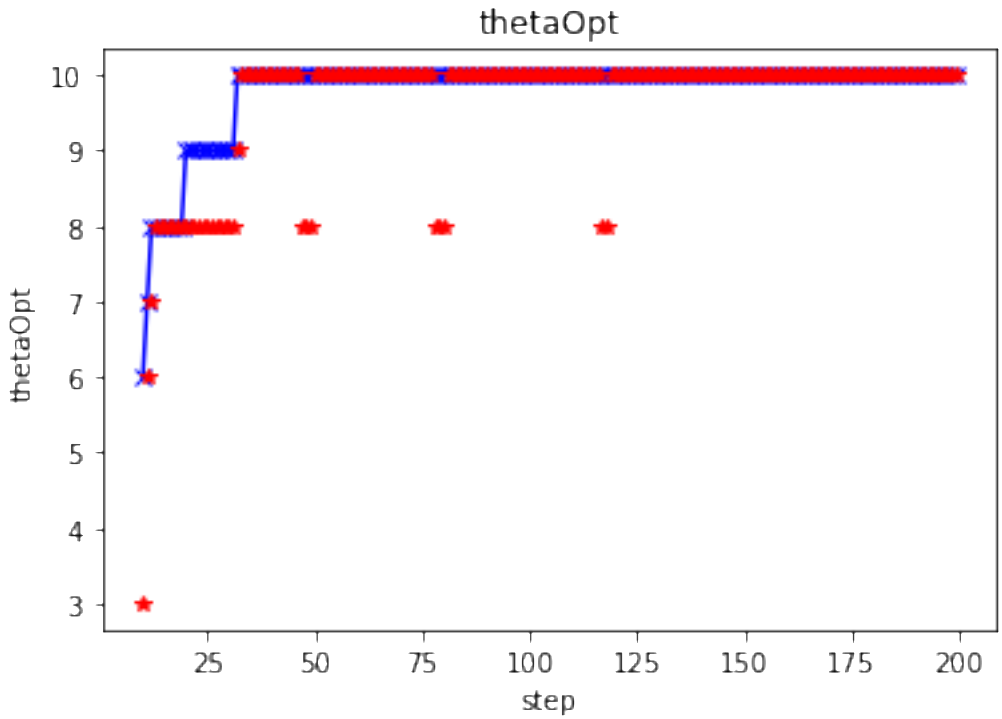
b. Based on the means, the global optimal is selected.

c. Recursions. Back to a

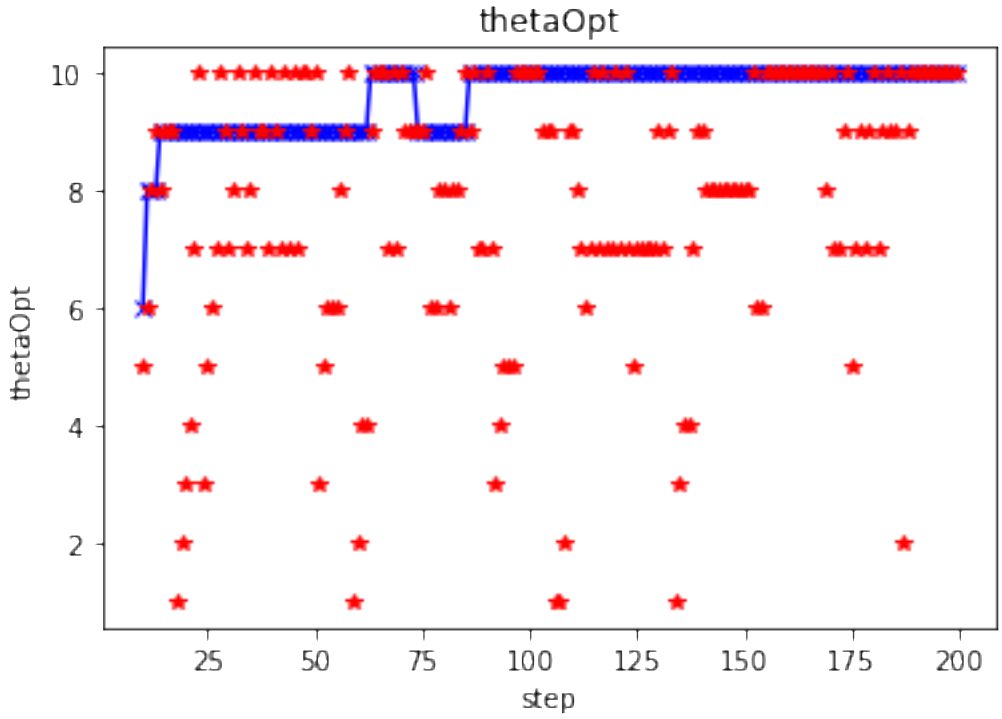
We can plot the process that each candidate solution picked (blue stars) and the global optimal (red line) at each recursion. 200 recursions are simulated.

We can see the algorithm has selected processes with large mean, and the global optimal is quickly converged.

Therefore, KL has a small regret of 0.57.



UCB has selected all processes and the regret is larger. 9.276.



43.

43. rsum = Sum(r) $r = [r_1, \cdots r_n]'$

$w = [w_1, \cdots w_n]'$ covariance matrix R.

Obj: $\min_w w'Rw$ subject to $w'r = rsum$, $w \cdot 1 = 1$, $w > 0_n$.

(a). Convex. R is covariance matrix, so it is symmatrix. and semi-positive.

(b). $w > 0_n \Rightarrow -w_i + z_k^2 = 0 \Rightarrow$

$\therefore$ defining $G(w, z) = -w + z^2 = 0_p$.

Lagrangian

$$L = w'Rw + \lambda_1(w'r - rsum) + \lambda_2(w \cdot 1 - 1) + \sum \mu_i(-w_i + z_i^2)$$

define: $(r, 1) = H$.

$$\nabla_w L = 2Rw + (r, 1)\begin{pmatrix}\lambda_1 \\ \lambda_2\end{pmatrix} = 2Rw + (r, 1)\lambda - \mu$$

$$\nabla_{z_k} L = 2\mu_k z_k \quad \therefore \nabla_z L = 2\mu * z \text{ (dot product)}.$$

$$\nabla_{\lambda_1} L = (w'r - rsum) \quad \nabla_{\lambda_2} L = (w'1 - 1)$$

$$\Rightarrow \nabla_\lambda L = \begin{pmatrix}\nabla_{\lambda_1} L \\ \nabla_{\lambda_2} L\end{pmatrix} = \begin{pmatrix}w'r - rsum \\ w'1 - 1\end{pmatrix}$$

$$\nabla_{\mu_i} L = -w_i + z_i^2 \Rightarrow \nabla_\mu L = -w + z^2$$

$\therefore$ Primal Dual Algorithm:

$$w(k+1) = w(k) - \varepsilon_k(2Rw(k) + H\lambda(k) - \mu)$$

$$z(k+1) = z(k) - \varepsilon_k(2\mu(k) * z(k))$$

$$\lambda(k+1) = \lambda(k) - \varepsilon_k \begin{pmatrix}w(k) \cdot r - rsum \\ w'(k) \cdot 1 - 1\end{pmatrix}$$

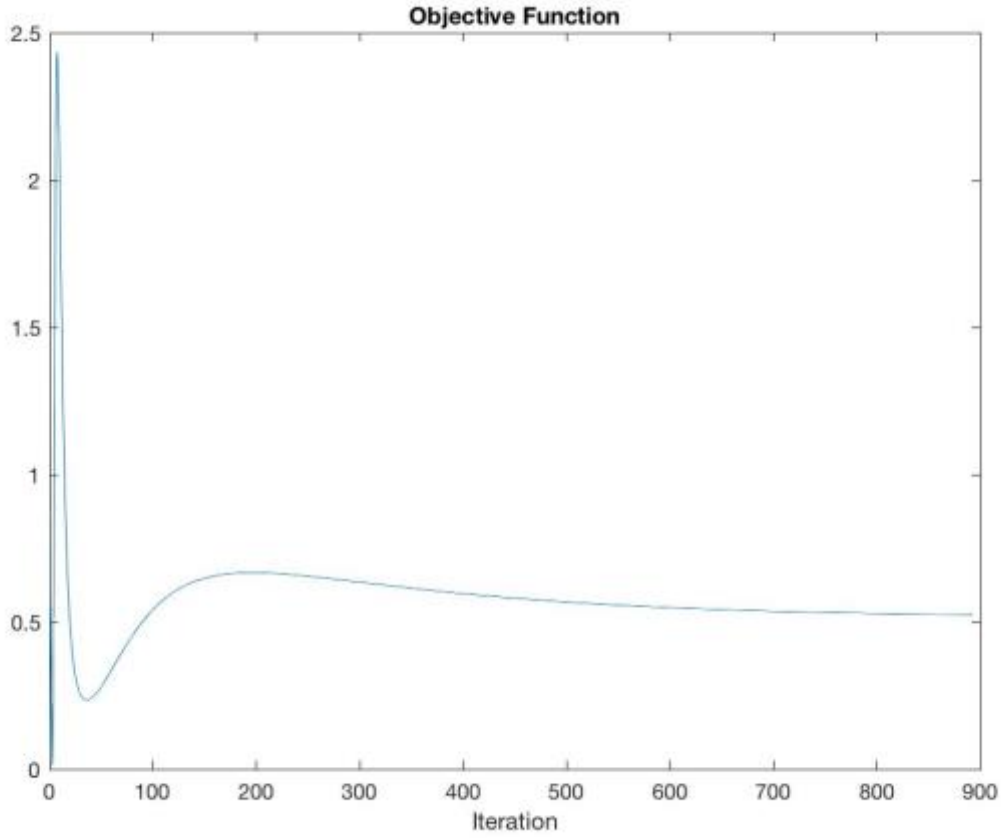
$$\mu(k+1) = \mu(k) - \varepsilon_k(-w(k) + z(k) \cdot z(k))$$

Based on the computation, this algorithm is implemented in MATLAB. R and r are generated by random and scaled to sensible scale. Then if the norm distance of w between two iterations is less than 0.05, algorithm stops and gets the optimal w and z. The distance of w between two iterations can be plotted as below.

```
while dist(k)>0.05
    k = k + 1;
    step = 1/k;
    wnow = w(:,k-1);
    znow = z(:,k-1);
    lamnow = lam(:,k-1);
    munow = mu(:,k-1);

    wnext = wnow - step*(2*R*wnow + H*lamnow - munow);
    znext = znow - step*(2*munow.*znow);
    lamnext = lamnow - step*[wnow'*r-rsum; wnow'*sig-1];
    munext = munow - step*(-wnow+znow.*znow);

    dist = [dist, norm(wnow - wnext)];

    w = [w, wnext];
    z = [z, znext];
    lam = [lam,lamnext];
    mu = [mu, munext];
    if k > 50000 break;
    end
end
```

By setting R = diag(10) we can get the final w and the plot of w'Rw

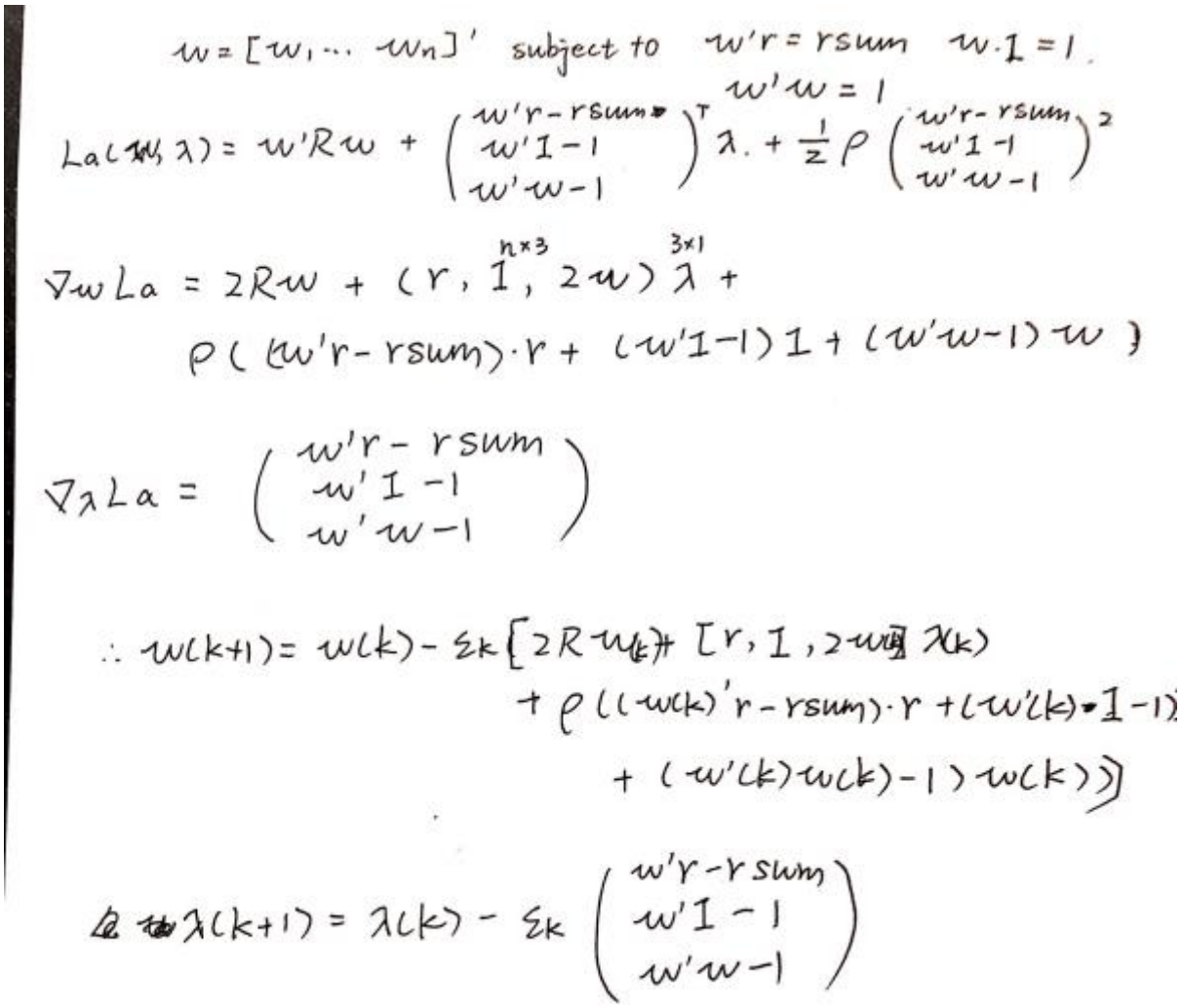
0.0236
0.0364
0.0491
0.0619
0.0747
0.0875
0.1003
0.1131
0.1259
0.1386



(c) There are only two linear constrains:

$$w'r = r, \mathbf{1}'w = 1,$$

(d)
Augmented Lagrangian algorithm in this setting can be expressed as:



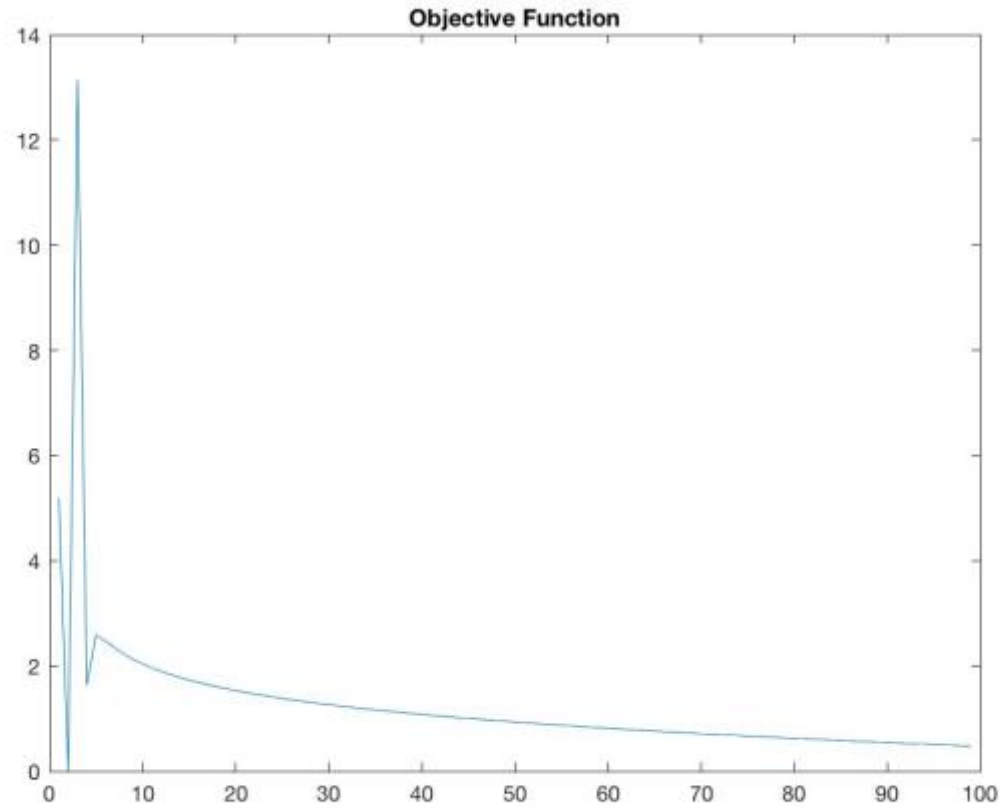

```
while dist > 0.01
    step = 1/2/k;
    wnow = w(:,k-1);
    lamnow = lam(:,k-1);
    H = [r, sig, 2*wnow]
    wnext = wnow - step*(2*R*wnow + H*lamnow...
   +rou*((wnow'*r - rsum)*r + (wnow'*sig - 1)*sig + (wnow'*wnow - 1)*wnow));
    lamnext = lamnow - step*[wnow'*r - rsum; wnow'*sig - 1; wnow'*wnow - 1];
    dist = [dist, norm(wnow - wnext)];
    w = [w, wnext];
    lam = [lam, lamnext];
end
```

If R is randomly generated, the final w in one case would be:

-0.6170
0.1495
-0.3929

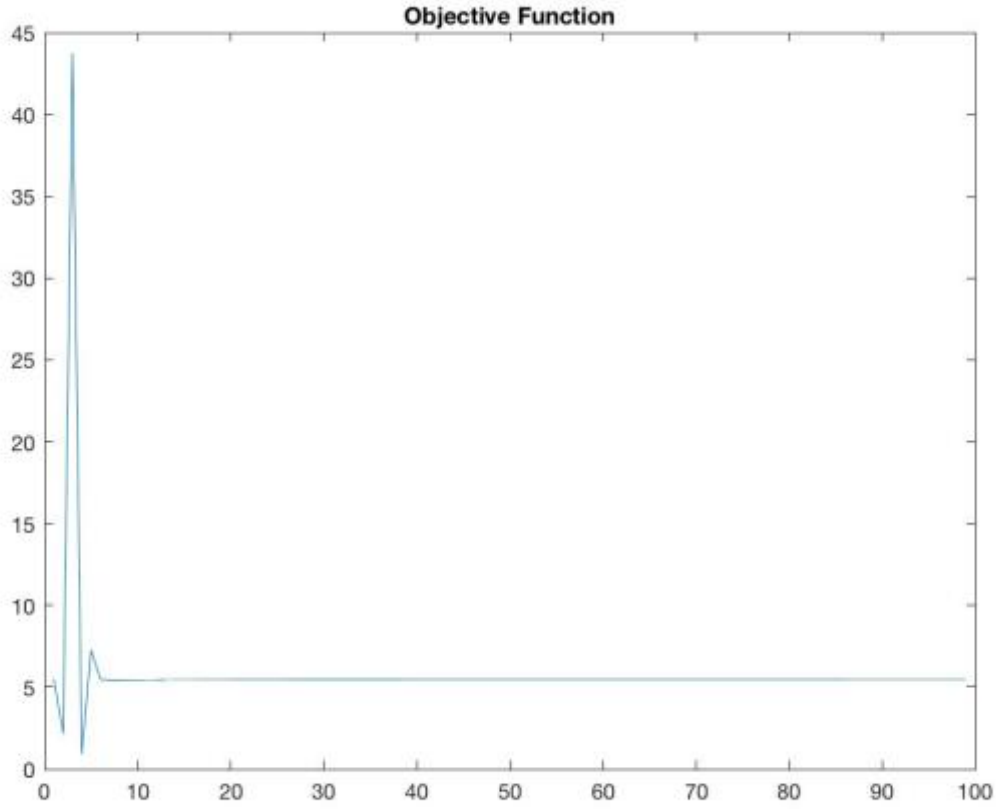
-0.0544
-0.0405
0.4239
0.4116
0.6165
0.3930
0.5734

and the plot of w'Rw

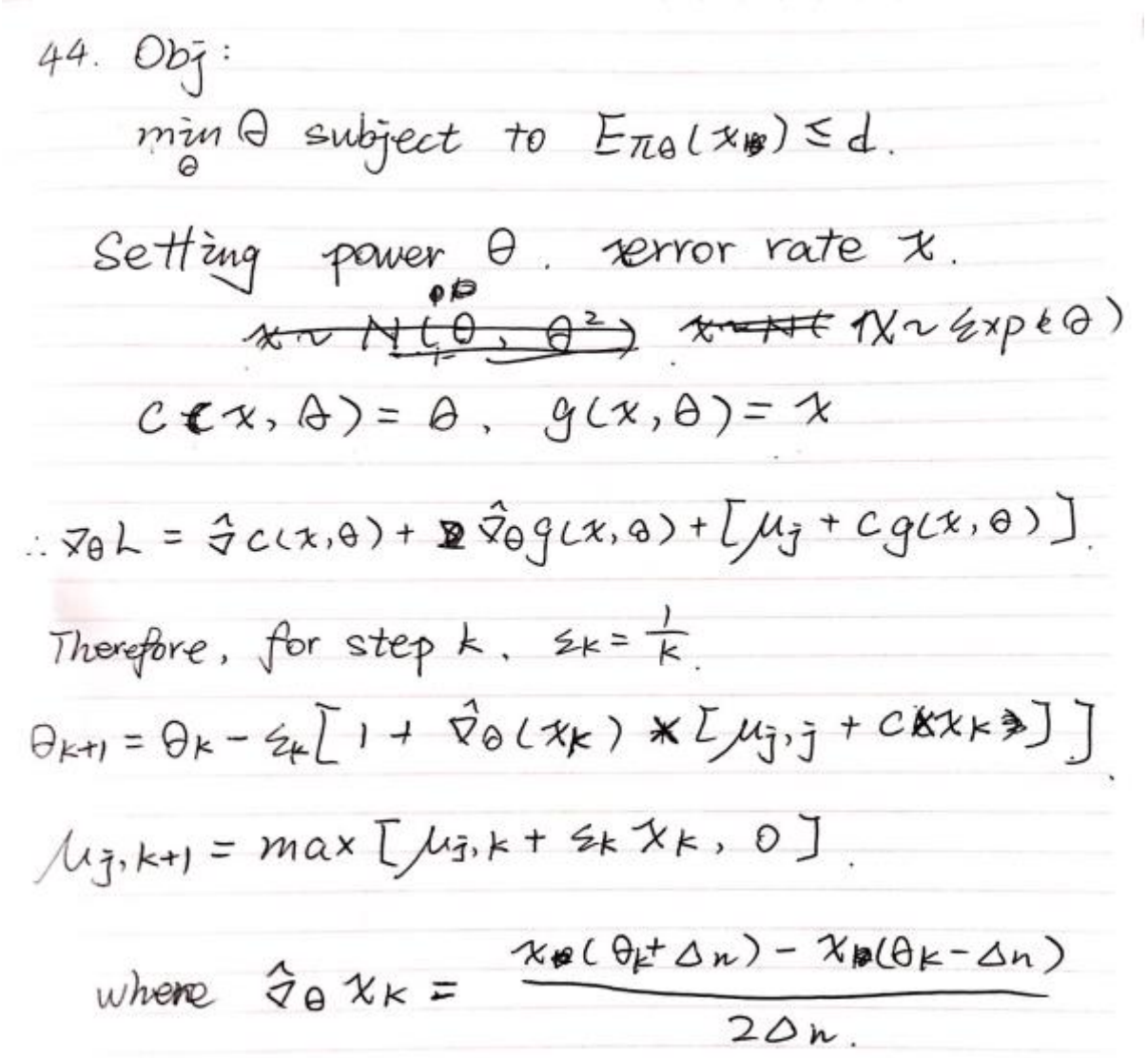


If R id diag(10), we can get an equal w and the plot of w'Rw

0.2997
0.3056
0.3109
0.3142
0.3160
0.3169
0.3174
0.3177
0.3179
0.3181


Objective Function
45
40
35
30
25
20
15
10
5
0
0
10
20
30
40
50
60
70
80
90
100

44.



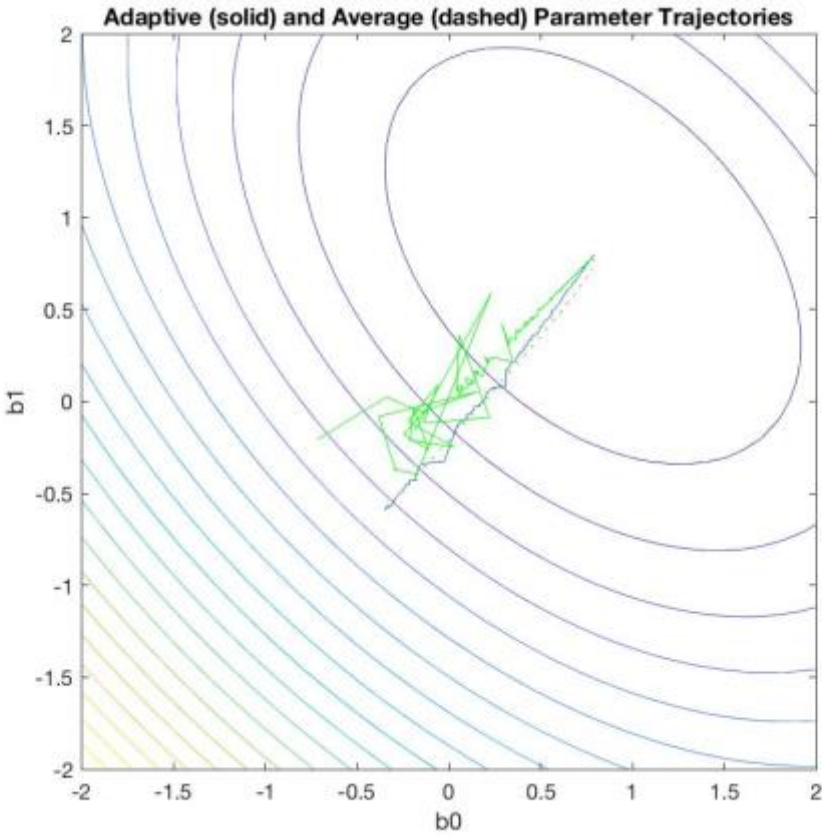
Based on computation, we can implement the augmented Lagrangian algorithm

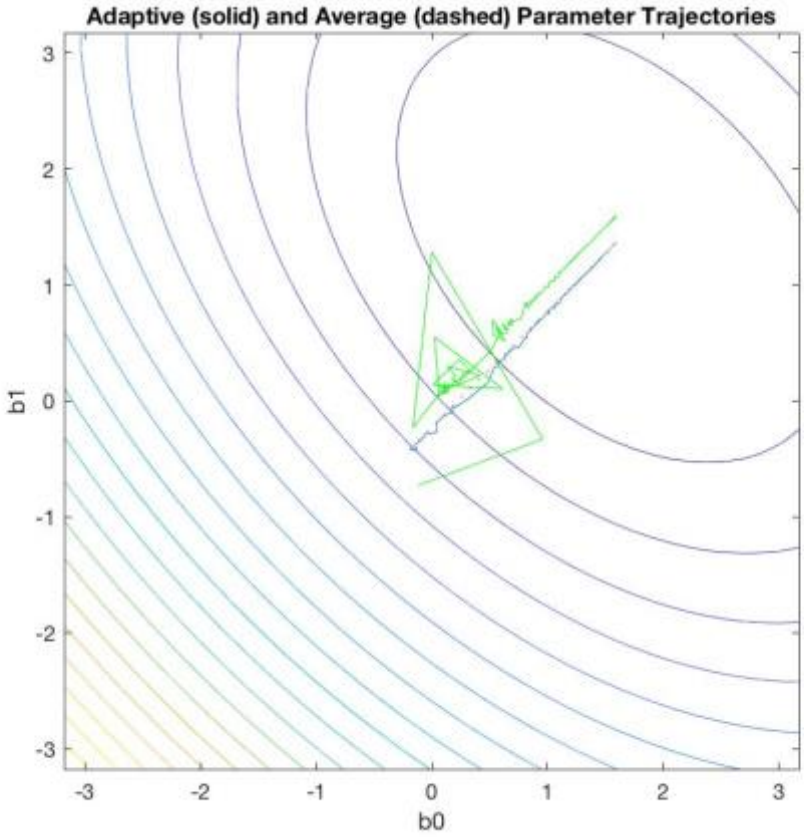
```
for k = 1:300
    th = theta(k);
    munow = mu(k);

    step = 1/10/k;
    xnow = exprnd(th);

    gradStep = 0.1/(1+k)^0.51;
    grad_xk = (exprnd(th + gradStep) - exprnd(th - gradStep))/2/gradStep;
    thetanext = th - step*(1+grad_xk*(munow + c*(xnow-d)));

    munext = max([munow + step * xnow], 0);
```

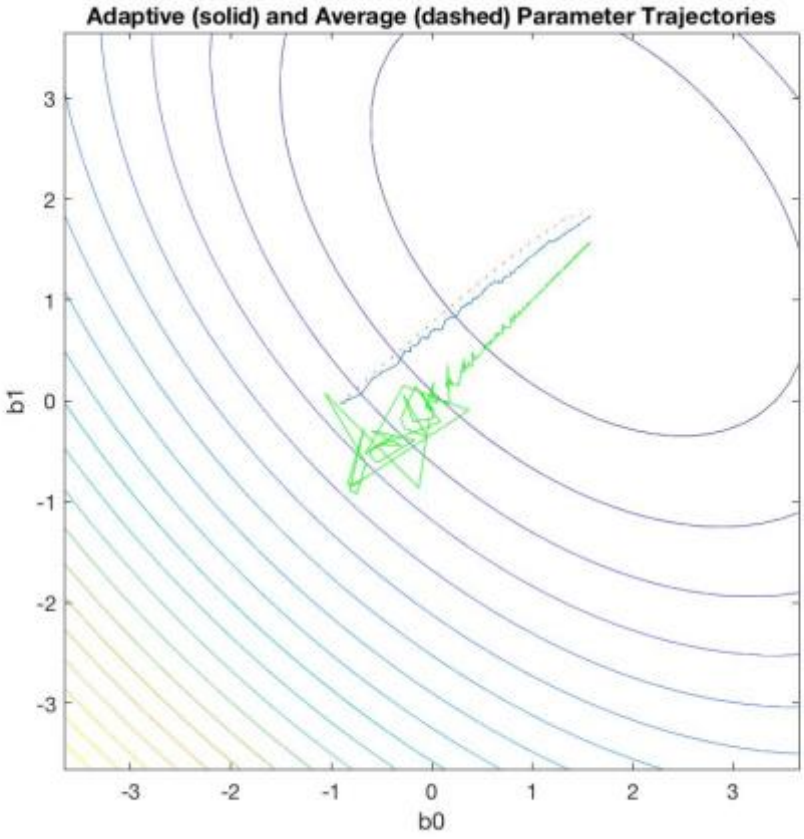
```
    mu = [mu, munext];
    theta = [theta, thetanext];
end
```

45.

Parameters of the system are randomly generated.
The contour of LS, LMS and RLS is plotted to compare two algorithms.
LMS in blue, RLS in green and LS in orange dots.

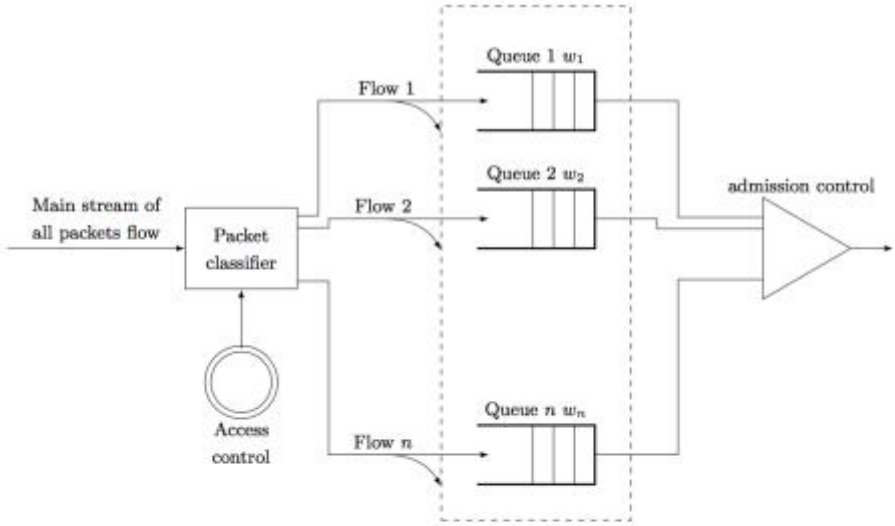



Adaptive (solid) and Average (dashed) Parameter Trajectories
b1
b0
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3


Adaptive (solid) and Average (dashed) Parameter Trajectories
b1
b0
3
2
1
0
-1
-2
-3
-3
-2
-1
0
1
2
3

Due to the randomized parameters, we can cover many circumstances of RLS vs LMS. In some cases, RLS and LMS converge to the some optimal, while RLS may have bias.

48. Problem:

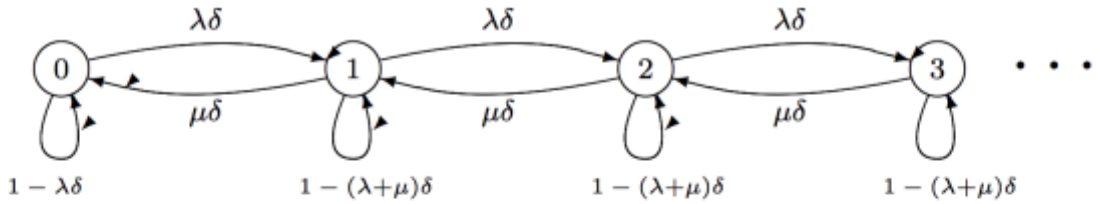


Toy Example. Control of 2 queues with single server.
Assume each queue has max length 4. If a customer arrives to a full queue, she is not served.

Queue i: state $x_k^{(i)}$ is number of customers in queue at time k.
Service time for customer $\mu^{(i)}$, arrival rate $\lambda^{(i)}$, $i = 1, 2$.
Which queue to serve at each time to minimize waiting cost?
Waiting cost at time k is $c^{(1)}x_k^{(1)} + c^{(2)}x_k^{(2)}$
Action u_k: which queue to serve at time k; $u_k \in \{1, 2\}$

Then we can set up the FSM and transition matrices.

To simply the problem, we assume that two queues are independent when computing the cumulative cost in each step of the algorithm.



Transition probability: If $u = 1$, then $P(2) = I$, and

$P(1) =$

$$\begin{bmatrix} 1-\lambda^{(1)}\delta & \lambda^{(1)}\delta & 0 & 0 \\ \mu^{(1)}\delta & (1-(\lambda^{(1)}+\mu^{(1)})\delta) & \lambda^{(1)}\delta & 0 \\ 0 & \mu^{(1)}\delta & (1-(\lambda^{(1)}+\mu^{(1)})\delta) & \lambda^{(1)}\delta \\ 0 & 0 & \mu^{(1)}\delta & 1-\mu^{(1)}\delta \end{bmatrix}$$

and vice versa.

Cumulative Cost: Minimize $\min \mathbf{E}_{\boldsymbol{\mu}}\{\sum_{k=0}^{N-1} c^{(1)}x_k^{(1)} + c^{(2)}x_k^{(2)}\}$

Use Bellman Stochastic Dynamic Programming alg:

Parameters

```
c1 = 0.8; c2 = 1.5; %waiting time weight
mu1 = 0.3; mu2 = 0.25; %Service time for customer Î¼(i), arrival rate (i), i = 1, 2
lambda1 = 0.4; lambda2 = 0.45;
```

State space u:

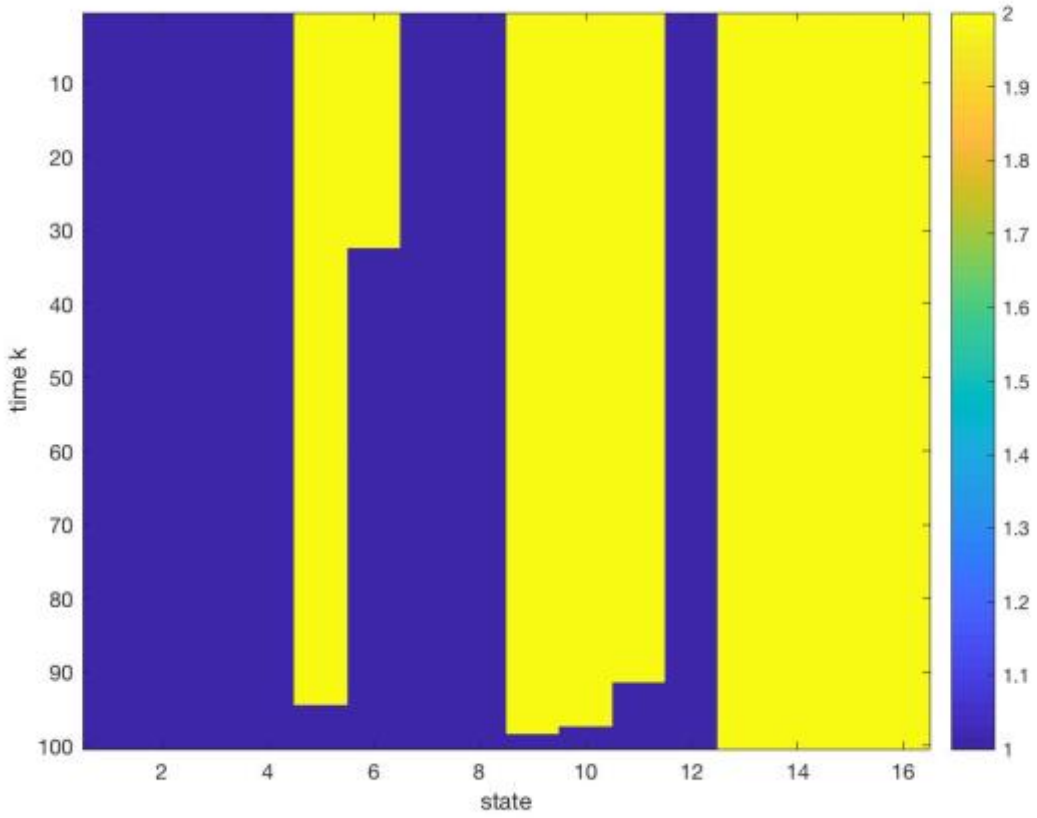
```
u = [0, 0; 1, 0; 2, 0; 3, 0; % State set u(i,:) = [x1, x2]
    0, 1; 1, 1; 2, 1; 3, 1;...
    0, 2; 1, 2; 2, 2; 3, 2;...
    0, 3; 1, 3; 2, 3; 3, 3]; %i = x1 + 1 + x2 * 4
```

Initialization: Set up terminal cost as the remaining guests final waiting time in two queues.

```
J( NN+1, sta) = c1 * x1 * (x1 + 1) / 2 + c2 * x2 *(x2 +1)/ 2;
```

Then backward recursion to compute the cumulative cost in two actions for each state(total 16 state), then find the optimal action for this state at this time mu(i,k)

Final time is set 100, then we can get the action lookup table imagesc.


time k
10
20
30
40
50
60
70
80
90
100
2
4
6
8
10
12
14
16
state
2
1.9
1.8
1.7
1.6
1.5
1.4
1.3
1.2
1.1
1