Problem set 4 - Exercise 5

Remarks on the following results

This file contains two different functions to produce the results for each data specifiation.

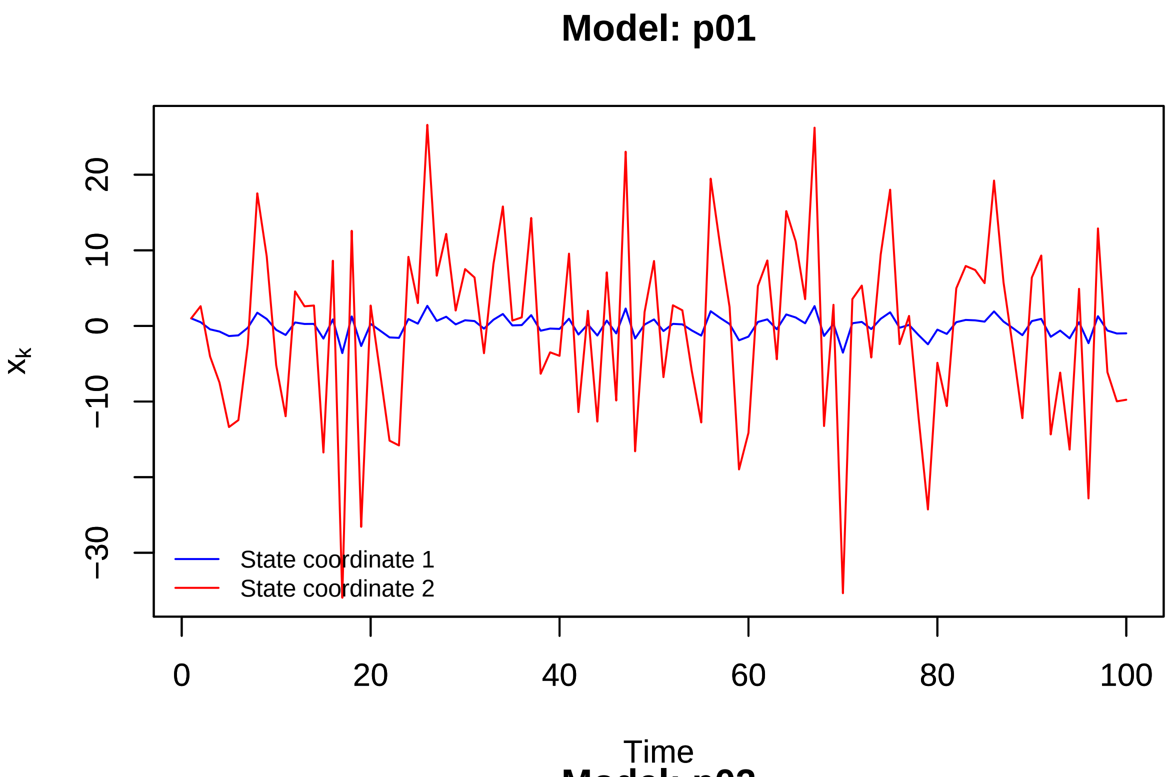
1) compute.states( )
2) rigati.states( )

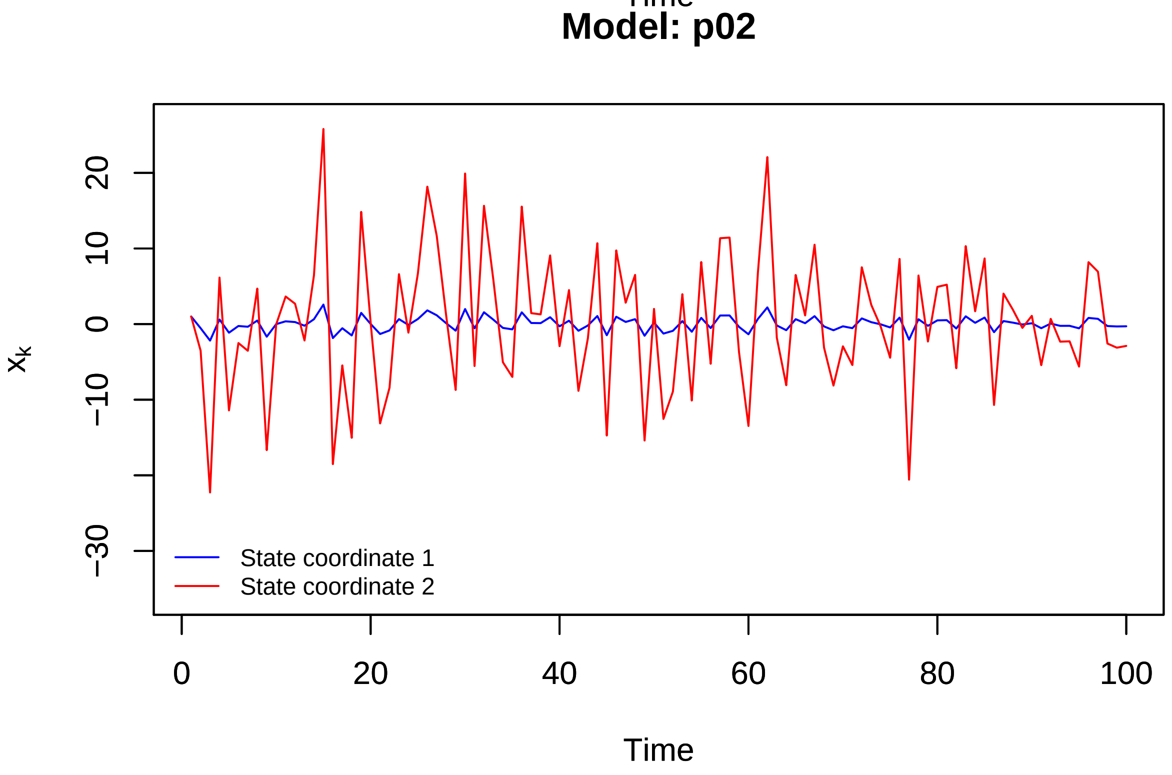
The first function computes the states by dynamically updating both K and L. The second function produces the result applying the algebraic rigati equation. Both function spit out a $Nx2$ matrix with all states. The complete code for both functions can be found in the appendix section. If not explicatly stated otherwise the function uses the following values as default function arguments.

```
# Defaults
N  = 100
x0 = c( 1, 1 )
D  = diag(2) * c(1,1)
R  = diag(2) * c( 1, 2 )
A  = matrix( c(0,0,1,0), 2, 2)
B  = matrix( c(3,0,3,2), 2, 2)
C  = c( 1, 1 )
```

Specifiaction 1 - Comparison of different covariances

```
# Initializecovariance matrices
D1  <- diag( 2 ) * c( 1 ,   1 )
D2  <- diag( 2 ) * c( 100 , 100 )

# Compute states for both covariance matrices
p01 <- compute.states( D = D1  )
p02 <- compute.states( D = D2  )
```

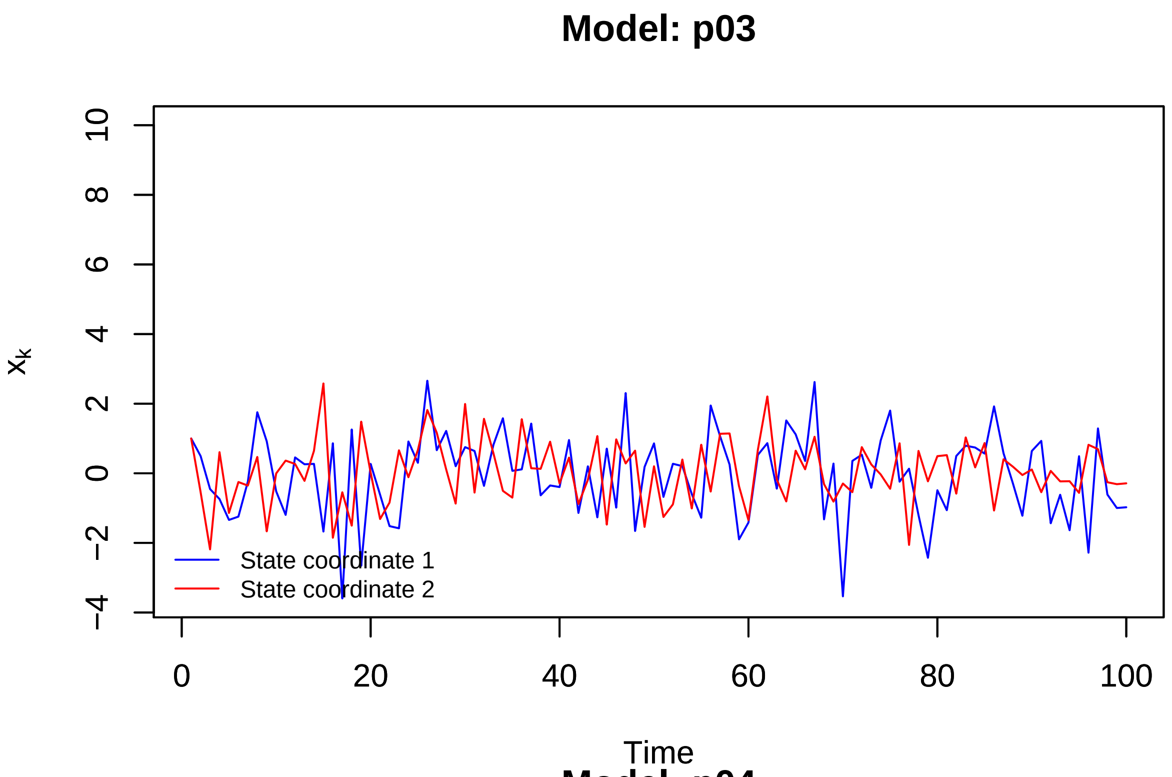


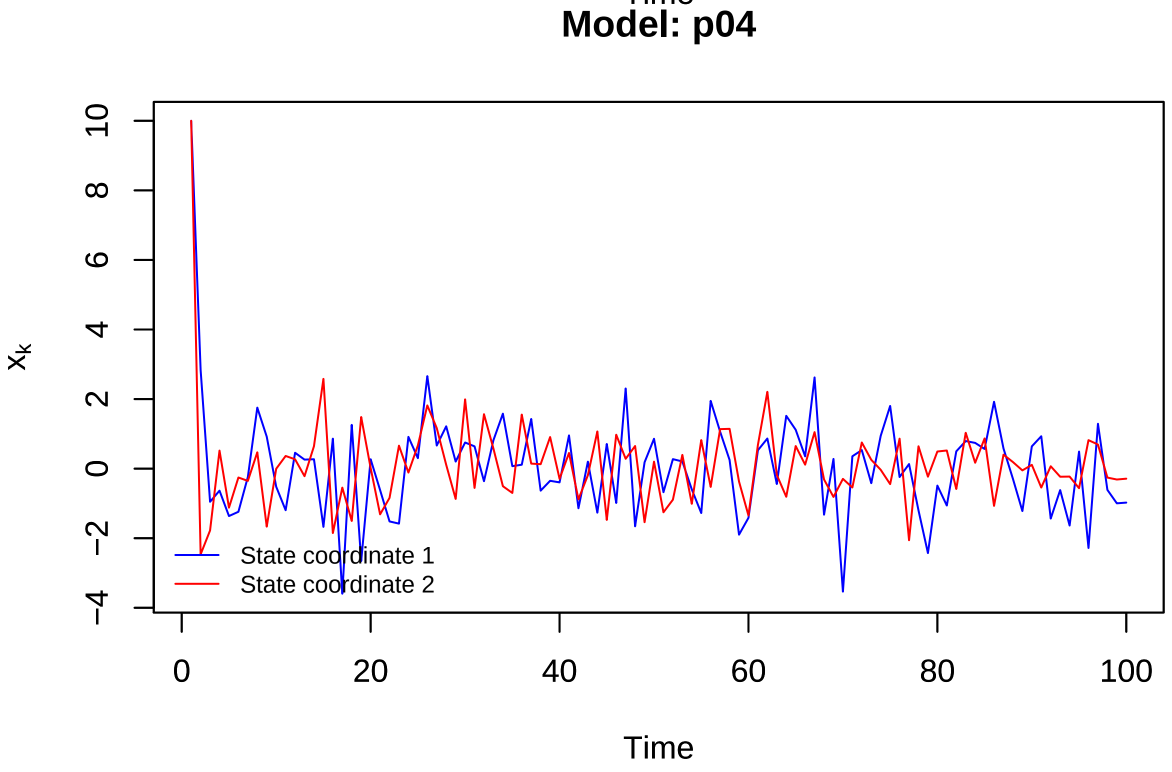


Specification 2 - Comparison of different starting states

```r
# Set different levels of starting state
x1 <- c( 1 , 1)
```

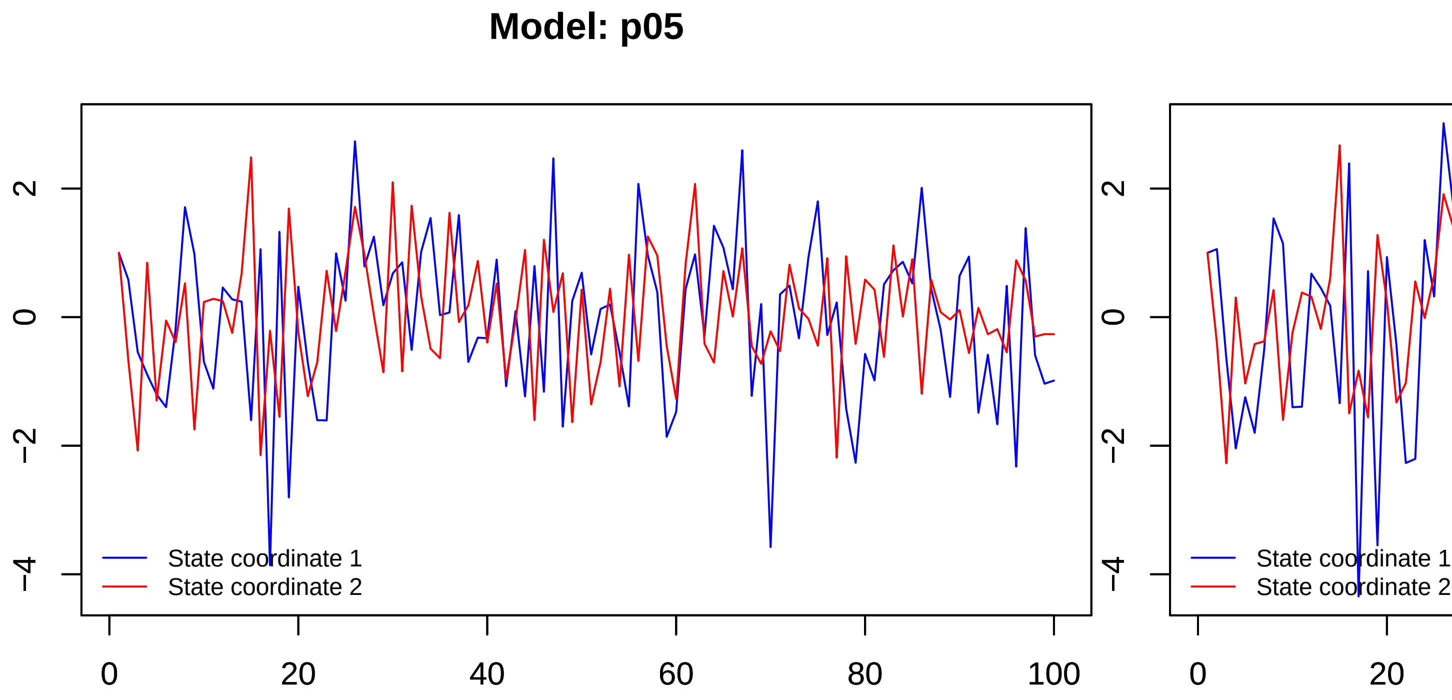
```r
x2 <- c( 10, 10)

# Compute states for different starting states
p03 <- compute.states( x0 = x1,A = A, B = B  )
p04 <- compute.states( x0 = x2,A = A, B = B  )
```





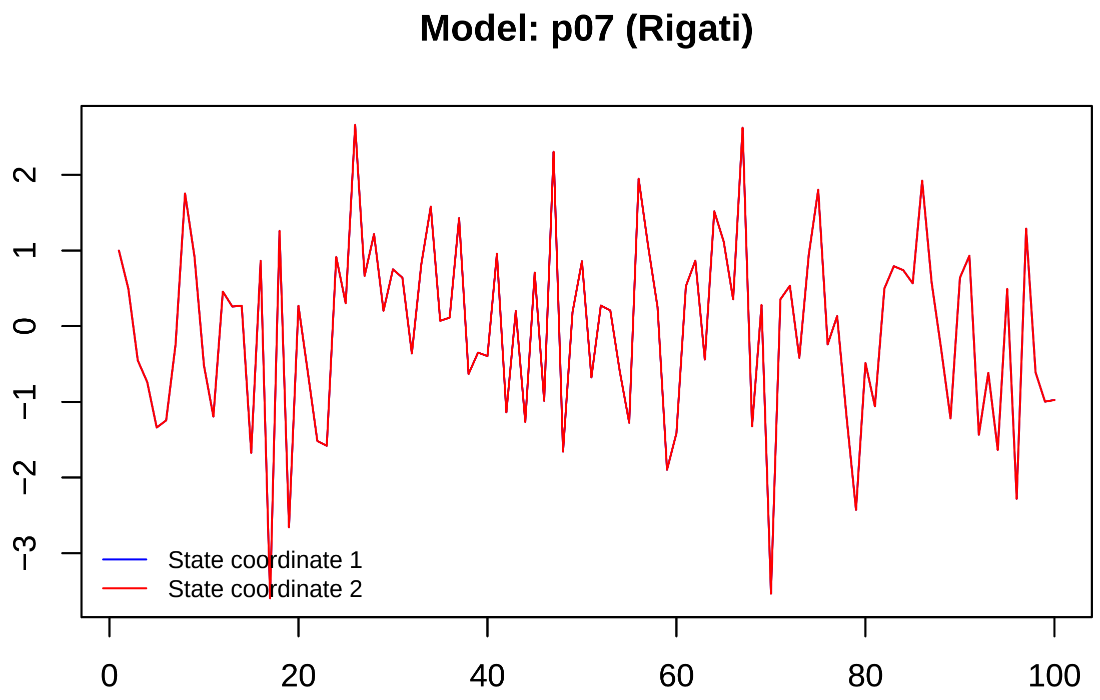
Specification 3 - Comparison of different input-cost

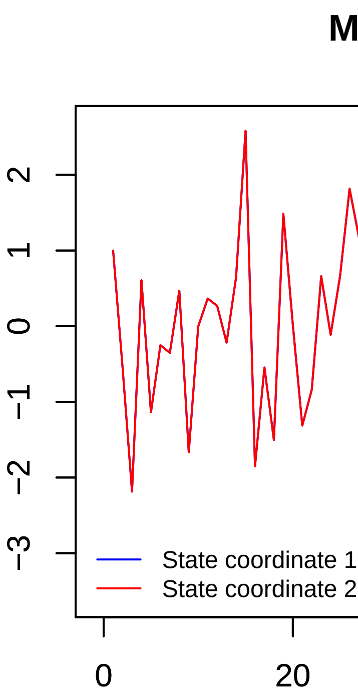
```
# Initialize different input-cost matrices
R1 <- diag(2) * c(0.5,0.2)
R2 <- diag(2) * c(100,100)

# Compute states for different input-cost matrices
p05 <- compute.states( A = A, B = B, R = R1  )
p06 <- compute.states( A = A, B = B, R = R2  )
```



Specification 4 - System under optimal control vs. steady-state control

```
# Compute states for different optimal control and steady-state control
p07 <- rigati.states(   )
p08 <- compute.states(  )
```





Appendix 1 - Code compute.states function

```
compute.states <- function(  N  = 100,
                             x0 = c( 1, 1 ),
                             D  = diag(2) * c(1,1),
                             R  = diag(2) * c( 1, 2 ) ,
                             A  = matrix( c(0,0,1,0), 2, 2),
                             B  = matrix( c(3,0,3,2), 2, 2),
                             C  = c( 1, 1 ) )
{
# Set mean for disturbance
mu      <- c( 0 , 0 )
# Initialize storage objets and set corresponding starting conditions
x       <- matrix( NA , nrow = N , ncol = 2 )
x[1,]   <- x0
Q       <- C %*% t(C)
K       <- list()
K[[N]]  <- Q
L       <- list()

# Compute K and L matrices
for (k in (N-1):1){

K[[k]] <-   t(A) %*% ( K[[(k+1)]] - K[[(k+1)]] %*% B %*% solve( t(B) %*% K[[(k+1)]] %*% B + R ) %*% t(B
L[[k]] <- - solve( t(B) %*% K[[(k+1)]] %*% B + R) %*% t(B) %*% K[[(k+1)]] %*% A

}

# Solve for the states
set.seed(111)
w       <- rmvnorm( 100 , mean = mu , sigma = D )
for( k in 2:N ){

u.temp <- L[[(k - 1)]] %*% x[(k-1),]
x[k,]  <- A %*% x[(k-1),] + B %*% u.temp + w[(k-1),]

}

# Return states
return(x)

}
```

Appendix 2 - Code rigati.states function

```
rigati.states <- function(   N  = 100,
                             x0 = c( 1, 1 ),
                             D  = diag(2) * c(1,1),
                             R  = diag(2) * c( 1, 2 ) ,
                             A  = matrix( c(0,0,1,0), 2, 2),
                             B  = matrix( c(3,0,3,2), 2, 2),
                             C  = c( 1, 1 )  )
{
# Set mean for disturbance
mu      <- c( 0 , 0 )
# Initialize storage objets and set corresponding starting conditions
x       <- matrix( NA , nrow = N , ncol = 2 )
x[1,]   <- x0
Q       <- C %*% t(C)
K  <-list()
K[[N]]  <- Q

# Compute K and L matrices
for (k in (N-1):1){

K[[k]] <-   t(A) %*% ( K[[(k+1)]] - K[[(k+1)]] %*% B %*% solve( t(B) %*% K[[(k+1)]] %*% B + R ) %*% t(B

if(  sum( K[[k]] - K[[(k+1)]] )  == 0 ){

      K <- K[[(k+1)]]
      break
}

}
set.seed(111)
w <- rmvnorm( 100 , mean = mu , sigma = D )
L <- - solve( t(B) %*% K  %*% B + R) %*% t(B) %*% K %*% A
# Compute states
for( k in 2:N ){

u.temp <- L %*% x[(k-1),]
```

```r
x[k,]  <- A %*%  x[(k-1),]  + B %*% u.temp + w[(k-1),]

}

# Return states
return(x)

}
```