Stochastic Modeling and Optimization Problemset 3

Aimee Barciauskas, Andreas Lloyd, Francis Thomas Moynihan IV, and Seda Yilmaz

17 March 2016

Problem 1

Show $\Delta J_{k+1}(x_k) \leq \Delta J_k(x_k)$

Given $\Delta J_k(x_k + 1) \leq \Delta J_k(x_k)$

By definition of Δ:

$$J_k(x_k + 1) - J_k(x_k) \leq \Delta J_k(x_k)$$

Using the definition of $J_k(x + i)$ from the problem:

$$\big(r_k(d^*) - d^* \Delta J_{k+1}(x_k + 1) + J_{k+1}(x_k + 1)\big) - \big(r_k(d^*) - d^* \Delta J_{k+1}(x_k) + J_{k+1}(x_k)\big) \leq \Delta J_k(x_k)$$

The $r_k(d^*)$ terms cancel and grouping d^* terms simplifies:

$$d^* \big(\Delta J_{k+1}(x_k) - \Delta J_{k+1}(x_k + 1)\big) + \Delta J_{k+1}(x_k + 1) \leq \Delta J_k(x_k)$$

$$d^* \Delta J_{k+1}(x_k) + (1 - d^*) \Delta J_{k+1}(x_k + 1) \leq \Delta J_k(x_k)$$

Swapping sides of d^*:

$$\Delta J_k(x_k) + (d^* - 1) \Delta J_{k+1}(x_k + 1) \leq d^* \Delta J_k(x_k)$$

$$(d^* - 1) \Delta J_{k+1}(x_k + 1) \leq (d^* - 1) \Delta J_k(x_k)$$

The $d^* - 1$ on both sides term cancels:

$$\Delta J_{k+1}(x_k) \leq \Delta J_k(x_k)$$

Problem 2

Given definitions:

- x_k the state vector,
- u_k the control vector, and
- w_k is the disturbance vector

In order for the terminal state $x_N = f_{N-1}(x_{N-1}, u_{N-1}) + g_{N-1}(w_{N-1})$ to be in the target set X_N, it is necessary and sufficient for $f_{N-1}(x_{N-1}, u_{N-1})$ belong to the effective target set E_N as defined below.

Part a

Given a prescribed target set $X_N \in \mathbb{R}^N$, we recursively solve for x_k in this target set. To do this, we can define the effective target set:

$$E_N = \left\{ z \in \mathbb{R}^n : z + g_{N-1}(w_{N-1}) \in X_N \forall w_{N-1} \in W_{N-1} \right\}$$

which will only be defined given the updated target set T_{N-1}, defined by:

$$T_{N-1} = \left\{ z \in \mathbb{R}^n : f_{N-1}(z, u_{N-1}) \in E_N \text{ for some } u_{N-1} \in U_{N-1} \right\}$$

The DP recursion becomes:

$$E_{k+1} = \left\{ z \in \mathbb{R}^n : z + g_k(w_k) \in T_{k+1} \forall w_k \in W_k \right\}$$

Which is used to update the target set:

$$T_k = \left\{ z \in \mathbb{R}^n : f_k(z, u_k) \in E_{k+1} \right\}$$

$$T_N = X_N$$

What follows is that for X_N to be reachable from set X_k of x_k if and only if $X_k \in T_k$

Part b

X_{k+1} is defined as before but must be contained in X_k for all $w_k \in W_k$

The target tube is defined as:

$$\left\{ \big(X_k, k\big), k = 1, ..., N \right\}$$

If we consider all sets of X_k but X_N to be in $\mathbb{R}^n$, the problem is the same as stated in part 1 with the additional requirement that all $x_k \in X_k$

To initialize the recursion, define the modeified target set:

$$X^*_{N-1} = T_{N-1} \cap X_{N-1}$$

it is necessary and sufficient that

$$x_{N-1} \in X^*_{N-1} \text{ and } T_{N-1}$$

e.g. x_{N-1} is from the modified target set.

The DP recursion is:

$$E^*_{k+1} = \left\{ z \in \mathbb{R}^n : z + g_k(w_k) \in X^*_{k+1} \forall w_k \in W_k \right\}$$

$$T_k^* = \left\{ z \in \mathbb{R}^n : f_k(z, u_k) \in E_{k+1} \text{ for some } u_k \in U_k \right\}$$

$$X_k^* = T_k^* \cap X_k$$

$$X_N^* = X_N$$

The target tube $\left\{ X_j, j; j = k+1, ..., N \right\}$ is reachable at time k if and only if $x_k \in T_k^*$ e.g. the target tube is reachable if and only if $X_0 \subset T_0^*$

Problem 3

The set up of the problem is that we have the DP algorithm of the form

$$J_N(x_N, y_N) = x_N^{'} Q_N x_N J_k(x_k, y_k) = minE\{x_k^{'} Q_k x_k + u_k^{'} R_k u_k + \Sigma p_i^{k+1} J_{k+1}(x_{k+1}, i)|y_k\}$$

Then we need to prove $J_k(x_k, y_k) = x_k^{'} K_k x_k + x_k^{'} b_k(y_k) + c_k(y_k)$ by induction. We start by assuming it is true for $J_{k+1}(x_{k+1}, y_{k+1})$, then for $J_k(x_k, y_k)$

$$J_k(x_k, y_k) = minE\{x_k^{'} Q_k x_k + u_k^{'} R_k u_k + \Sigma p_i^{k+1} [x_{k+1}^{'} K_{k+1} x_{k+1} + x_{k+1}^{'} b_{k+1}(i) + c_{k+1}(i)]|y_k\} = x_k^{'} Q_k x_k + min\{u_k^{'} R_k u_k + E\{(A_k x_k$$

Where we have absorbed the sum terms to pull out the terms with A_k and B_k, and $b_{k+1} = \Sigma p_i^{k+1} b_{k+1}(i)$, $\gamma_{k+1} = \Sigma p_i^{k+1} c_{k+1}(i)$. The expectations are over w_k, and the minimisation over u_k, as normal, so further simplifications can be made to give us a term dependent on u_k and one not. The term dependent on u_k is the one that gives our optimal control law,

$$min\{u_k^{'}(R_k + B_k^{'} K_{k+1} B_k) u_k + 2u_k^{'} B_k^{'} K_{k+1} A_k x_k + 2u_k^{'} B_k^{'} K_{k+1} E[w_k|y_k] + u_k^{'} B_k^{'} b_{k+1}\}$$

The minimum here can be found by differentiation as the first term is positive,

$$2(R_k + B_k^{'} K_{k+1} B_k) u_k^* + 2B_k^{'} K_{k+1}(A_k x_k + E[w_k|y_k] + B_k^{'} b_{k+1}) = 0$$

This gives an optimal control law as

$$u_k^* = -(R_k + B_k^{'} K_{k+1} B_k)^{-1} B_k^{'} K_{k+1}(A_k x_k + E[w_k|y_k]) - \frac{1}{2}(R_k + B_k^{'} K_{k+1} B_k)^{-1} B_k^{'} b_{k+1}$$

And the final term can be referred to as α_k.

Substituting this back in to the equation gives us a set of quadratic, linear and constant terms in x_k, which can be re-formulated in the desired form to give,

$$J_k(x_k, y_k) = x_k^{'} K_k x_k + x_k^{'} b_k(y_k) + c_k(y_k)$$

which is the desired form. So the proof is complete and the desired form of the optimal control was found along the way.

Problem 4

Now the aim is to prove that the optimal control law depends linearly on the state. Considering the cost function, the formulation of the DP algorithm is

$$J_N(X_N) = e^{x_N^2} J_k(x_k) = min\{e^{x_k^2 + ru_k^2} E(J_{k+1}(x_{k+1}))\}$$

and we want to proceed by proving $J_k(x_k) = \alpha_k e^{\beta_k x_k^2}$. Assume it is true for $J_{k+1}(x_{k+1}$ and considering $x_{k+1} = a_k x_k + b_k u_k + w_k$, then for $J_k(x_k)$,

$$J_k(x_k) = min\{e^{x_k^2 + ru_k^2} E(\alpha_{k+1} e^{\beta_{k+1} x_{k+1}^2})\} = min\{e^{x_k^2 + ru_k^2} \frac{\alpha_{k+1}}{\sqrt{Z_{k+1}}} e^{\frac{\beta_{k+1}(a_k x_k + b_k u_k)^2}{Z_{k+1}}}\}$$

where the given relation was used and $Z_{k+1} = 1 - 2\beta_{k+1}\sigma^2$ was used as a dummy variable to make things look a bit neater.

Now, the only non-constant terms wrt u_k are the exponentials, so the minimisation is a simple problem of differentiating the exponential parts to give,

$$2ru_k^* + 2\frac{b_k u_k^*}{Z_{k+1}} = 0$$

This is clearly a linear equation, so we have $u_k^* = \gamma_k x_k$, as desired. Similar to the last problem, subsituting in the form for u_k^* will complete the induction, as there is a x_k^2 term introduced in the exponential. We can then absorb all remaining constants and re-using the relation given by the question, a form of $J_k(x_k) = \alpha_k e^{\beta_k x_k^2}$ is returned simply.

Problem 5

Below we define the function called `get.states` which returns a 2×1 vector of states from time 1 to N.

The plots which follow are given 2 per sub-problem, 1 each of the values of x state vector. In other words, one plot for each x_1 and x_2 over time given the different system arguments. The blue line is always for the "small" or "non-riccardi" version of the system and the red line is always for the "large" or "riccardi" of the system.

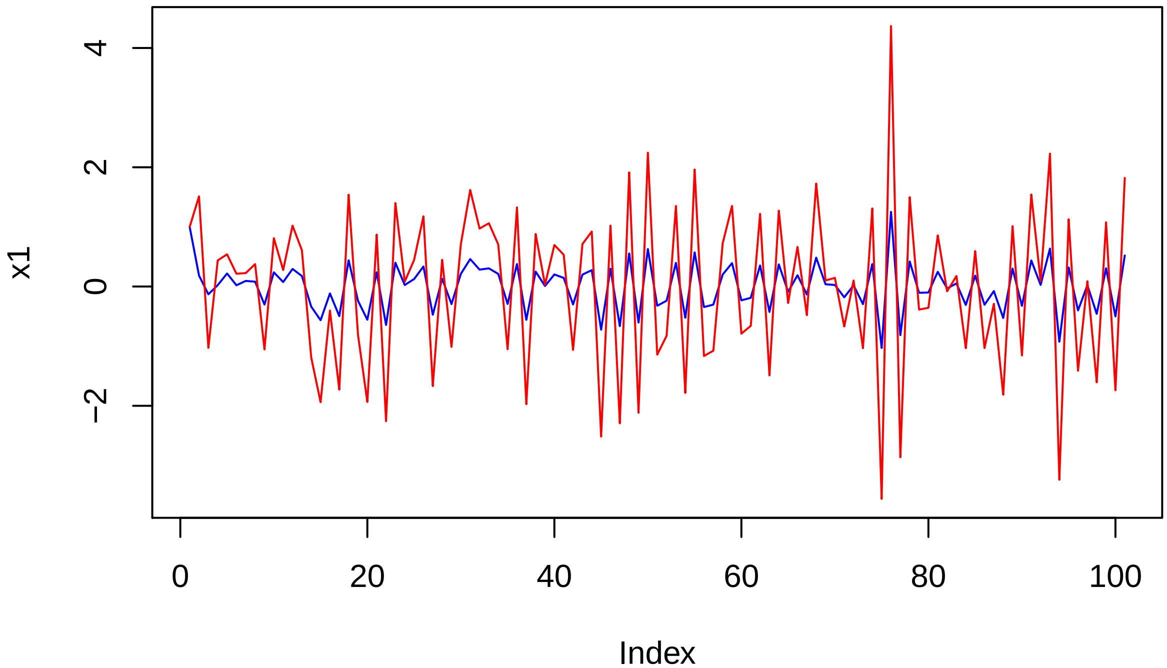
```
if (!require('assertthat')) install.packages('asserthat')
if (!require('matrixcalc')) install.packages('matrixcalc')
if (!require('mvtnorm')) install.packages('mvtnorm')
if (!require('Matrix')) install.packages('Matrix')

# Initial values for x, A, B, C and R
# go to 101 since R's indexing starts at 1
# the first element of everything is associted with t = 0,
# and the last element (the 101th element) with t = 100
get.states <- function(N = 101,
                       x0 = c(1,1),
                       A = matrix(c(0,1,1,0), nrow = 2, ncol = 2),
                       B = matrix(c(1,1,7,1), nrow = 2, ncol = 2),
                       C = c(2,1),
                       Q = C %*% t(C),
                       R = diag(x = c(2,3)),
                       ws = rmvnorm(N-1, mean = c(0,0), sigma = diag(x = c(0.1,0.2))),
                       riccardi = FALSE) {
```

```r
  set.seed(321)
  are_equal(rankMatrix(cbind(A, A%*%B))[1], max(nrow(A), ncol(A)))
  assert_that(is.positive.definite(R))
  # Initialize stores for x, L and K
  x.mat <- matrix(NA, nrow = N, ncol = length(x0))
  # store the initial state x0 as the first element in x
  x.mat[1,] <- x0
  #L.mat <- matrix(NA, nrow = N, ncol = length(x0))
  L.list <- list()
  # K's are (length of C) x (length of C), e.g. NxN
  K.list <- list()

  # Terminal K_N = C'C
  K.list[[N]] <- Q

  # for t = 99 to 0, solve for K and L
  # R indices 100 to 1
  for (t in (N-1):1) {
    # if riccardi, stop updating K once it converges
    K.tplus1 <- K.list[t+1][[1]]
    # check to see if we have converged
    K.tplus2 <- K.list[t+2][[1]]
    K.list[[t]] <- if ((riccardi == FALSE) || (!(t == N-1) && !(K.tplus2 == K.tplus1))) {
      # break into parts for readability
      inner <- K.tplus1 - K.tplus1 %*% B %*% solve(R + t(B) %*% K.tplus1 %*% B) %*% t(B) %*% K.tplus1
      t(A) %*% (inner) %*% A + Q
    } else {
      K.tplus1
    }
    L.list[[t]] <- -solve(R + t(B)%*%K.tplus1%*%B) %*% t(B) %*% K.tplus1 %*% A
  }

  # Use L's to solve for optimal control
  # from t = 1 to 100 (e.g. R indices 2 to 101)
  # first component of xs is x0 -> x0, so need to index from t+1
  for (t in 2:N) {
    lastperiod <- t-1
    x.lastperiod <- x.mat[lastperiod,]
    L.lastperiod <- L.list[[lastperiod]]
    w.lastperiod <- ws[lastperiod,]
    # solve for x_{k+1}
    x.mat[t,] <- A %*% x.lastperiod + B %*% (L.lastperiod %*% x.lastperiod) + w.lastperiod
  }

  return(list(x.matrix = x.mat))
}
```

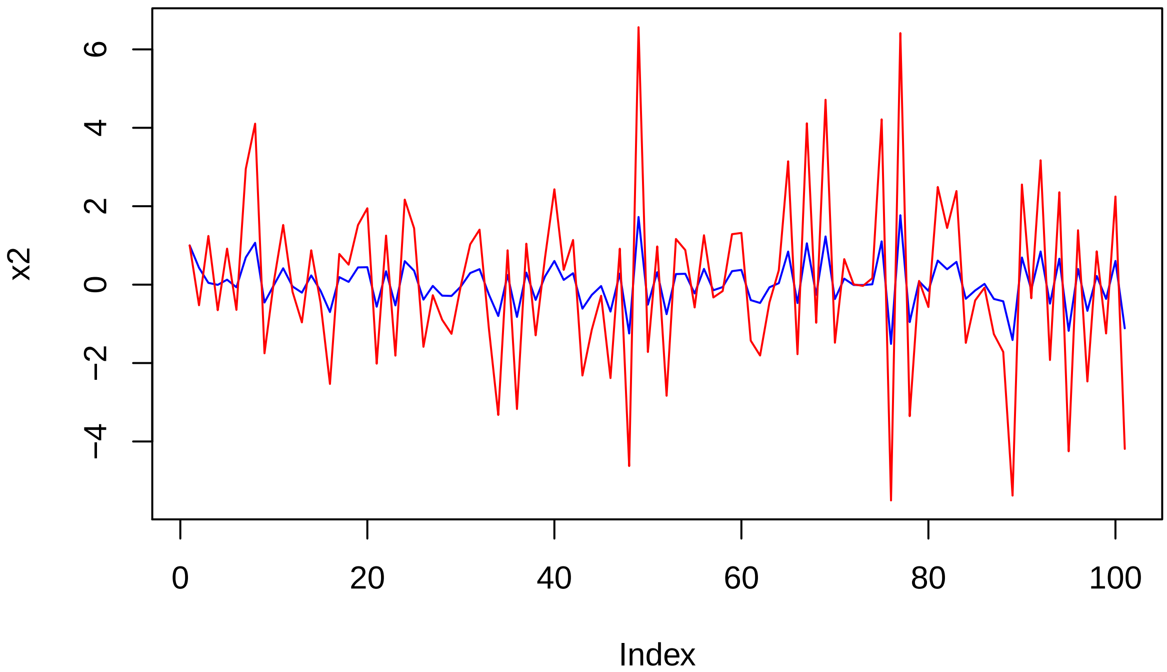
We use this function with modified arguments below:

(i) Fix R and x_0, and compare the behavior of the system for two covariance matrices for the disturbances, one "much larger" than the other, under optimal control (given by the discrete-time Riccati equation) small

```r
first.q.small <- get.states()
# explicitly state N for clarity
N = 101
first.q.large <- get.states(N = N, ws = rmvnorm(N-1, mean = c(0,0), sigma = diag(x = c(1.2,3.2))))

min.x1 <- min(first.q.large$x.matrix[,1])
max.x1 <- max(first.q.large$x.matrix[,1])
plot(first.q.small$x.matrix[,1],
     type = 'l',
     col = 'blue',
     ylim = c(min.x1, max.x1),
     ylab = 'x1')
lines(first.q.large$x.matrix[,1], type = 'l', col = 'red')
```

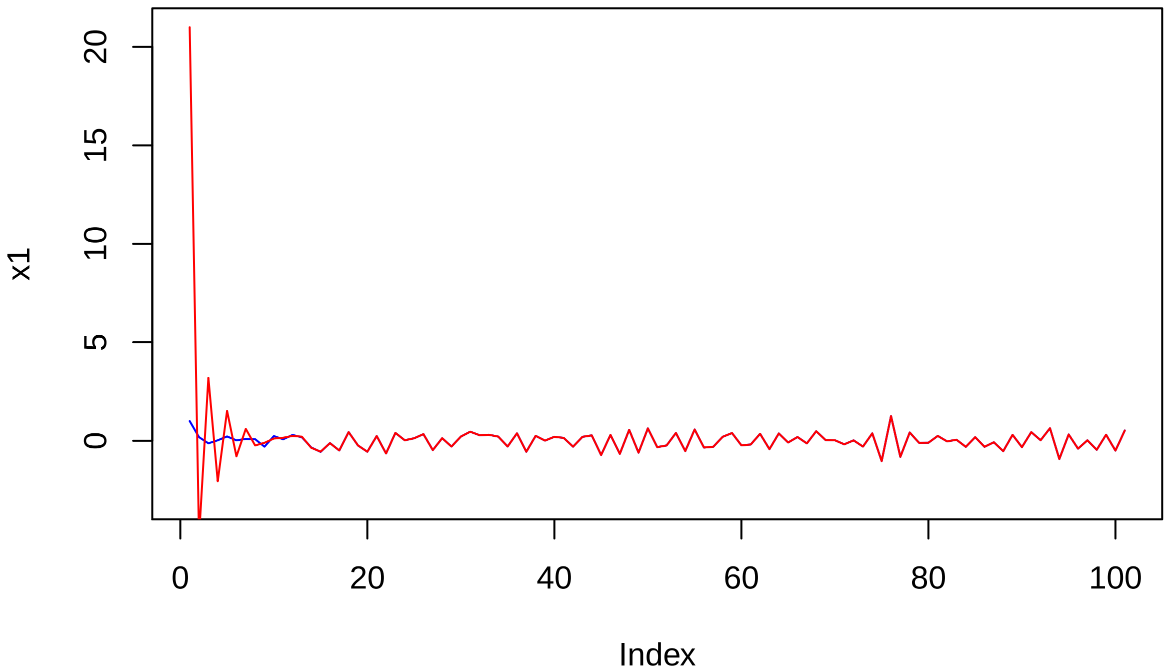



```r
min.x2 <- min(first.q.large$x.matrix[,2])
max.x2 <- max(first.q.large$x.matrix[,2])
plot(first.q.small$x.matrix[,2],
     type = 'l',
     col = 'blue',
     ylim = c(min.x2, max.x2),
     ylab = 'x2')
lines(first.q.large$x.matrix[,2], type = 'l', col = 'red')
```

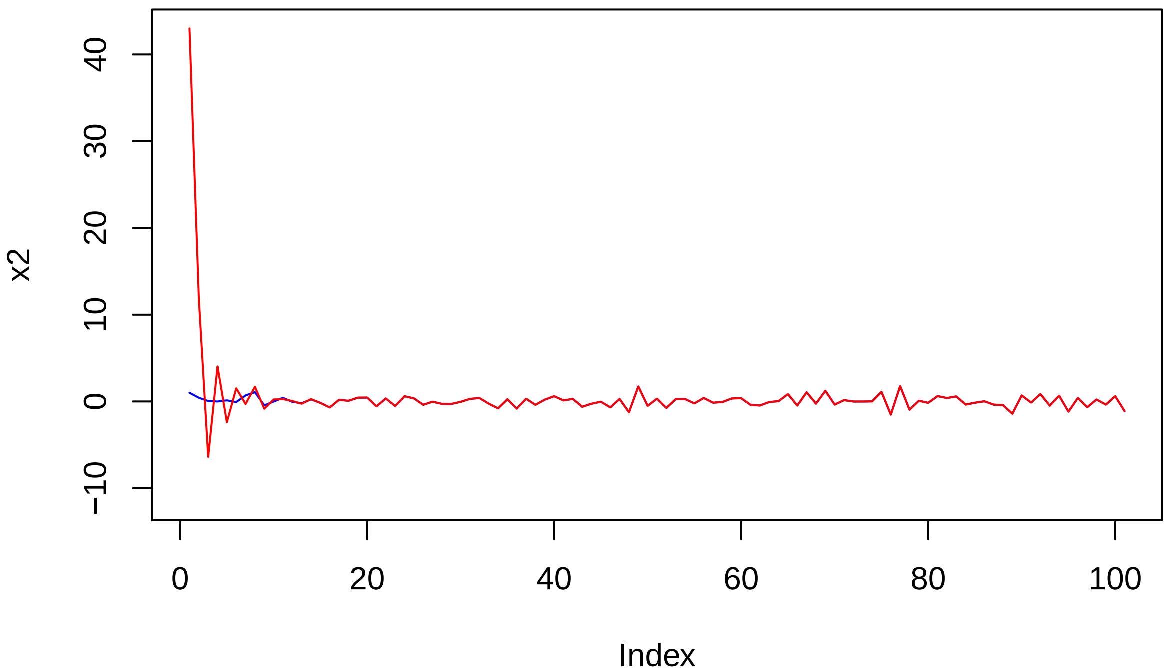


(ii) Fix R and D, and compare the behavior of the system for two initial conditions; one "much larger" than the other, under optimal control;

```
second.q.small <- get.states()
second.q.large <- get.states(x0 = c(21,43))
min.x1 <- min(second.q.small$x.matrix[,1])
max.x1 <- max(second.q.large$x.matrix[,1])
plot(second.q.small$x.matrix[,1],
     type = 'l',
     col = 'blue',
     ylim = c(min.x1-2, max.x1),
     ylab = 'x1')
lines(second.q.large$x.matrix[,1], type = 'l', col = 'red')
```

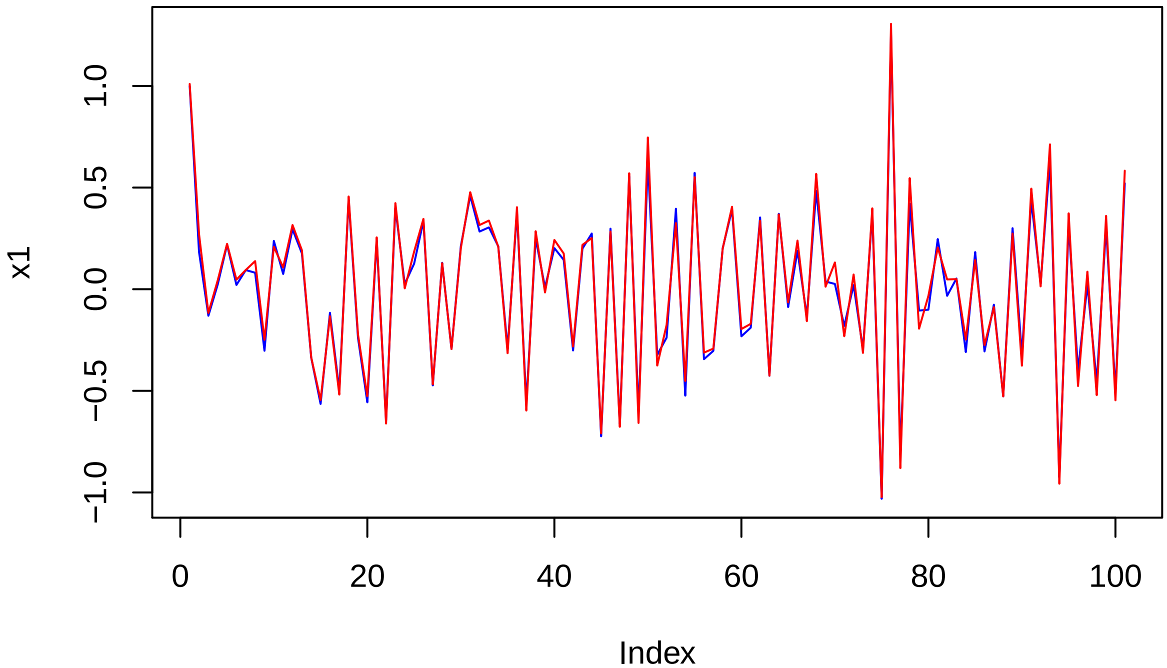



```
min.x2 <- min(second.q.small$x.matrix[,2])
max.x2 <- max(second.q.large$x.matrix[,2])
plot(second.q.small$x.matrix[,2],
     type = 'l',
     col = 'blue',
     ylim = c(min.x2-10, max.x2),
     ylab = 'x2')
lines(second.q.large$x.matrix[,2], type = 'l', col = 'red')
```

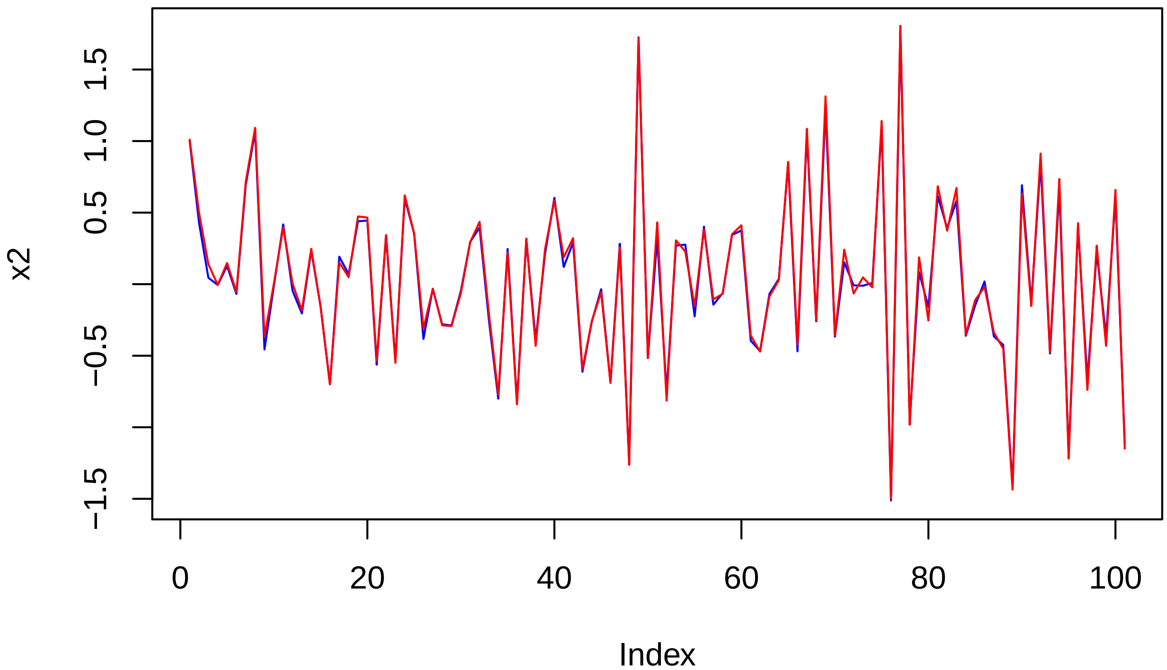


(iii) Fix x_0 and D, and compare the behavior of the system for two input-cost matrices, one "much larger" than the other, under optimal control;

```r
third.q.small <- get.states()
third.q.large <- get.states(R = diag(x = c(99,23)))
min.x1 <- min(third.q.small$x.matrix[,1])
max.x1 <- max(third.q.large$x.matrix[,1])
plot(third.q.small$x.matrix[,1],
     type = 'l',
     col = 'blue',
     ylim = c(min.x1, max.x1),
     ylab = 'x1')
lines(third.q.large$x.matrix[,1]+1e-2, type = 'l', col = 'red')
```

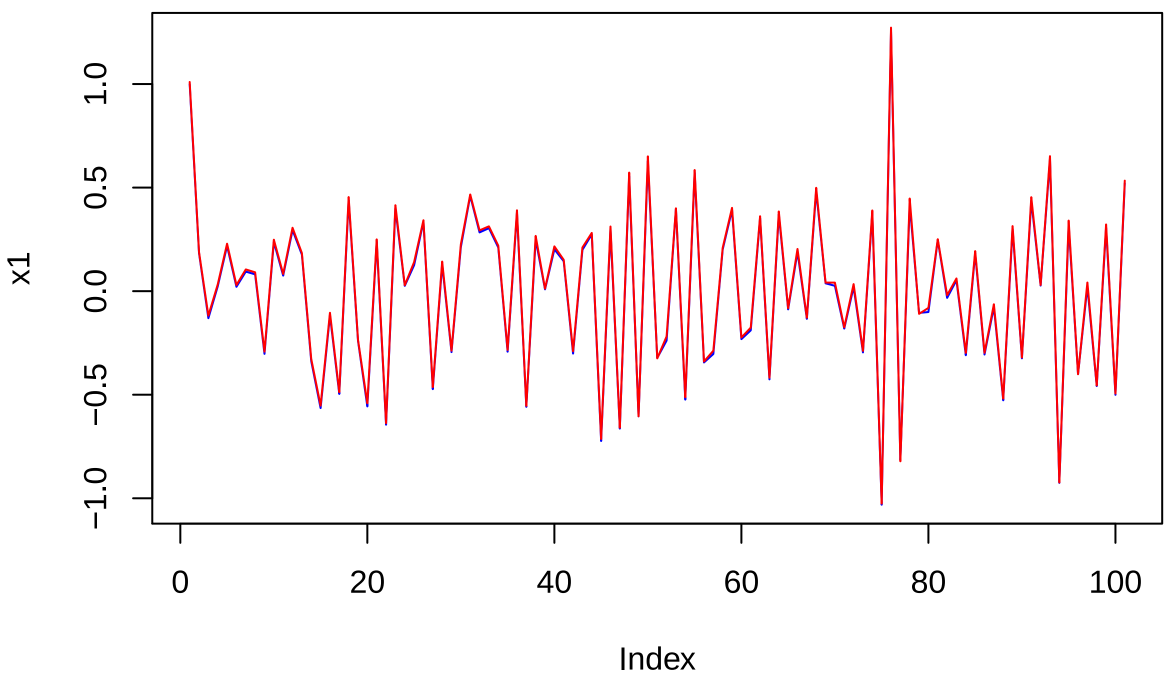



```r
min.x2 <- min(third.q.small$x.matrix[,2])
max.x2 <- max(third.q.large$x.matrix[,2])
plot(third.q.small$x.matrix[,2],
     type = 'l',
     col = 'blue',
     ylim = c(min.x2, max.x2),
     ylab = 'x2')
lines(third.q.large$x.matrix[,2]+1e-2, type = 'l', col = 'red')
```



(iv) Fix R, x_0, and D, and compare the behavior of the system under optimal control vs. steady-state control (given by the algebraic Riccati equation).

```
fourth.q.normal <- get.states(riccardi = FALSE)
fourth.q.riccardi <- get.states(riccardi = TRUE)
plot(fourth.q.normal$x.matrix[,1],
     type = 'l',
     col = 'blue',
     ylab = 'x1')
lines(fourth.q.riccardi$x.matrix[,1]+1e-2, type = 'l', col = 'red')
```



```r
plot(fourth.q.normal$x.matrix[,2],
     type = 'l',
     col = 'blue',
     ylab = 'x2')
lines(fourth.q.riccardi$x.matrix[,2]+1e-2, type = 'l', col = 'red')
```