Chapter 8: Control theory illustration using dynr

Sy-Miin Chow

Jan 22, 2022

Contents

Overview

A dynamic system is said to be "controllable" if the internal, underlying values of the system can be driven or directed toward a specific target level within a finite time interval regardless of its initial values through the use of one or more exogenous inputs. Examples may include the use of insulin (the "input") to drive the glucose level of a diabetic patient toward a target glucose level. This tutorial implements a type of control theory algorithm, called Linear-Quadratic Controller, using intermediate output from the dynr package. This tutorial is also featured as an example in Gates, Chow, and Molenaar (in progress).

Linear-Quadratic Controller is one special case of a class of control strategies called receding horizon controls (RHCs) or model predictive controls. This special case assumes that the true values of a system's latent underlying (i.e., state) values are known, but instead, have to be estimated from noisy observed variables. Details can be found in Kwon and Ham (2005), and Molenaar (2010).

References:

Gates, K., Molenaar, P. C. M., & Chow, S.-M. (in progress). Analysis of Intra-individual Variation. New York: Taylor & Francis.

Goodwin, G., Seron, M. M., & de Don, J. A. (2005). Constrained control and estimation: An optimisation approach (1st ed.). London, United Kingdom: Springer-Verlag London Ltd.

Kwon, W. H., & Han, S. H. (2005). Receding horizon control: model predictive control for state models. London: Springer.

Molenaar, P. C. (2010). Note on optimization of individual psychotherapeutic processes. Journal of Mathematical Psychology, 54 (1), 208{213.

Wang, Q., Molenaar, P., Harsh, S., Freeman, K., Xie, J., Gold, C., . . . Ulbrecht, J. (2014a). Personalized state-space modeling of glucose dynamics for type-1 diabetes using continuously monitored glucose, insulin dose, and meal intake: An extended kalman filter approach. Journal of Diabetes Science and Technology, 8 (2), 331-345. Retrieved from http://dst.sagepub.com/ content/8/2/331.abstract doi: 10.1177/1932296814524080

Load dependent R packages and set up options

```
## Loading required package: mvtnorm

## Loading required package: ggplot2

## Loading required package: dynr

## Loading required package: dplyr

##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: plyr

## ------------------------------------------------------------------------------

## You have loaded plyr after dplyr - this is likely to cause problems.
## If you need functions from both plyr and dplyr, please load plyr first, then dplyr:
## library(plyr); library(dplyr)

## ------------------------------------------------------------------------------

##
## Attaching package: 'plyr'

## The following objects are masked from 'package:dplyr':
##
##     arrange, count, desc, failwith, id, mutate, rename, summarise,
##     summarize

## Loading required package: gridExtra

##
## Attaching package: 'gridExtra'

## The following object is masked from 'package:dplyr':
##
##     combine

## Loading required package: zeallot
```

Sourcing some handy functions

```
source('ControllerFunctions.R')
```

Generate data using a VAR(1)-X model

In this tutorial, we use a group-based vector autoregressive model of order 1 (lag 1) with one exogenous input variable, $z_i(t)$. That is, we are using a VAR(1)-X model, with both process noises and measurement errors, and we assume that all units (e.g., individuals) within the group can be described by the same model with the same parameters.

Further more, we implement a scenario where the the system's desired target level is at the value of 0, but we incur unusually large shocks at some randomly selected, individual-specific time points such that the system is "perturbed" and pushed away even further from its target level than it normally would be. We show that by applying the optimal input values calculated by the controller, the system can return to its target level more quickly than if we just let the system run its course and "recover" at its own rate.

The VAR-X model is specified in state-space form as: Measurement model:

$$V_{1i}(t) = \eta_{1i}(t) + \epsilon_{1i}(t)$$

$$V_{2i}(t) = \eta_{2i}(t) + \epsilon_{2i}(t)$$

$$[\epsilon_{1i}(t), \epsilon_{2i}(t)]' \sim N\left([0,0]', \text{diag}[v_{e1}, v_{e2}]\right)$$

Dynamic model:

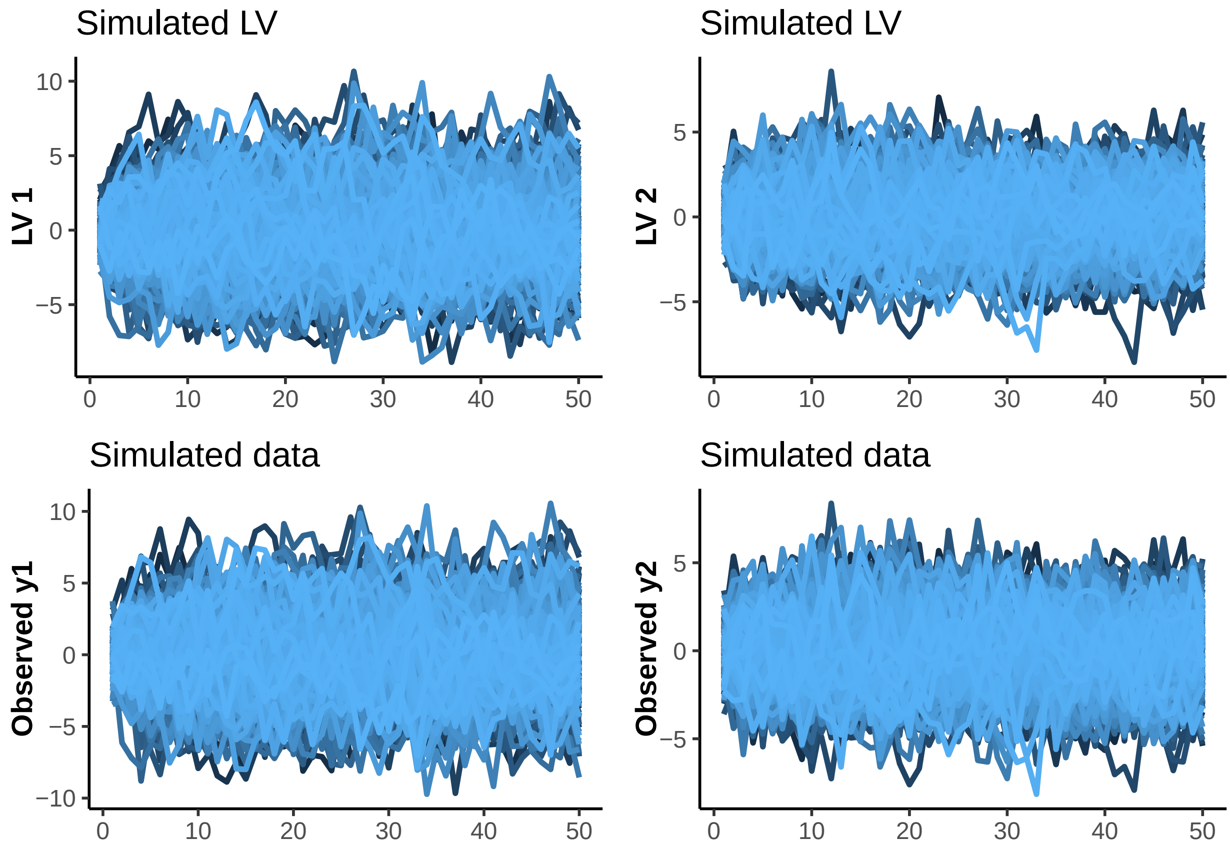
$$\eta_{1i}(t) = \phi_{11}\eta_{1i}(t-1) + \phi_{12}\eta_{2i}(t-1) + g_1 z_i(t-1) + \zeta_{1i}(t)$$

$$\eta_{2i}(t) = \phi_{21}\eta_{1i}(t-1) + \phi_{22}\eta_{2i}(t-1) + g_2 z_i(t-1) + \zeta_{2i}(t)$$

$$[\zeta_{1i}(t), \zeta_{2i}(t)]' \sim N\left([0,0]', \begin{pmatrix} \psi_{11} & \psi_{12} \\ \psi_{12} & \psi_{22} \end{pmatrix}\right)$$

```
noTime      <-  50  #Total number of time points
npad      <- 0 #Number of occasions to throw out to wash away
               #the effects of initial condition
noSubj        <- 300 #Total number of participants
noStates        <- 2 #Number of latent variables
noObs        <- 2 #Number of manifest variables
noInput         <- 1 # Number of input variables
psi       <- matrix(c(2, .5,  # Process noise variance-covariance matrix
                      .5, 1.5),
                    ncol = noStates, byrow = TRUE)
lambda    <- matrix(c(1, 0, # Lambda matrix containing contemporaneous relations among
                      0, 1),
                    ncol = noStates, byrow = TRUE)
theta     <- diag(.5, ncol = noObs, nrow = noObs)  # Measurement error variances.
Phi_SS      <- matrix(c(0.7, -.3,    # Lagged directed relations among latent variables.
                      -.2, .6),
                    ncol = noStates, byrow = TRUE)
a0        <- c(0,0) #Means of latent variables at initial time point
P0        <- matrix(c(1,0,
                      0,1),
                    ncol=noStates,byrow=TRUE) #Initial latent variable covariance matrix
eta_intercept <- matrix(c(0,0),ncol=1)
G_SS <- matrix(c(-0.7, -.4), ncol=noInput, byrow = TRUE)
```

```
zt <- matrix(0,nrow=noTime*noSubj,ncol=noInput)
for (p in 1:noSubj){
  zt[(1+(p-1)*noTime):(p*noTime),] = matrix(rnorm(noTime,0,1),ncol=noInput)
}

file0 = paste0('VARy.txt')
file1 = paste0('VAReta.txt')
shocks = matrix(0,noSubj*noTime,noStates)


testData = GenerateData(noTime,npad,noSubj,noStates,noObs,noInput,psi,lambda,theta,
           Phi_SS,eta_intercept,G_SS,a0,P0,zt,fileObs=file0,
           fileState=file1,shocks)
```



Testing the controllability of a system

Here, we will illustrate how to construct the controllability matrix and check its rank. We will first consider:

$$\Phi_{SS} = \begin{bmatrix} 0.7 & -0.3 \\ -0.2 & 0.6 \end{bmatrix}$$

$$G_{SS} = \begin{bmatrix} -0.7 \\ -0.4 \end{bmatrix}$$

, and then setting: the cross-regression parameters in Φ_{SS} (i.e., the [1,2] and [2,1] elements) and the second weight in G_{SS} to be zero.

```
#Since noStates = 2 in this example, we need to raise Phi_SS to the power of 1 only.
#The controllability matrix is of dimension noStates x noStates = 2 x 2
C = matrix(cbind(G_SS,Phi_SS%*%G_SS),ncol=noStates,byrow=TRUE)
qr(C)$rank
```

```
## [1] 2
```

```
#Full rank

#Now let's try G_SS = matrix(c(.7,0),ncol=1) and setting the cross-regression
#parameters to 0
Phi_SS2 = Phi_SS
Phi_SS2[1,2] = 0
Phi_SS2[2,1] = 0
G_SS2 = matrix(c(.7,0),ncol=1)
cat(paste0('Phi_SS2 = ', Phi_SS2));
```

```
## Phi_SS2 = 0.7 Phi_SS2 = 0 Phi_SS2 = 0 Phi_SS2 = 0.6
```

```
cat(paste0('G_SS = ', G_SS));
```

```
## G_SS = -0.7 G_SS = -0.4
```

```
C2 = matrix(cbind(G_SS2,Phi_SS2%*%G_SS2),ncol=noStates,byrow=TRUE)
qr(C2)$rank
```

```
## [1] 1
```

```
#Rank is < 2; This system is not controllable
```

Specify VAR(1)-X model in *dynr* and cook it.

Specify the *dynr* code to fit the VAR(1)-X model

```
## Read in data and set up data structure in dynr
etaall = read.table(file1,header=FALSE,sep=",")
thedat = read.table(file0,header=FALSE,sep=",")
colnames(thedat) = c("ID","Time",paste0("V",1:noObs),"z1")

## Manually shift the covariates because in dynr, eta_t is matched with u_t in the
#dynamic model
thedat <- plyr::ddply(thedat, .(ID), mutate,
                      zlag1 = dplyr::lag(z1,1,default=0))

## Set up dynr Data
thedat2 <- dynr.data(thedat, id="ID", time="Time",
                     observed=paste0("V",1:noObs),covariates="zlag1")

## VAR(1) recipe specification code starts here

## Prepare the dynamic model
formula=list(
  eta1~ phi11*eta1 + phi12*eta2 + g1*zlag1,
  eta2~ phi21*eta1 + phi22*eta2 + g2*zlag1)

dynamics<-prep.formulaDynamics(formula=formula,
                               startval=c(phi11=.5,phi22=.5,
                                          phi12=-.1,phi21=-.1,
                                          g1=-.5, g2=-.5),
                               isContinuousTime=FALSE)

## Meausurement model
meas <- prep.measurement(
  #Starting values for factor loading matrix
  values.load=matrix(c(1,0,
                       0,1),ncol=noStates,byrow=TRUE),
  params.load=matrix(c(rep('fixed',noStates),
                       rep('fixed',noStates)),
                     ncol=noStates,byrow=TRUE), #Labels for fixed and freed parameters
  state.names=c("eta1","eta2"), #Labels for latent variables in eta(t)
  obs.names=paste0('V',1:noObs) #Labels for observed variables in y(t)
)

## Initial condition means and covariance matrix. Here we specify them as fixed.
initial <- prep.initial(
  values.inistate=c(0, 0),
  params.inistate=c('fixed','fixed'),
  values.inicov=matrix(c(1,0,
                         0,1),ncol=noStates,byrow=TRUE),
  params.inicov=matrix(c('fixed',0,
                         0,'fixed'),ncol=noStates))


## Process and measurement noise covariance matrices
mdcov <- prep.noise(
  values.latent=matrix(c(2,.5,
                         .5,1.5),ncol=noStates,byrow=TRUE),
  params.latent=matrix(c('psi_11','psi_12',
                         'psi_12','psi_22'),ncol=noStates,byrow=TRUE),
  values.observed=diag(rep(.5,noObs),noObs),
  params.observed=diag(paste0('var_e',1:noObs),noObs)
)

## Put recipes and data together to prepare the full model
model <- dynr.model(dynamics=dynamics, measurement=meas,
                    noise=mdcov, initial=initial,
                    data=thedat2,outfile="VAR.c")
## Cook it!
res <- dynr.cook(model,debug_flag=TRUE,verbose = FALSE)
```

```
## Optimization function called.
## Starting Hessian calculation ...
## Finished Hessian calculation.
## Original exit flag:  3
## Modified exit flag:  3
## Optimization terminated successfully: ftol_rel or ftol_abs was reached.
## Original fitted parameters:  0.705573 0.593608 -0.3030289 -0.2029258 -0.7002757
## -0.4004068 0.6709134 0.2405456 0.3763576 -0.6645845 -0.7833649
##
## Transformed fitted parameters:  0.705573 0.593608 -0.3030289 -0.2029258
## -0.7002757 -0.4004068 1.956023 0.4705127 1.570148 0.5144873 0.4568661
##
## Doing end processing
## Successful trial
## Total Time: 1.867897
## Backend Time: 1.852672
```

```r
coef(res)
```

```
##      phi11      phi22      phi12      phi21         g1         g2     psi_11
##  0.7055730  0.5936080 -0.3030289 -0.2029258 -0.7002757 -0.4004068  1.9560232
##     psi_12     psi_22     var_e1     var_e2
##  0.4705127  1.5701478  0.5144873  0.4568661
```

```r
summary(res)
```

```
## Coefficients:
##         Estimate Std. Error t value  ci.lower  ci.upper            Pr(>|t|)
## phi11   0.705573   0.006487  108.78  0.692860  0.718286 <0.0000000000000002 ***
## phi22   0.593608   0.008787   67.56  0.576386  0.610829 <0.0000000000000002 ***
## phi12  -0.303029   0.008334  -36.36 -0.319364 -0.286694 <0.0000000000000002 ***
## phi21  -0.202926   0.005263  -38.56 -0.213240 -0.192611 <0.0000000000000002 ***
## g1     -0.700276   0.013440  -52.10 -0.726618 -0.673934 <0.0000000000000002 ***
## g2     -0.400407   0.012059  -33.20 -0.424042 -0.376772 <0.0000000000000002 ***
## psi_11  1.956023   0.052265   37.42  1.853585  2.058461 <0.0000000000000002 ***
## psi_12  0.470513   0.019835   23.72  0.431637  0.509388 <0.0000000000000002 ***
## psi_22  1.570148   0.047813   32.84  1.476435  1.663860 <0.0000000000000002 ***
## var_e1  0.514487   0.035314   14.57  0.445273  0.583701 <0.0000000000000002 ***
## var_e2  0.456866   0.035128   13.01  0.388016  0.525716 <0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## -2 log-likelihood value at convergence = 111046.08
## AIC = 111068.08
## BIC = 111151.86
```

Control theory input computation

Here we proceed as follows:

1. We simulate another set of data with the true parameters, and add shocks to particular time points. The shocked latent variables are in $X\_noControl$.
2. Implement the control theory algorithm. Here, a few decisions need to be made:

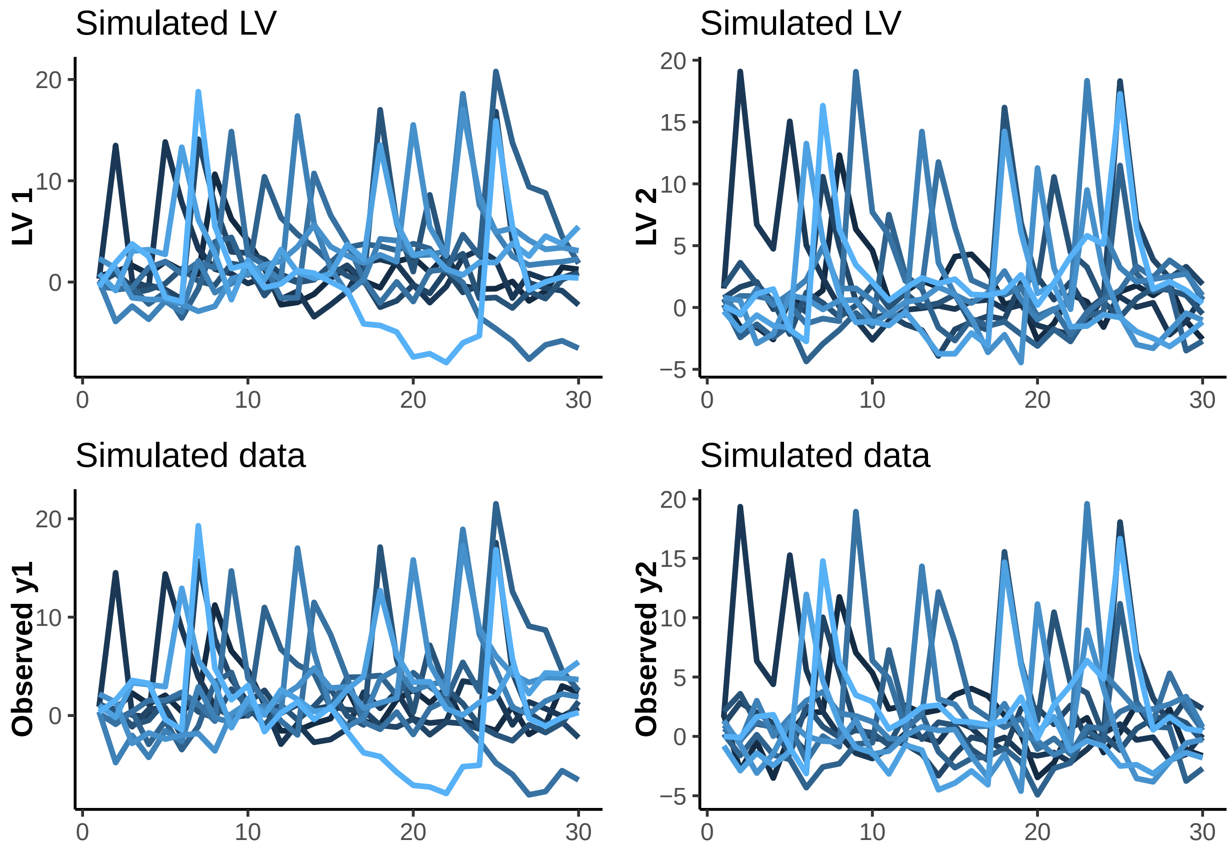
- Values of parameters in the operating state-space model. We set the parameters to those estimated with dynr using the previous test data set (primarily Φ_{SS} and G_{SS}).
- Choice of design matrices (Q, R, Q_f), and control horizon, h. Here, we illustrate how to vary the value of R keeping the values of Q and Q_f constant, and evaluating the consequences using one particular set of cost and benefit functions. We set h to 4 and 10.
- Choice of state estimates. When the true state values are unknown, some options may be to use the filtered state estimates, $E(\eta_i(t)|y_i(1), y_i(2), \ldots, y_i(t))$, or the predicted state estimates, $E(\eta_i(t)|y_i(1), y_i(2), \ldots, y_i(t-1))$, from dynr as state values to implement the controller.

3. Evaluation of results from the controller. In this illustration, we show the root mean squared differences in state values (with and without the controller) compared to the state target levels ($= 0$). We also

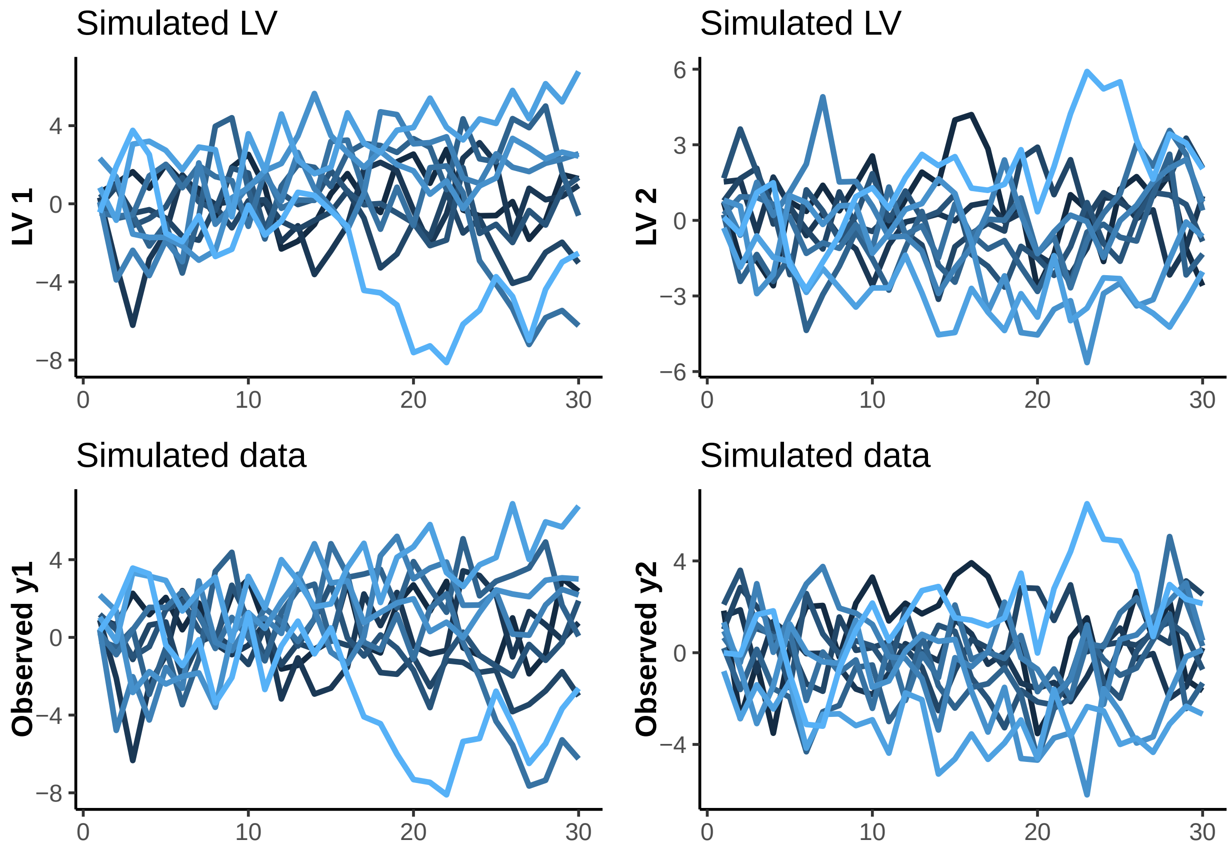
show how to use one possible set of cost and benefit functions to compare the effects of different choices of R.

Step 1. Generate data with shocks

```r
noTime=30; noSubj=10
shockWindow=25
nShocks = 2
#File names to write out and save data
file0 = paste0('yNoShocks.txt')
file1 = paste0('etaNoShocks.txt')
file0.1 = paste0('ywithShocks.txt')
file1.1 = paste0('etawithShocks.txt')
zt <- matrix(rnorm(noTime*noSubj,0,1),nrow = noTime*noSubj, ncol = noInput)
shockPoints = matrix(NA,noSubj,nShocks)
shocks = matrix(0,noSubj*noTime,noStates)

for (i in 1:noSubj){
  shockPoints[i,] = sample(1:shockWindow, nShocks, replace=FALSE)
  shocks[shockPoints[i,]+(i-1)*noTime,1:2] = runif(nShocks*2,10,20)
}

#We set the random number seed this way to generate two sets of data that are
#otherwise identical, except that one set is shocked and another set isn't.
set.seed(12345)
controlData = GenerateData(noTime,npad,noSubj,noStates,noObs,noInput,psi,
                           lambda,theta,Phi_SS,eta_intercept,
                           G_SS,a0,P0,zt,
                           fileObs=file0.1, fileState=file1.1,shocks)
```



```
set.seed(12345)
controlData2 = GenerateData(noTime,npad,noSubj,noStates,noObs,noInput,psi,
                            lambda,theta,Phi_SS,eta_intercept,
                            G_SS,a0,P0,zt,
                            fileObs=file0, fileState=file1)
```



```
#Store latent variable values under shocks and no shocks
X_Shocks = as.matrix(controlData$etaall[,3:4])
X_NoShocks = as.matrix(controlData2$etaall[,3:4])
```

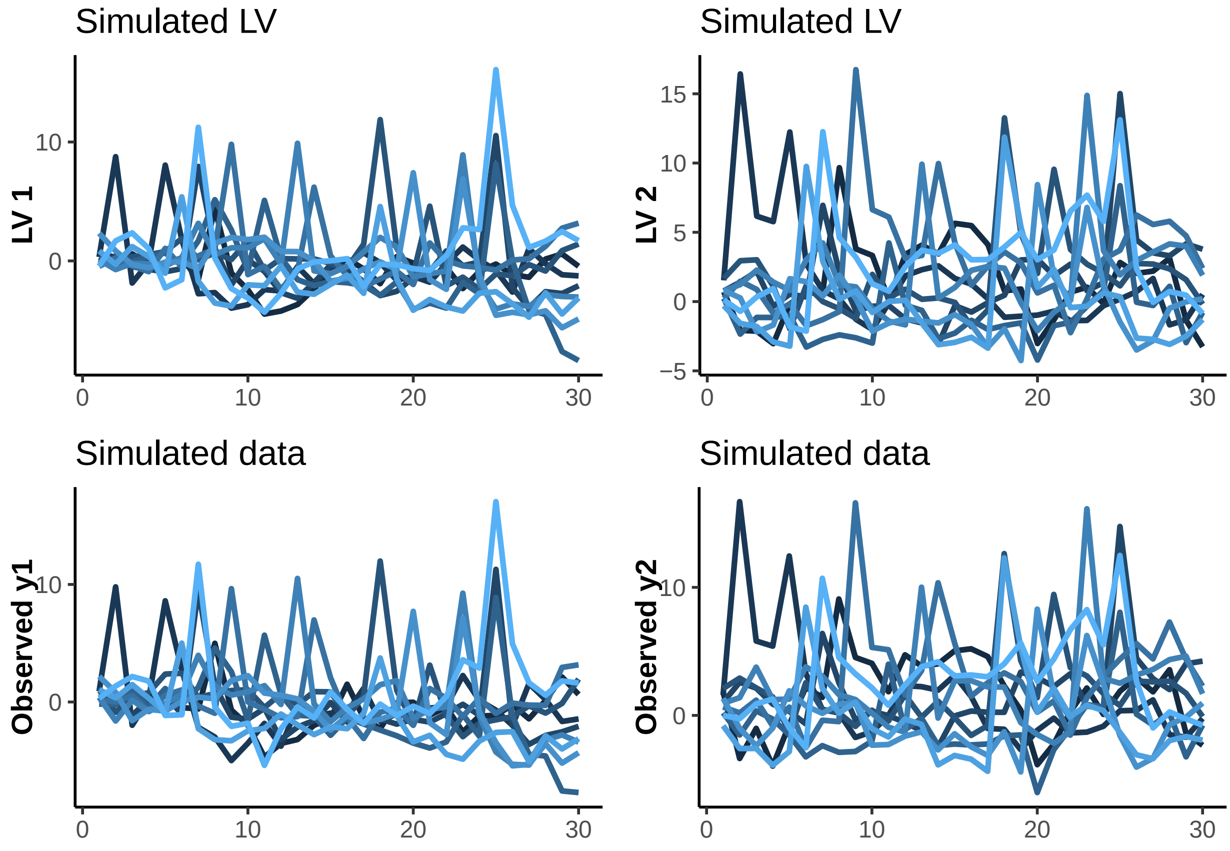
Step 2. Applying the controller to the new data set.

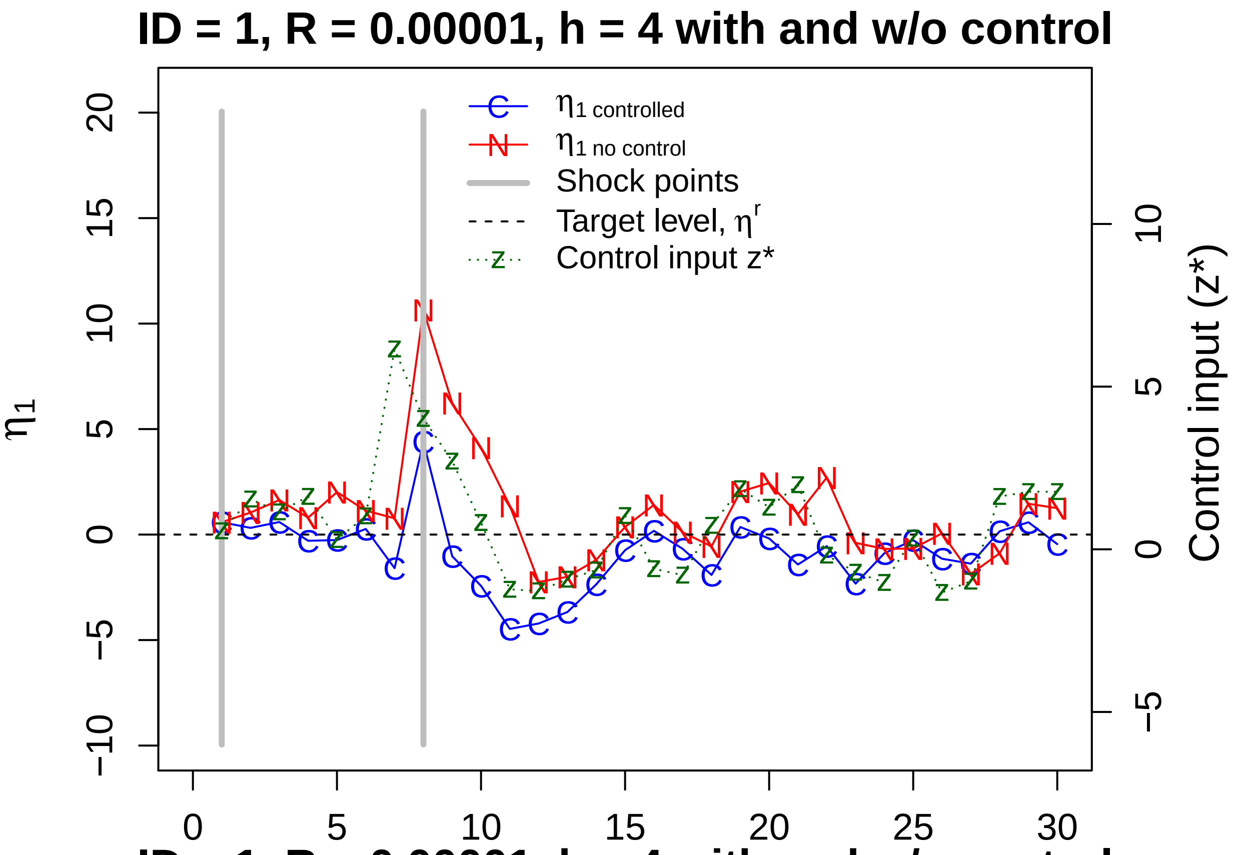
The basic concept of RHC can be summarized as follows. At the current time t, the optimal control is computed, over a finite horizon of a fixed size, $[t, t+h]$, that is, from time points t, $t+1, \ldots, t+h$. Of the control values over this time window, only the one associated with time t is adopted and administered to control the state values of the system at time $t+1$. The procedure is then repeated sequentially for the next time, namely, for time t+1, over the horizon $[t+1, ..., t+1+h]$, to yield a receding possible horizon over which future control values may be administered before the end of the time series.

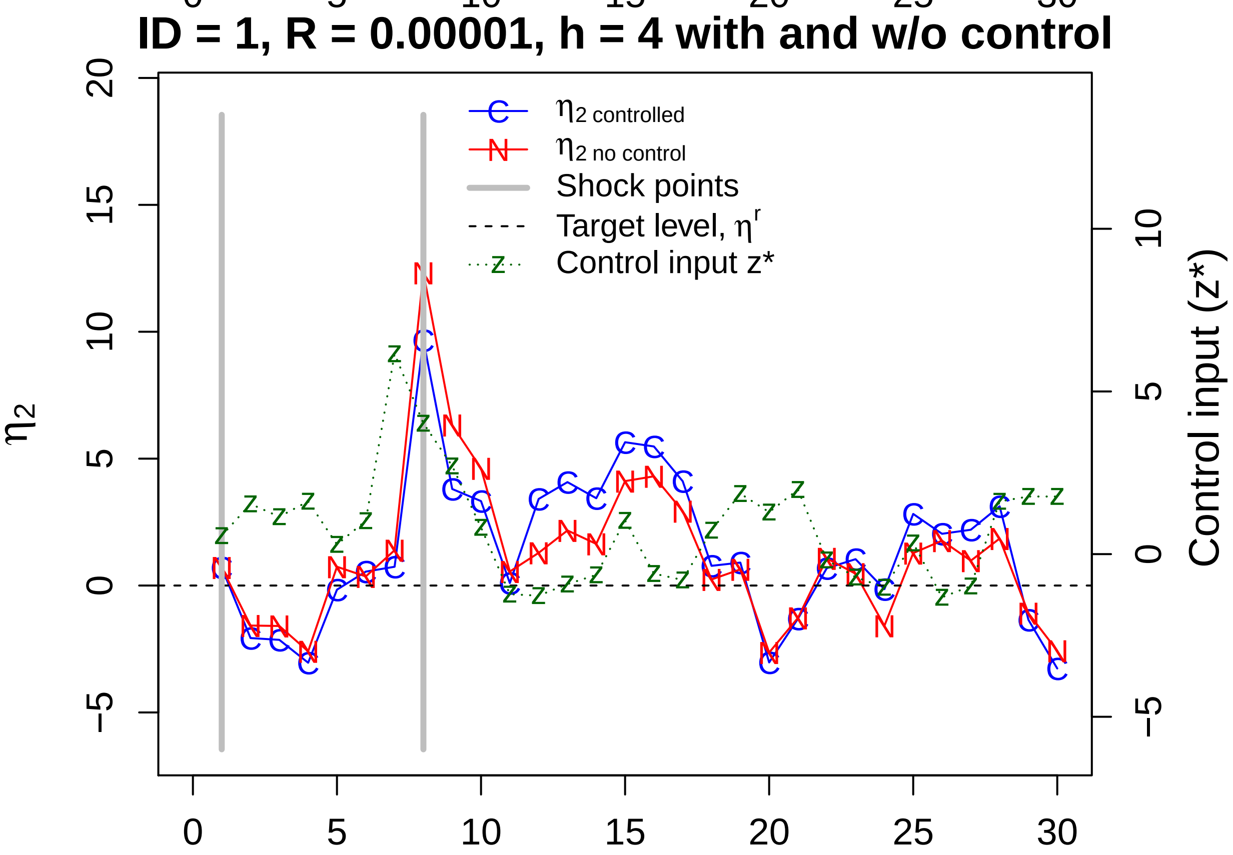
The for loop over Rmult repeats the control theory computations over a range of cost values for administering the control input (R = 0.00001, 0.001, 0.1, 1)

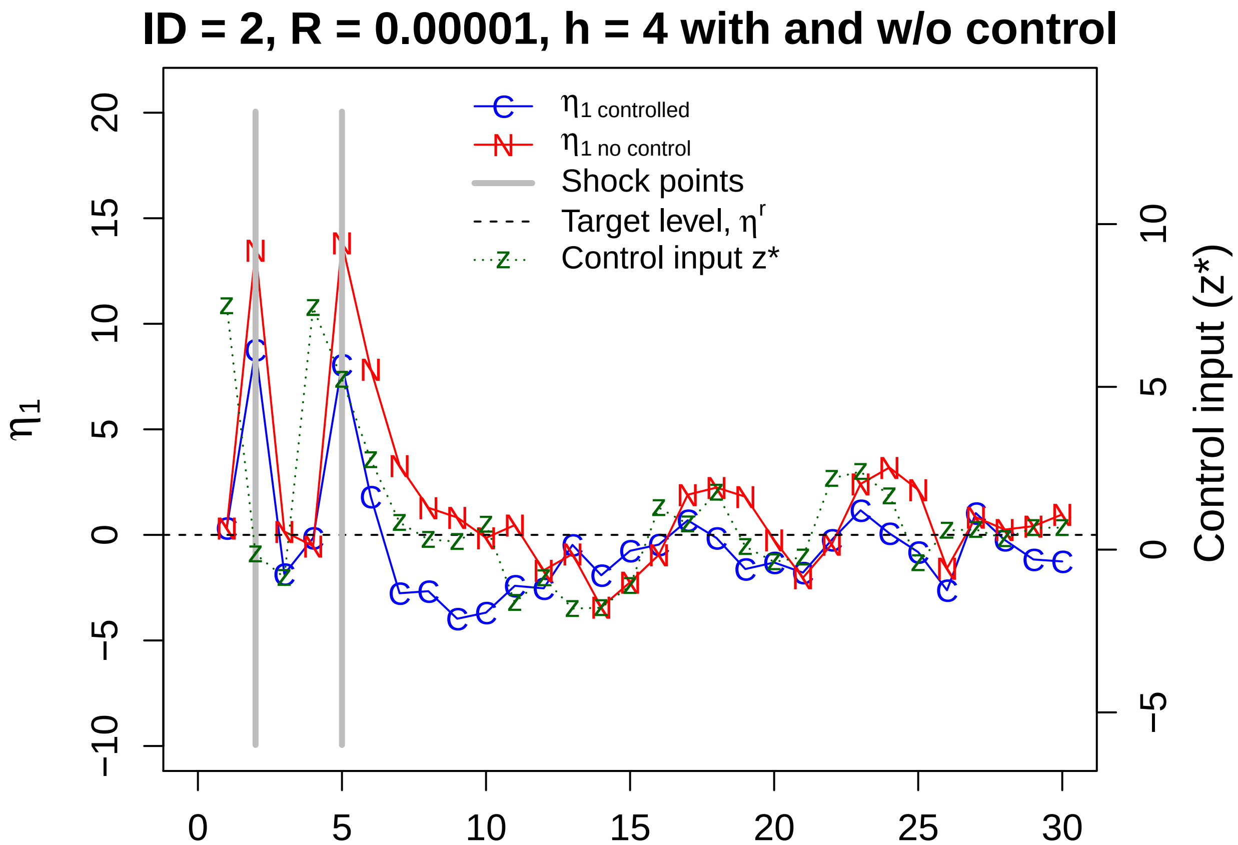
```
#truepar <- c(.5,.5,0,-0.2,.7,.4,2,.5,1.5,rep(.5,noObs))
#Set parameters to those estimated from dynr here
c(phi11, phi22, phi12, phi21, g1, g2, psi_11, psi_12,
  psi_22, var_e1, var_e2) %<-% coef(res)
psi       <- matrix(c(psi_11, psi_12,  # Process noise variance-covariance matrix
                      psi_12, psi_22),
                    ncol = noStates, byrow = TRUE)

Phi_SS = matrix(c(phi11, phi12,
                  phi21, phi22),ncol=noStates,byrow=TRUE)#Phi_SS
G_SS2 = matrix(c(g1,g2),ncol=noInput)
```

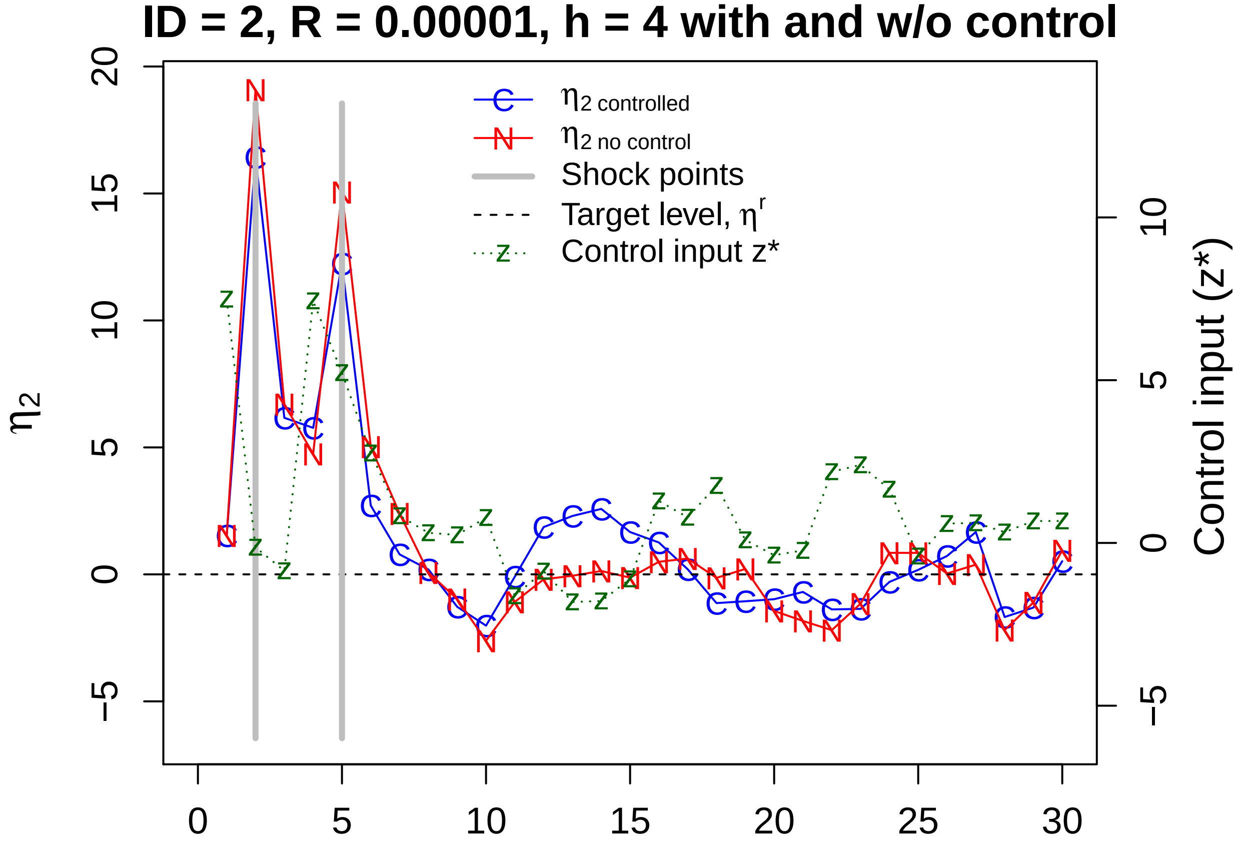
```r
#Now read in the new data for doing control theory implementation
thedat3 = read.table(file0.1,header=FALSE,sep=",")
colnames(thedat3) = c("ID","Time",paste0("V",1:noObs),"z1")
thedat3 <- plyr::ddply(thedat3, .(ID), mutate,
                       zlag1 = dplyr::lag(z1,1,default=0))

Rvalues = c(0.00001, .001,.1,1,10); h = 4 #Try h = 4, 10
#Matrix to hold relative cost and benefit values
costBenefitMatrix = matrix(NA,length(Rvalues),3)

#Place holder to keep track of state deviations
X_dev  = matrix(NA, ncol = 5*length(Rvalues), nrow = noSubj*noTime)
for (rr in 1:length(Rvalues)){
  Rmult = Rvalues[rr]
#  for (h in c(4,10)){
# Define Q and R
# Q =  is a matrix of cost coefficients that expresses the penalty to be imposed on
# state deviations from the desired level. Could set to identity matrix or
# t(lambda)%*%lambda, and play with the values of R.

# R = is a cost matrix for penalizing against the administration of input.
# Usually start with something small, like .1

Q = t(lambda)%*%lambda   #noStates x noStates weight matrix for state deviations
Q_f = t(lambda)%*%lambda #noStates x noStates weight matrix for state deviations
#at the terminal time point of each control horizon
R = Rmult*diag(1,noInput) #Weight matrix of size noInput x noInput representing cost of
#administering control input
eta_target = matrix(0,noTime,noStates)#Same set of target (reference) levels of states
#constrained to be the same across individuals. Could be made person-specific.

data = thedat3 #Observed data set
obsNames = paste0("V",1:noObs); #Observed variable names
idName = "ID"; #Name of participant ID variable in the data set
timeIndexName="Time"; #Name of time index variable in the data set
inputNames = "zlag1" #Name of control input variables
parameterValues = coef(res) #Parameter values from the previously cooked object

controlObject = runController(noStates, noObs, noInput, noSubj, noTime, data,
                         idName,timeIndexName,obsNames, inputNames,
                         dynamics, meas, mdcov,initial,
                         parameterValues, eta_target,Q, Qf, R, h, G_SS2)

#Extract output from controlObject
zControlled = controlObject$optimalInput
filteredStateswithoutControl = controlObject$filteredStateswithoutControl
filteredStateswithControl = controlObject$filteredStateswithControl
newData = controlObject$newData

#Here we are going to regenerate simulated data with the
#same random number seed, but now using zstar.
set.seed(12345)
controlData3 = GenerateData(noTime,npad,noSubj,noStates,noObs,noInput,psi,
                            lambda,theta,Phi_SS,eta_intercept,
                            G_SS2,a0,P0,zControlled,
                            fileObs="Controlled.obs", fileState="Controlled",shocks)
X_Controlled = controlData3$etaall[,3:4]
eta_target2 = do.call("rbind", rep(list(eta_target), noSubj))


#Here, we compute the cost and benefit relative to having the control
#input values be 0 at a particular set of R and h values

CBL.Controlled_KF = CostBenefit(X=t(X_Controlled),
                             u = t(zControlled),
                             Q = Q, R = R, Target = t(eta_target2),
                             Xbase = t(X_Shocks),
                             ubase = matrix(rep(0,noTime*noSubj),1,noTime*noSubj))
costBenefitMatrix[rr,1] = Rmult
costBenefitMatrix[rr,2] = CBL.Controlled_KF$cost
costBenefitMatrix[rr,3] = CBL.Controlled_KF$benefit
#Comparing the discrepancies between eta_target and the
#regenerated true states under zstar vs. no control input

#Original true states with no shocks
X_dev[,1+(rr-1)*5] = sqrt(rowMeans((X_NoShocks - eta_target2)^2))
#True states with shocks
X_dev[,2+(rr-1)*5] = rowMeans((X_Shocks - eta_target2)^2)
#Regenerated true states with zstar
X_dev[,3+(rr-1)*5] = rowMeans((X_Controlled - eta_target2)^2)
#Filtered states with no control input
X_dev[,4+(rr-1)*5] = rowMeans((filteredStateswithoutControl - eta_target2)^2)
#Filtered states with zstar
X_dev[,5+(rr-1)*5] = rowMeans((filteredStateswithControl - eta_target2)^2)

#Generate some plots
np_test = c(1,2,3)
for (id in np_test){
  for (toPlot in 1:noStates){
   # pdf(paste0("ControlChapterh",h,"R",R,"ID",id,"X",toPlot,".pdf"))
    par(cex.axis=1.2,cex.lab=1.3,cex.main=1.4,mar=c(2.1,4.1,2.1,4.1))
    plot(1:noTime,X_Controlled[1:noTime +(id-1)*noTime,toPlot],
         xlim=c(0,noTime),
         ylim = c(min(X_Shocks[,toPlot],na.rm=noTime)-2,
                  max(X_Shocks[,toPlot],na.rm=noTime)+.1),
         ylab=bquote(eta[.(toPlot)]),
         xlab="Time",
         main=paste0("ID = ",id,', R = ',R,", h = ",h, ' with and w/o control'),
         type="n")
    lines(1:noTime,X_Controlled[1:noTime +(id-1)*noTime,toPlot],col="blue")
    points(1:noTime,X_Controlled[1:noTime +(id-1)*noTime,toPlot],col="blue",pch="C")
    lines(1:noTime,X_Shocks[1:noTime +(id-1)*noTime,toPlot],col="red")
    points(1:noTime,X_Shocks[1:noTime +(id-1)*noTime,toPlot],col="red",pch="N")
    abline(h=eta_target[id,toPlot], col = 'black', lty =2 )
    seqq = seq(min(X_Shocks[,toPlot]-2,na.rm=noTime),
               max(X_Shocks[,toPlot],na.rm=noTime),1)
    lines(rep(shockPoints[id,1],length(seqq)),seqq,col="gray",lwd=3)
    lines(rep(shockPoints[id,2],length(seqq)),seqq,col="gray",lwd=3)
    legend("top", legend=c(as.expression(bquote(eta[.(toPlot)~"controlled"])),
                            as.expression(bquote(eta[.(toPlot)~"no control"])),
                            "Shock points",
                            as.expression(bquote("Target level,"~eta^r)),
                            "Control input z*"),
           col=c("blue", "red","gray","black","darkgreen"), lty=c(1,1,1,2,3),
           cex=1,pch=c("C","N",NA,NA,"z*"),bty="n",lwd=c(1,1,3,1,1))
    par(new=TRUE,mar=c(2.1,4.1,2.1,4.1))
    plot(1:noTime,newData$zlag1[newData$ID==id],type="n",
         xlab=NA,ylab=NA,xlim=c(0,noTime),
         ylim=c(-6, 14),axes=FALSE)
    lines(1:noTime,newData$zlag1[newData$ID==id],col="darkgreen",lty=3)
    points(1:noTime,newData$zlag1[newData$ID==id],pch="z",col="darkgreen")
    axis(side = 4)
    mtext(text='Control input (z*)',line=2.5,
          side = 4, at = 4.5,
          outer=FALSE,cex=1.3)
  #  dev.off()
  }}

}
```

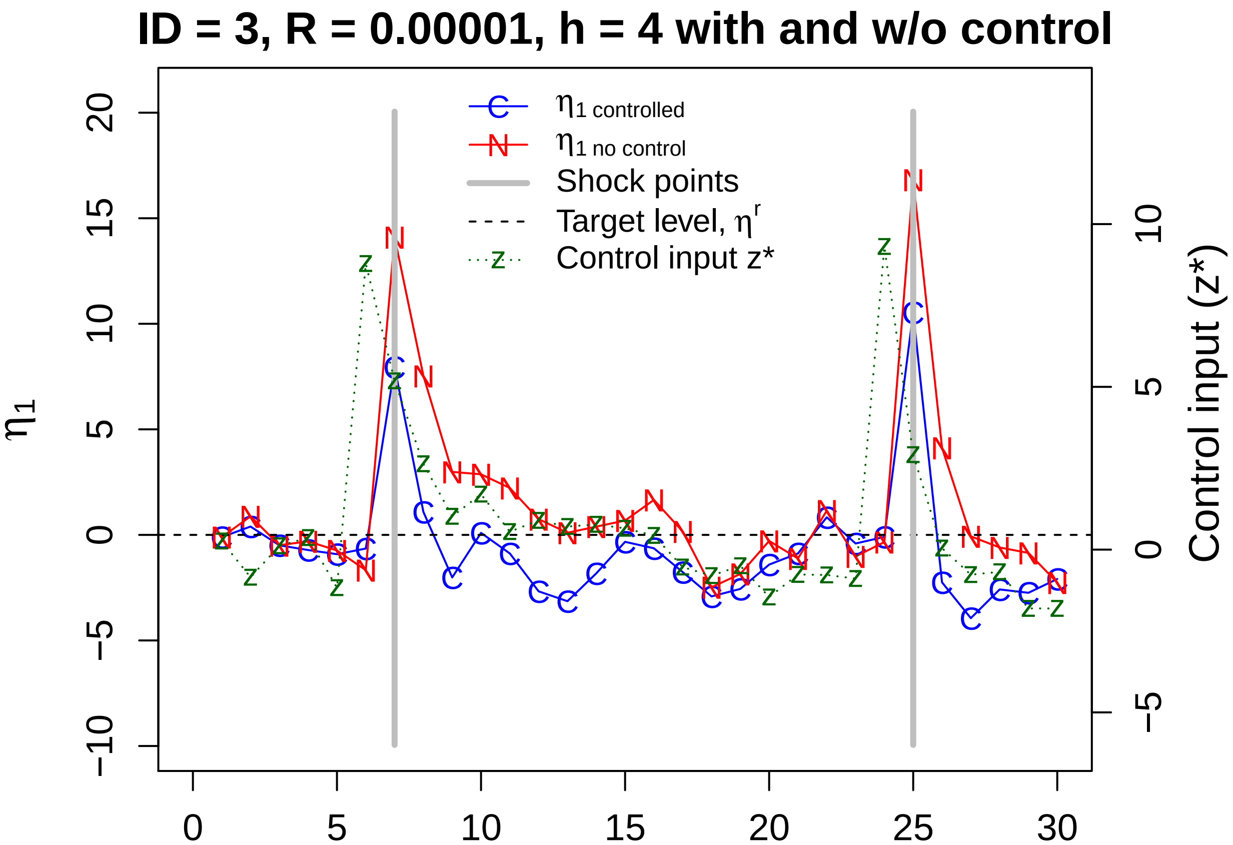


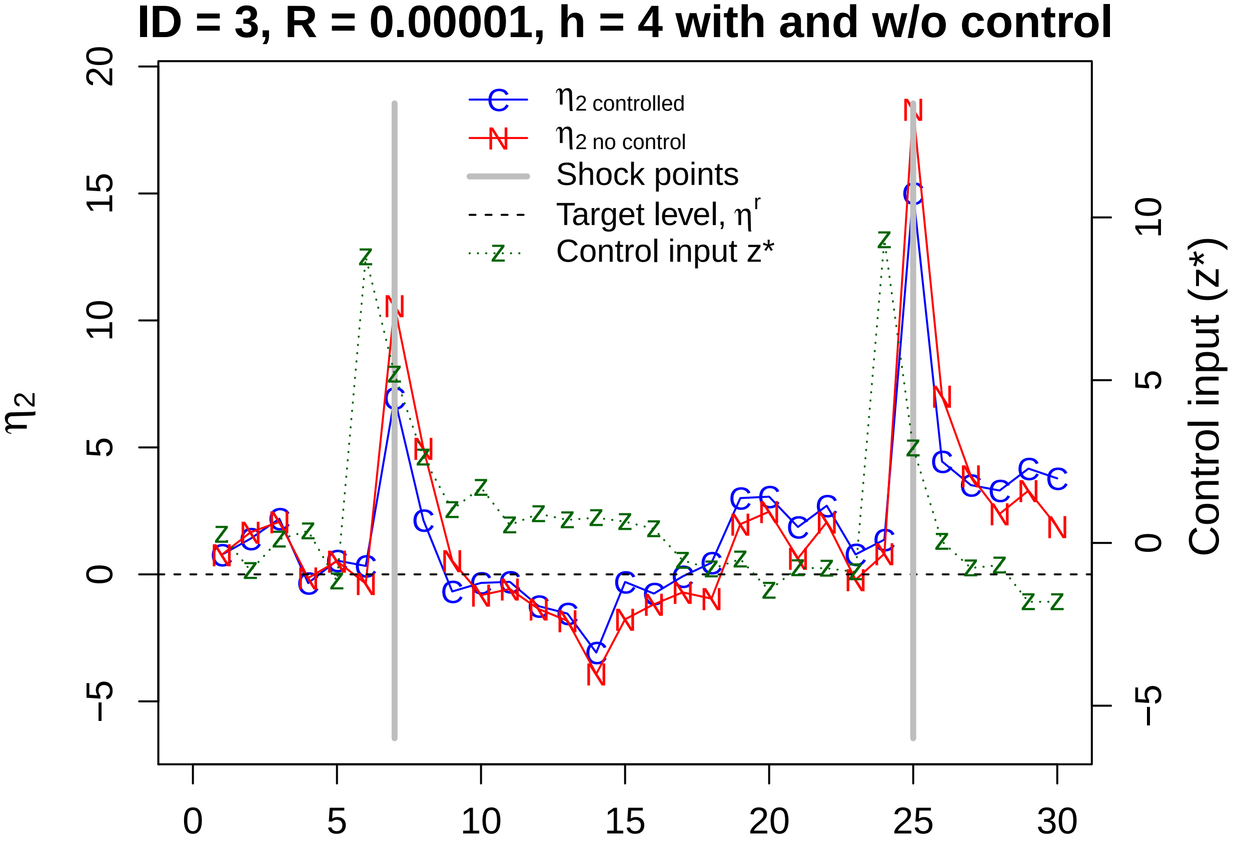

ID = 1, R = 0.00001, h = 4 with and w/o control
η_1 controlled
η_1 no control
Shock points
Target level, η^r
Control input z*
η_1
Control input (z*)
20
15
10
5
0
−5
−10
0
5
10
15
20
25
30

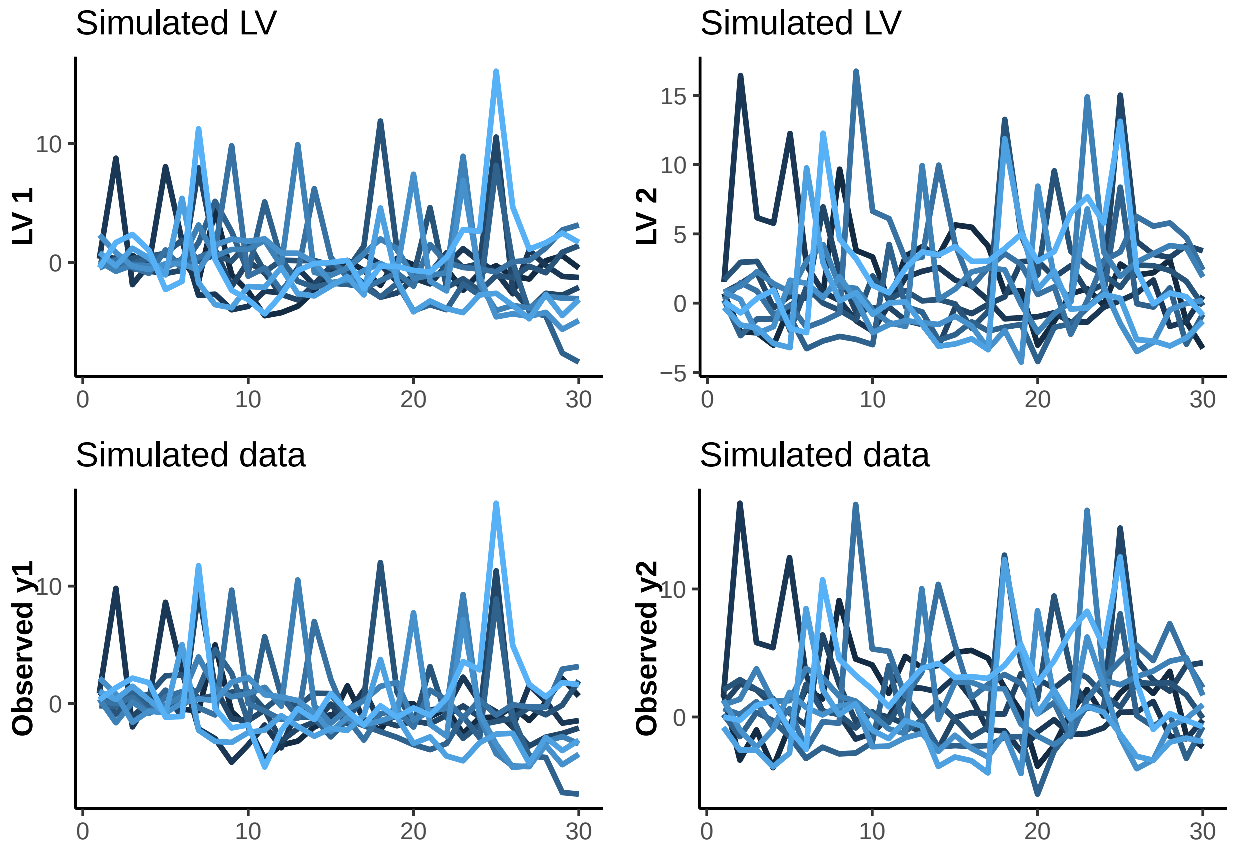

ID = 1, R = 0.00001, h = 4 with and w/o control
η_2 controlled
η_2 no control
Shock points
Target level, η^r
Control input z*
η_2
Control input (z*)
20
15
10
5
0
−5
0
5
10
15
20
25
30

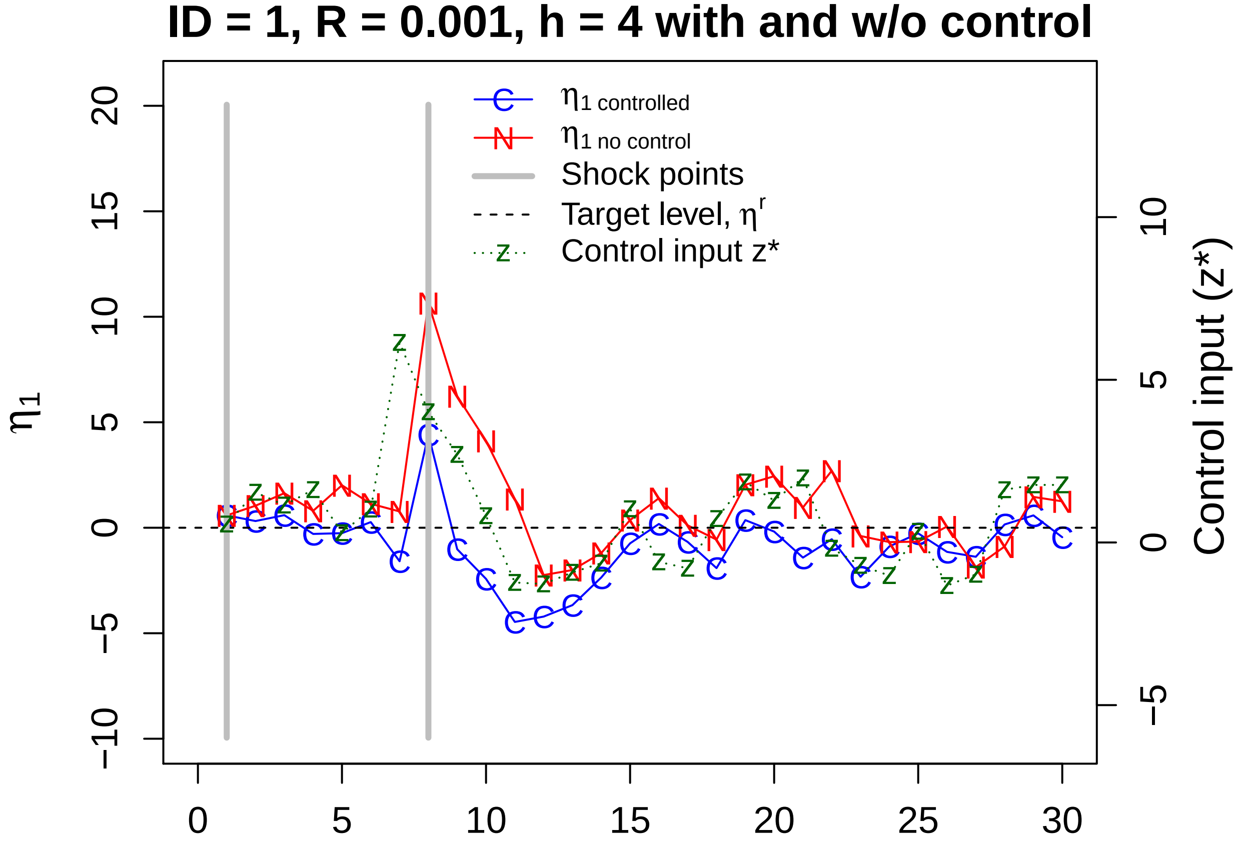

ID = 2, R = 0.00001, h = 4 with and w/o control
η1 controlled
η1 no control
Shock points
Target level, η^r
Control input z*
η1
Control input (z*)
−10
−5
0
5
10
15
20
0
5
10
15
20
25
30
−5
0
5
10

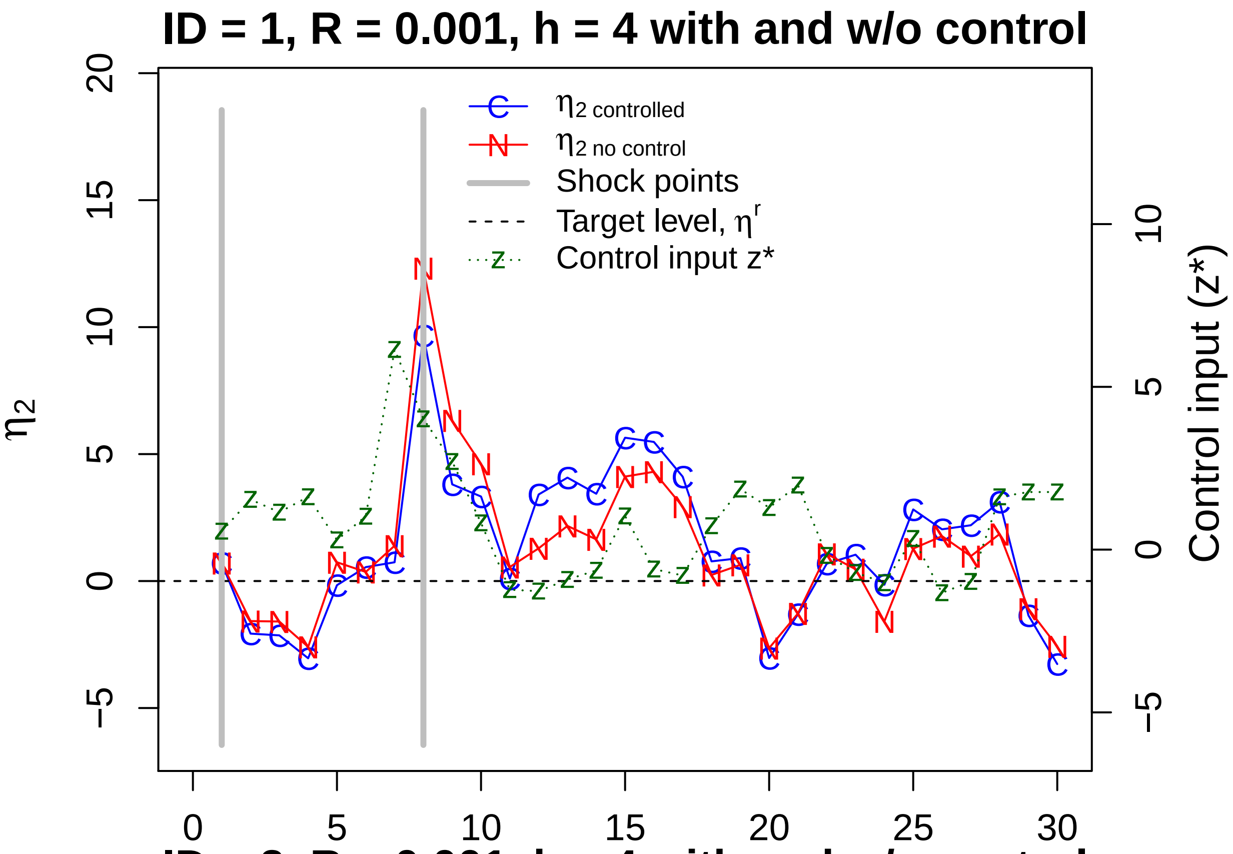

ID = 2, R = 0.00001, h = 4 with and w/o control
η2 controlled
η2 no control
Shock points
Target level, η^r
Control input z*
η2
Control input (z*)
−5
0
5
10
15
20
0
5
10
15
20
25
30
−5
0
5
10

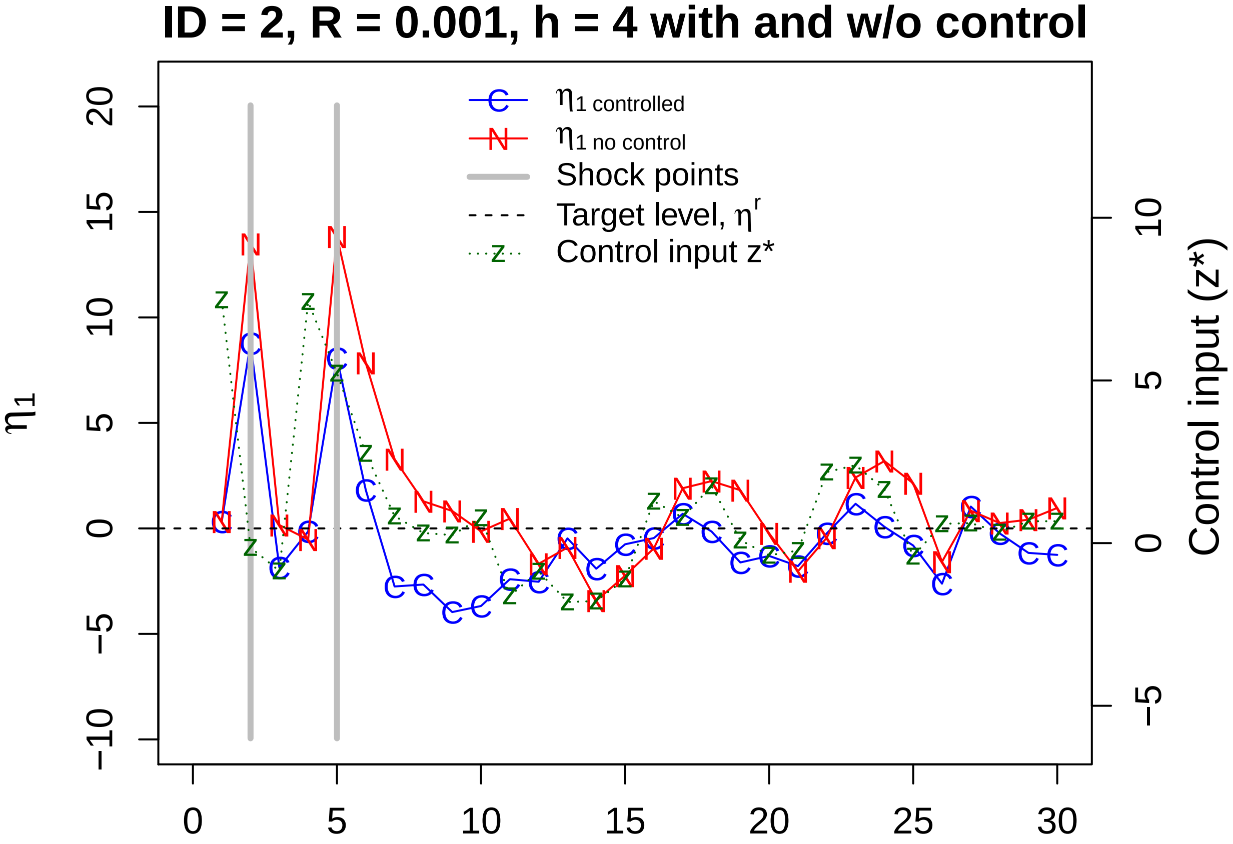

ID = 3, R = 0.00001, h = 4 with and w/o control


ID = 3, R = 0.00001, h = 4 with and w/o control

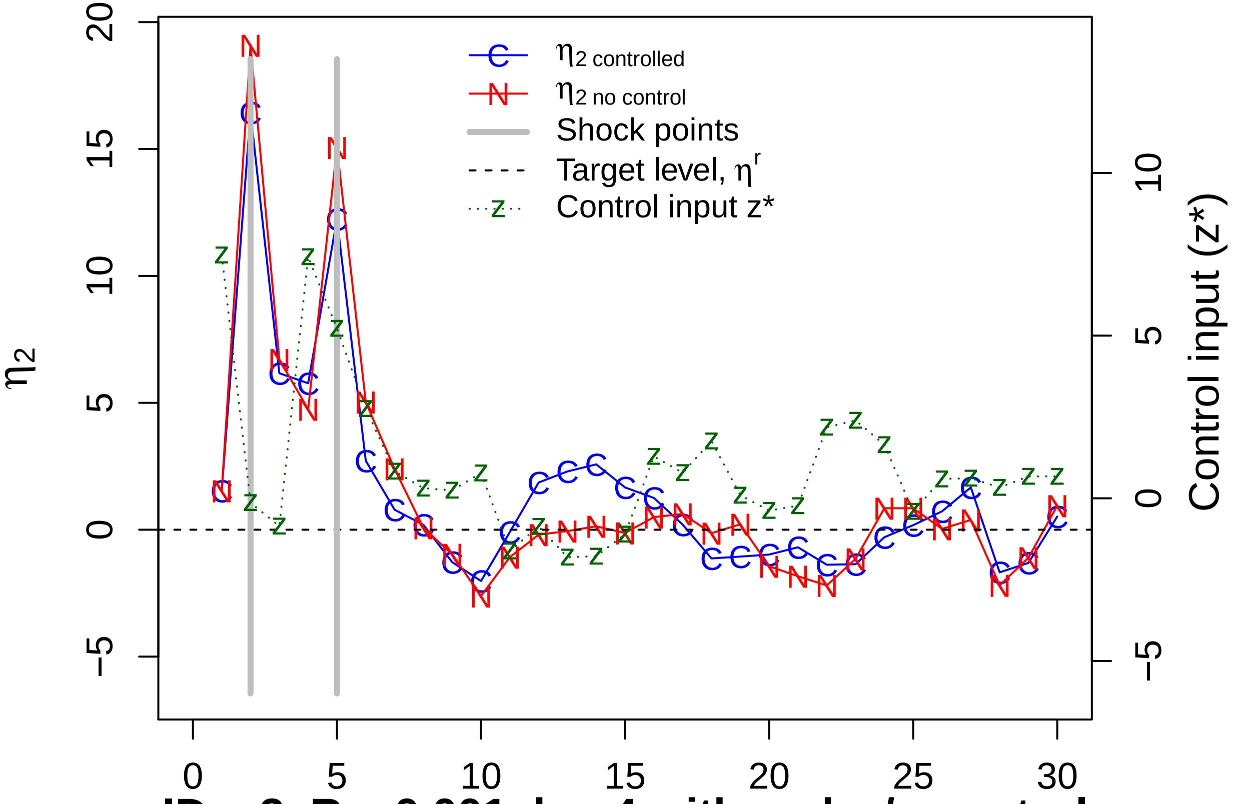

Simulated LV
LV 1
10
0
0
10
20
30
Simulated LV
LV 2
15
10
5
0
−5
0
10
20
30
Simulated data
Observed y1
10
0
0
10
20
30
Simulated data
Observed y2
10
0
0
10
20
30


ID = 1, R = 0.001, h = 4 with and w/o control
η1 controlled
η1 no control
Shock points
Target level, η^r
Control input z*
η1
20
15
10
5
0
−5
−10
Control input (z*)
10
5
0
−5
0
5
10
15
20
25
30

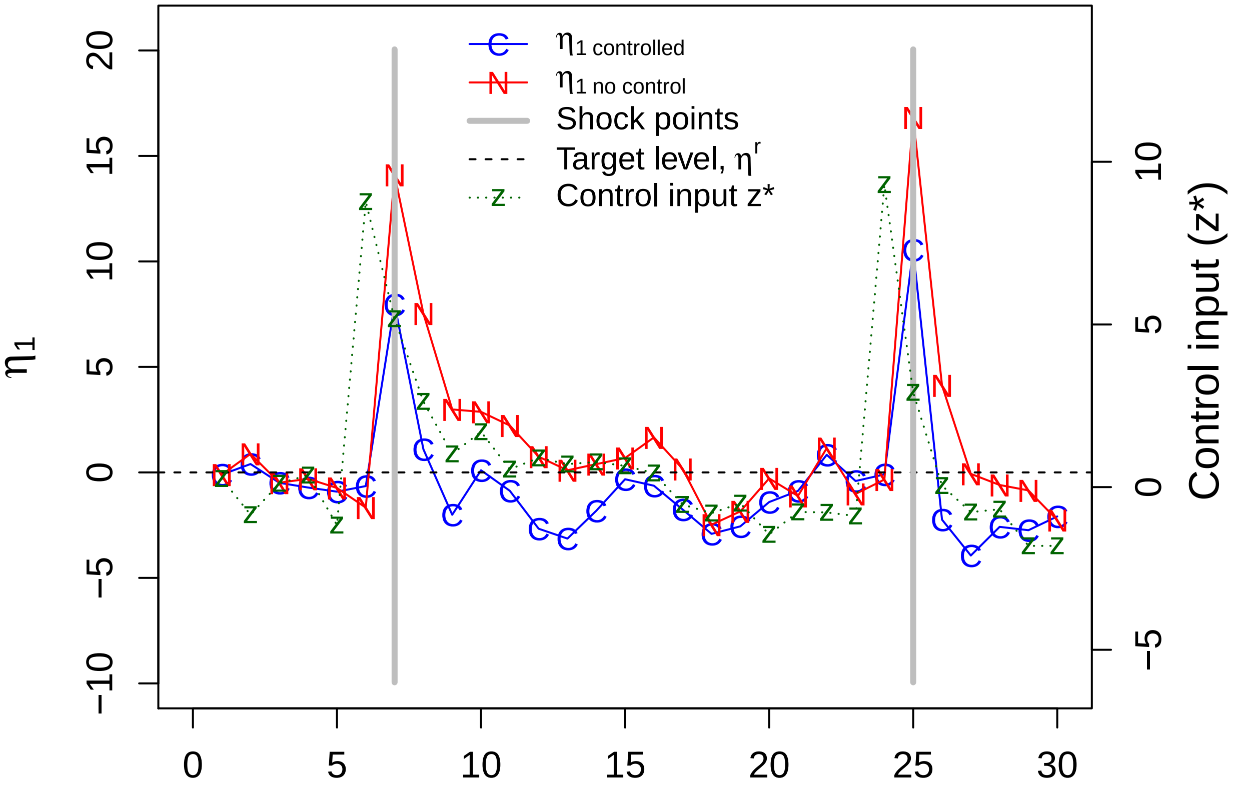

ID = 1, R = 0.001, h = 4 with and w/o control
η2 controlled
η2 no control
Shock points
Target level, ηr
Control input z*
η2
Control input (z*)
20
15
10
5
0
−5
0
5
10
15
20
25
30
10
5
0
−5

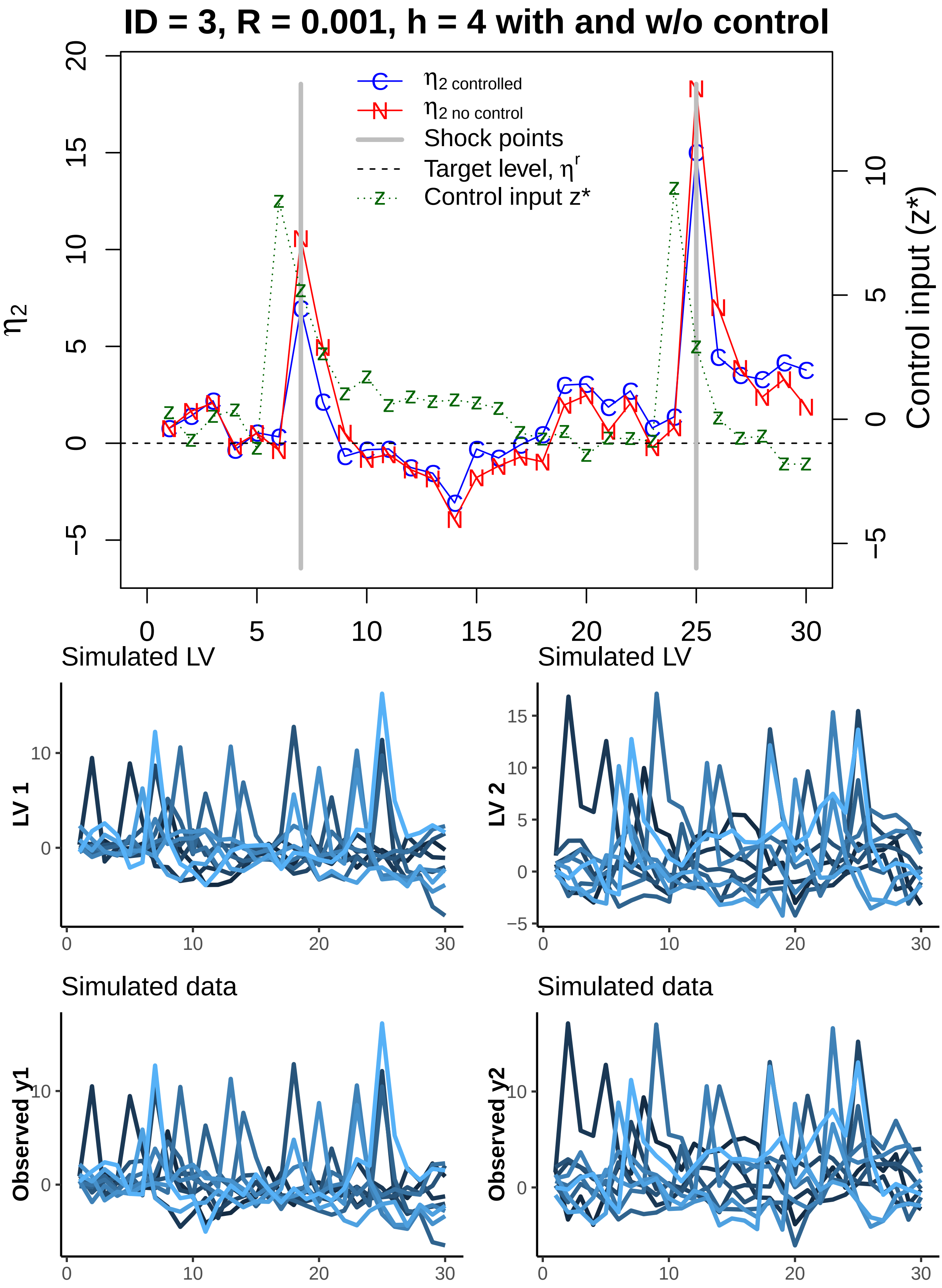

ID = 2, R = 0.001, h = 4 with and w/o control
η1 controlled
η1 no control
Shock points
Target level, ηr
Control input z*
η1
Control input (z*)
20
15
10
5
0
−5
−10
0
5
10
15
20
25
30
10
5
0
−5

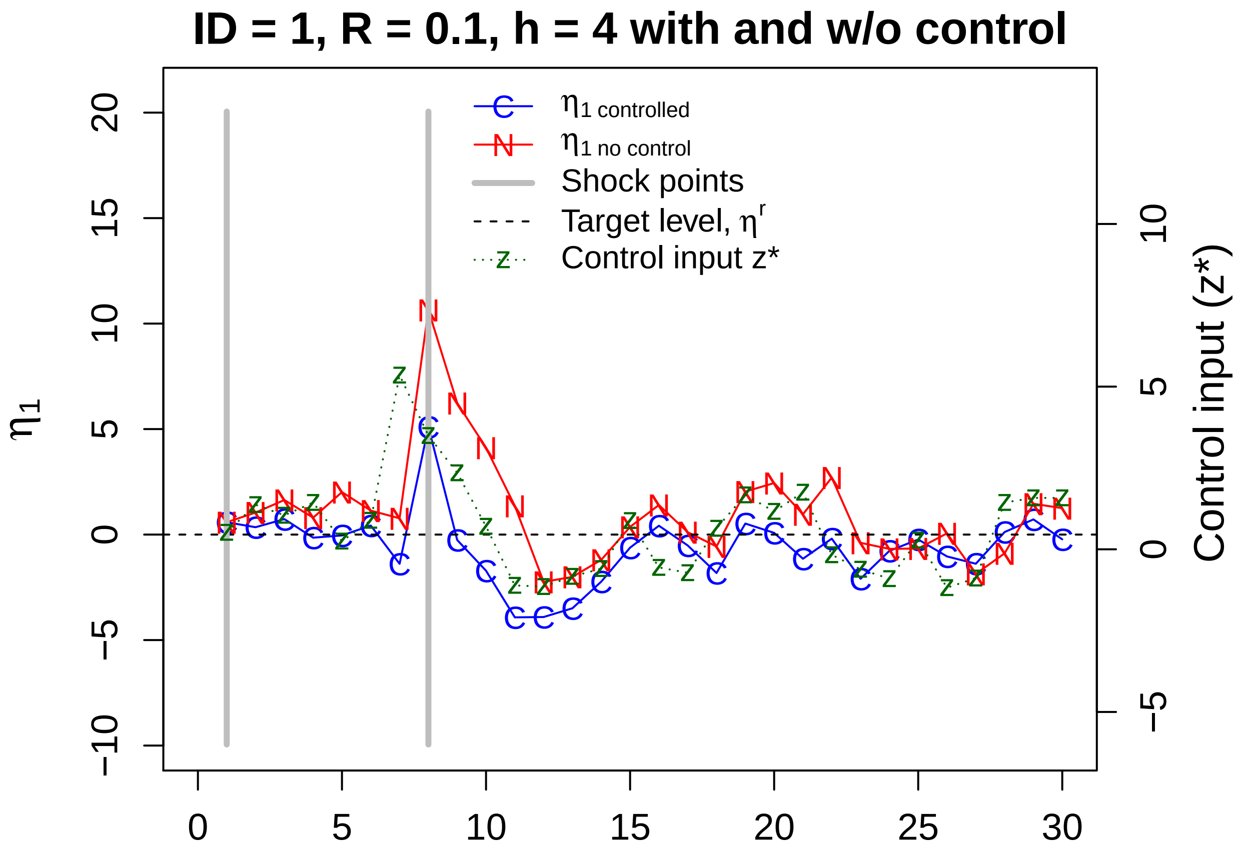
ID = 2, R = 0.001, h = 4 with and w/o control

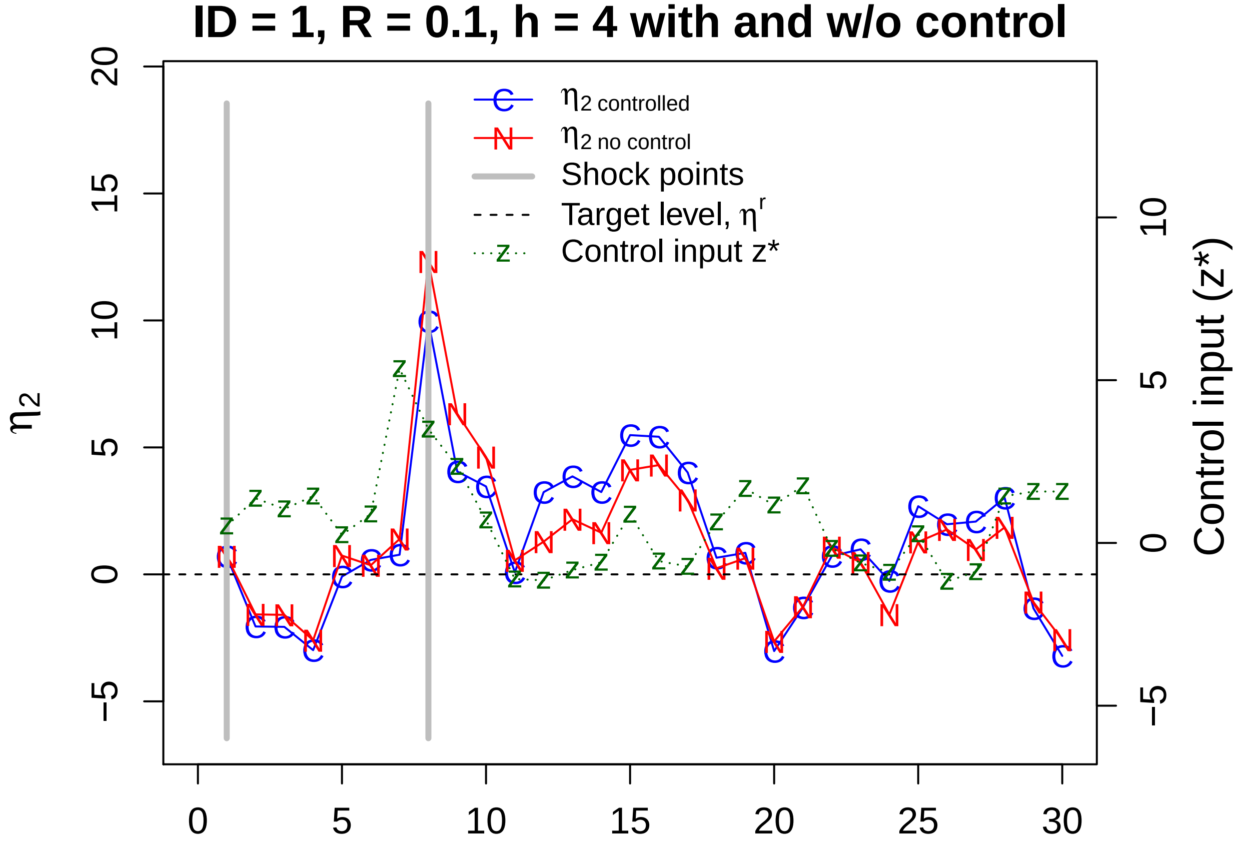


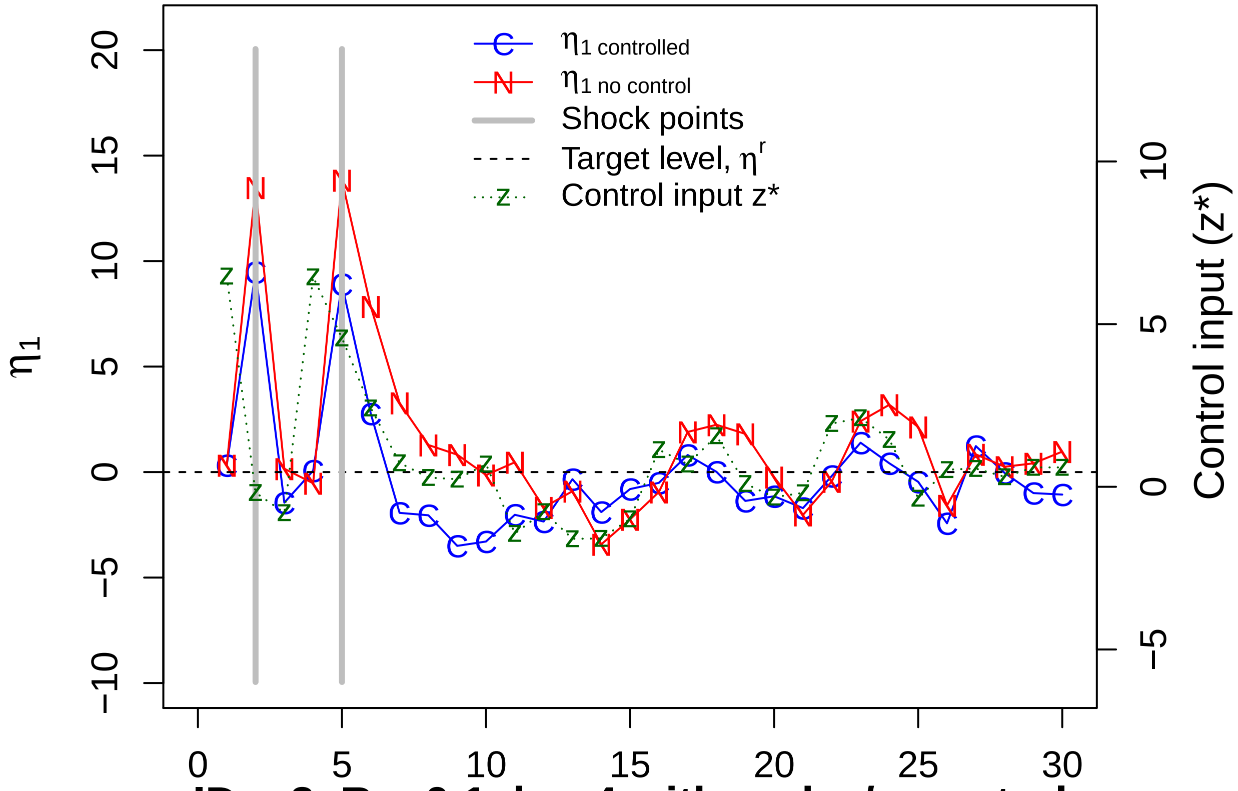
ID = 3, R = 0.001, h = 4 with and w/o control



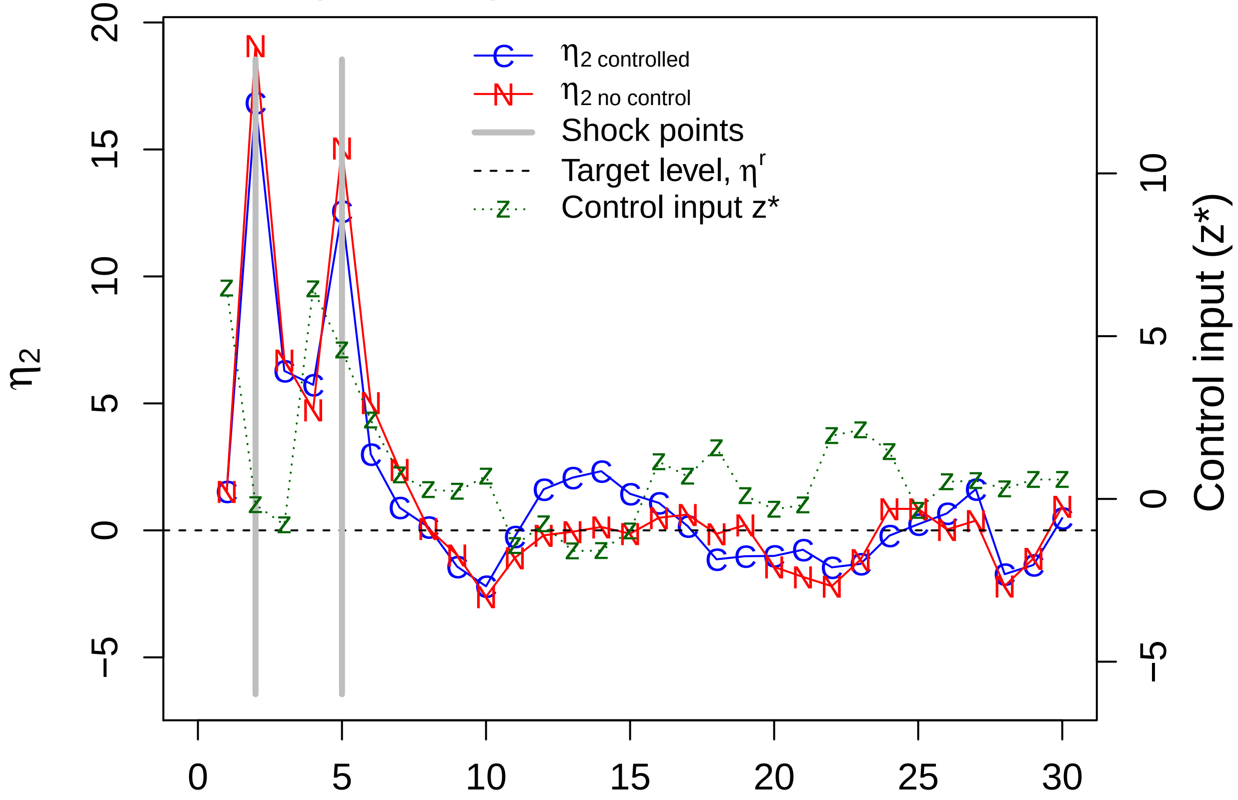

ID = 3, R = 0.001, h = 4 with and w/o control
η_2
Control input (z^*)
η_2 controlled
η_2 no control
Shock points
Target level, η^r
Control input z*
C
N
z
−5
0
5
10
15
20
0
5
10
15
20
25
30
Simulated LV
LV 1
LV 2
Simulated data
Observed y1
Observed y2


ID = 1, R = 0.1, h = 4 with and w/o control
η_1 controlled
η_1 no control
Shock points
Target level, η^r
Control input z*
η_1
Control input (z*)
−10
−5
0
5
10
15
20
−5
0
5
10
0
5
10
15
20
25
30

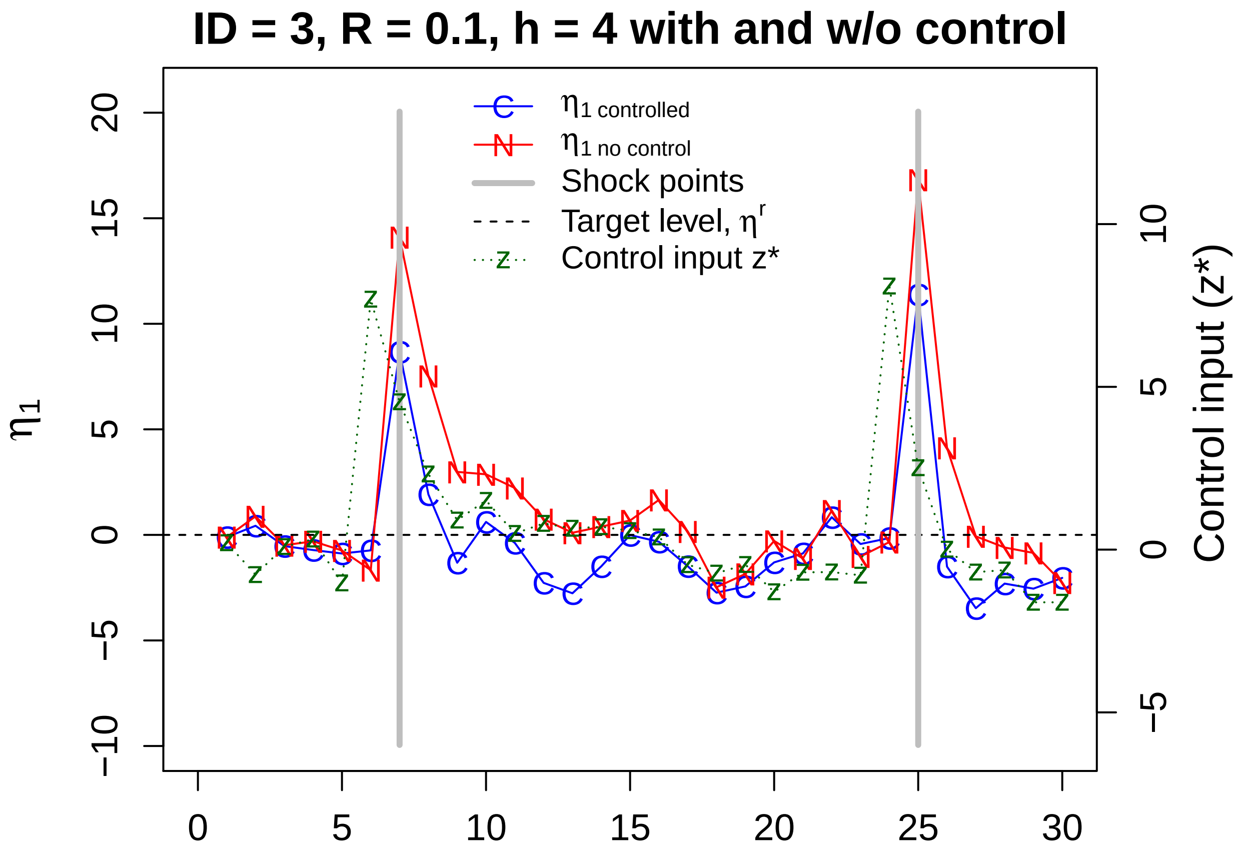

ID = 1, R = 0.1, h = 4 with and w/o control
η_2 controlled
η_2 no control
Shock points
Target level, η^r
Control input z*
η_2
Control input (z*)
−5
0
5
10
15
20
−5
0
5
10
0
5
10
15
20
25
30

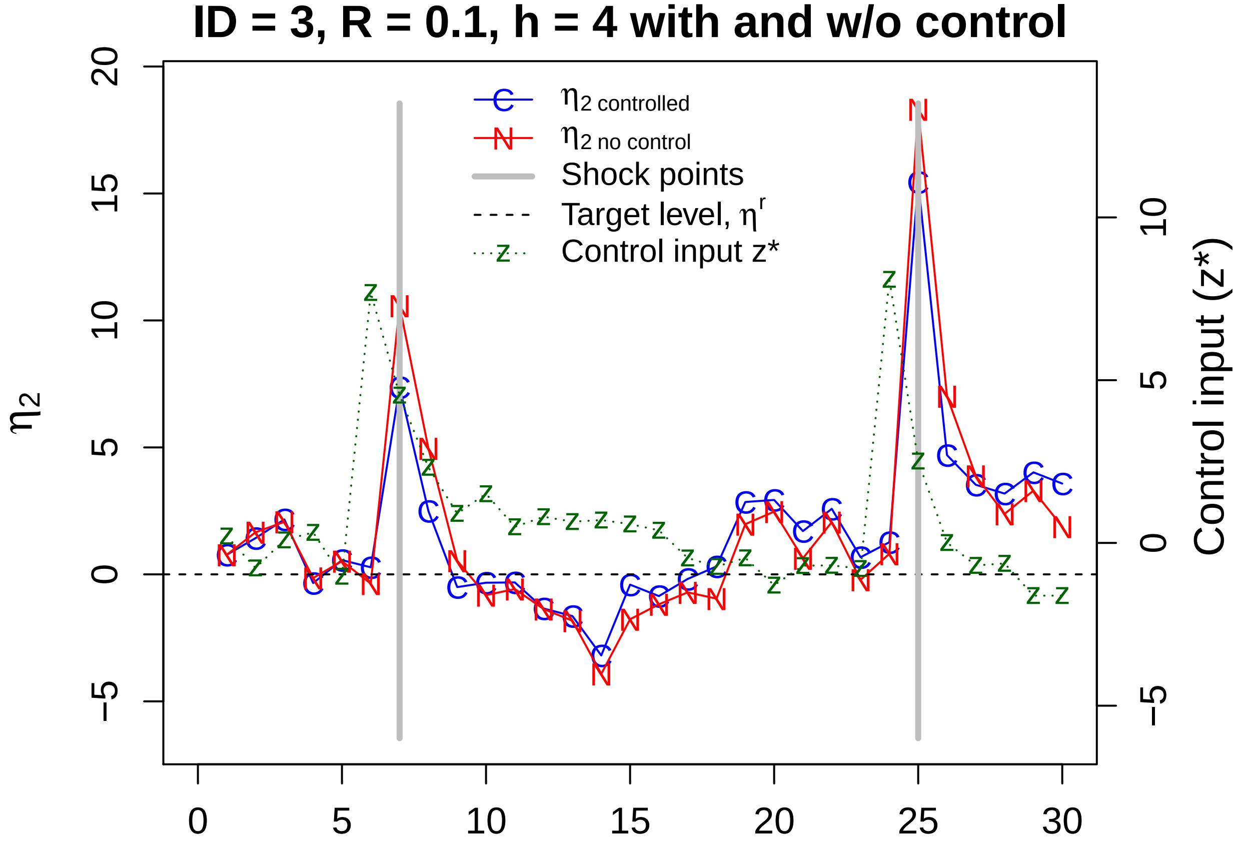


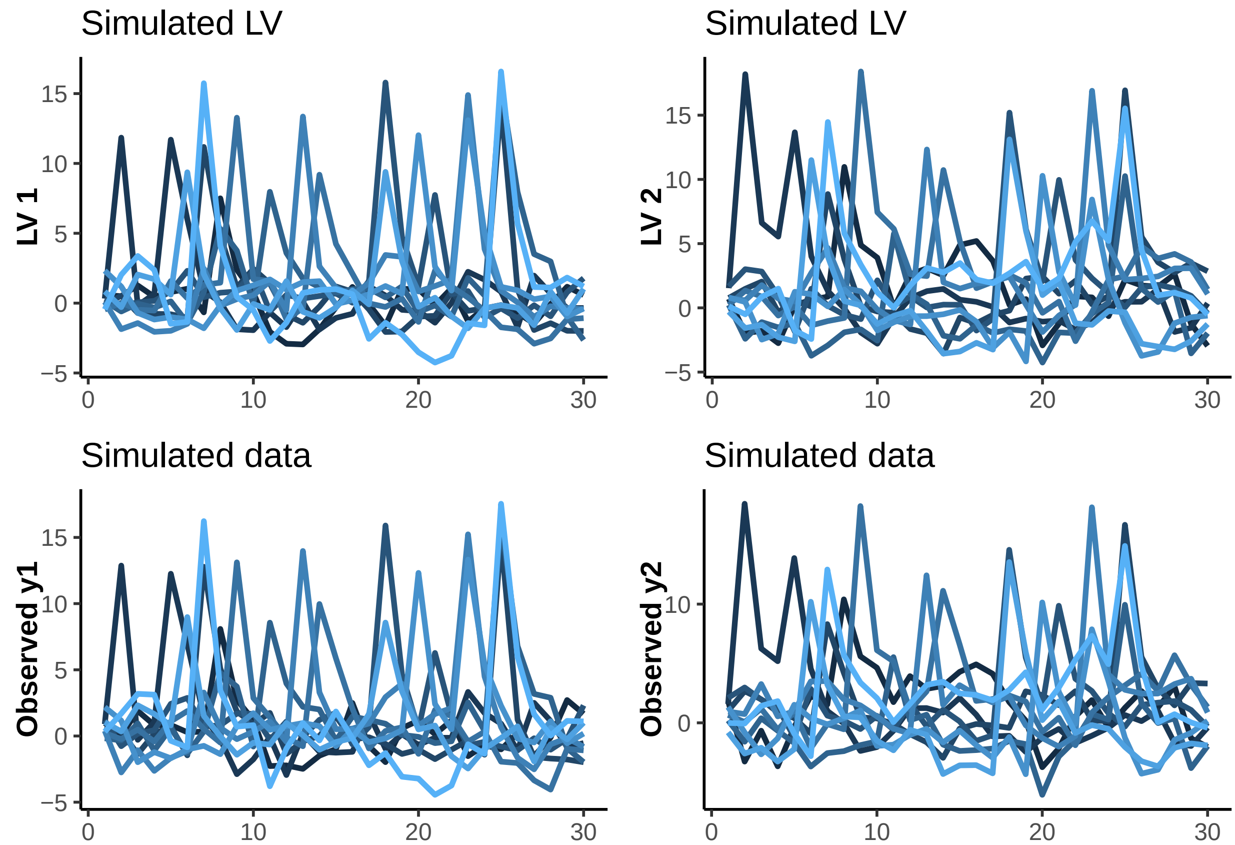
ID = 2, R = 0.1, h = 4 with and w/o control

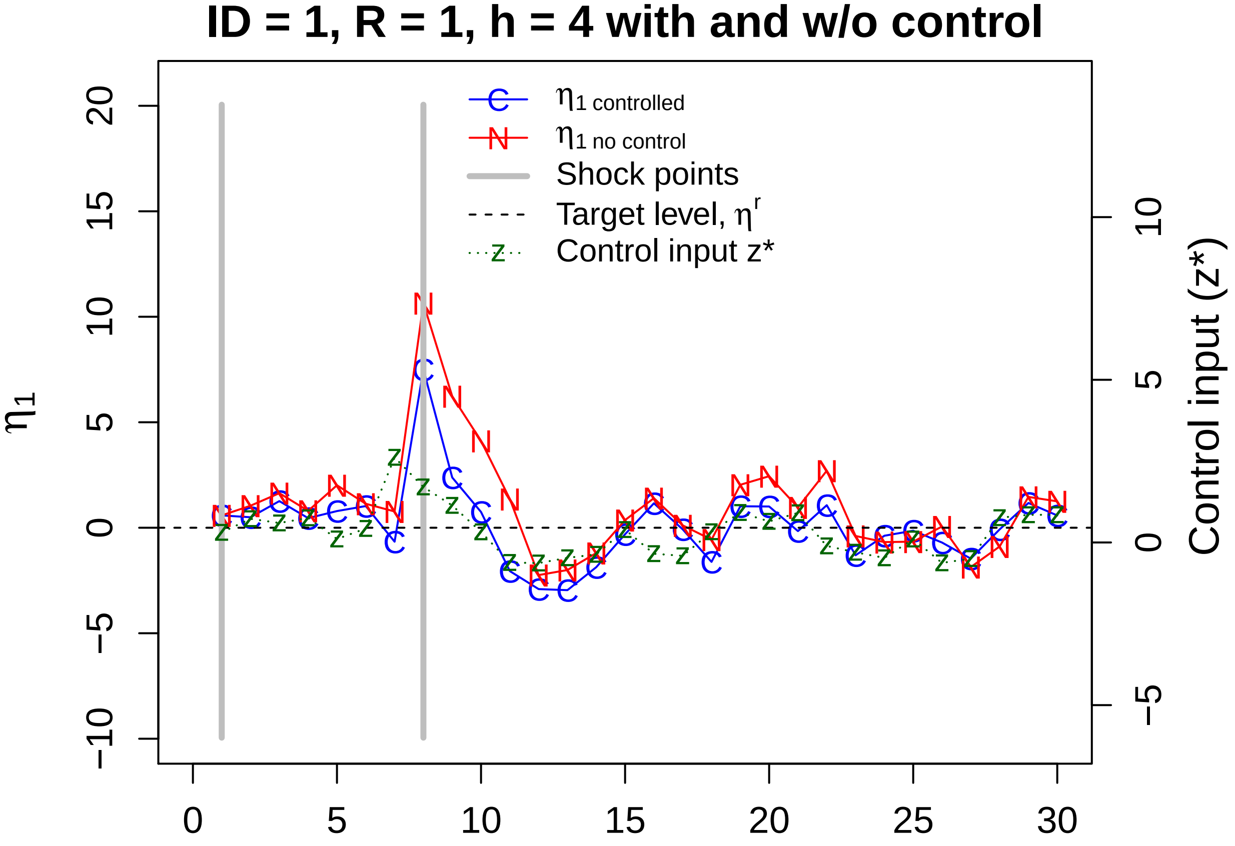


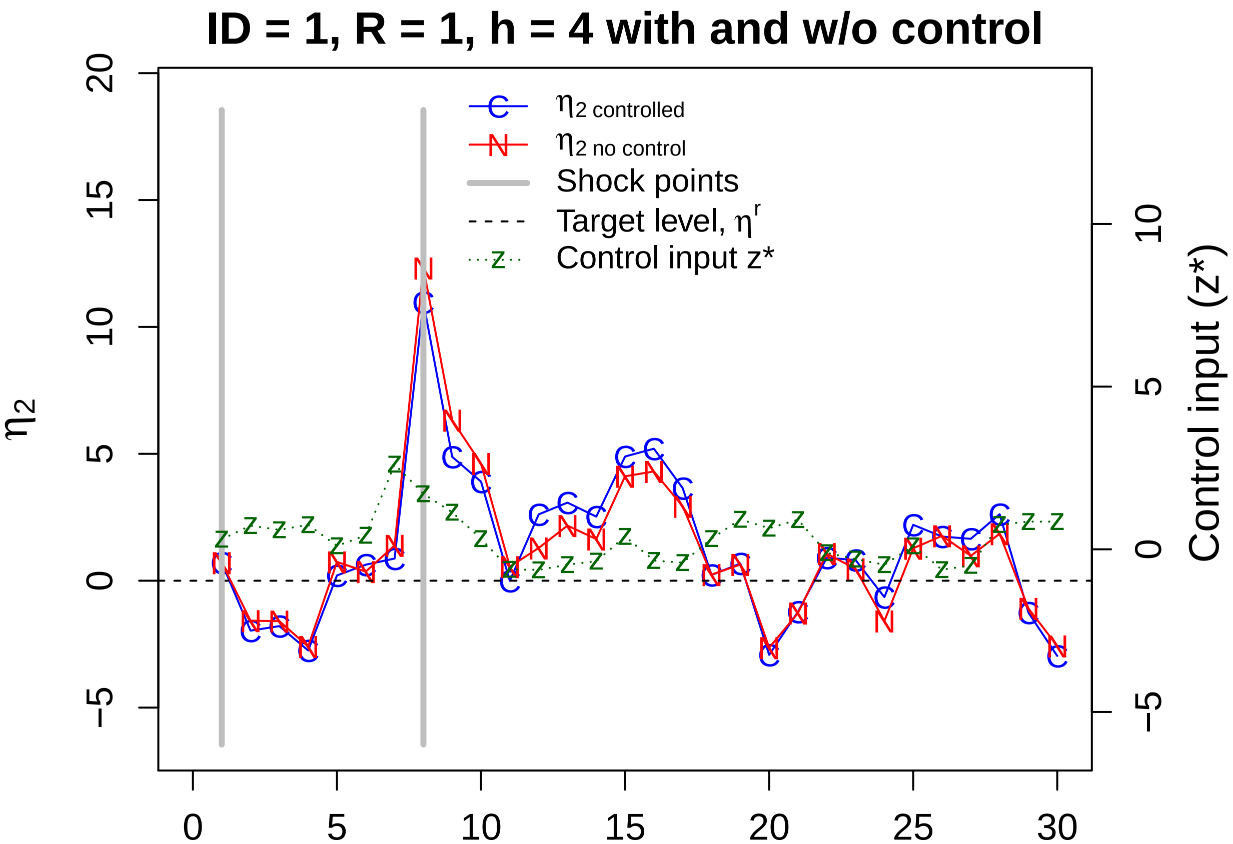
ID = 2, R = 0.1, h = 4 with and w/o control

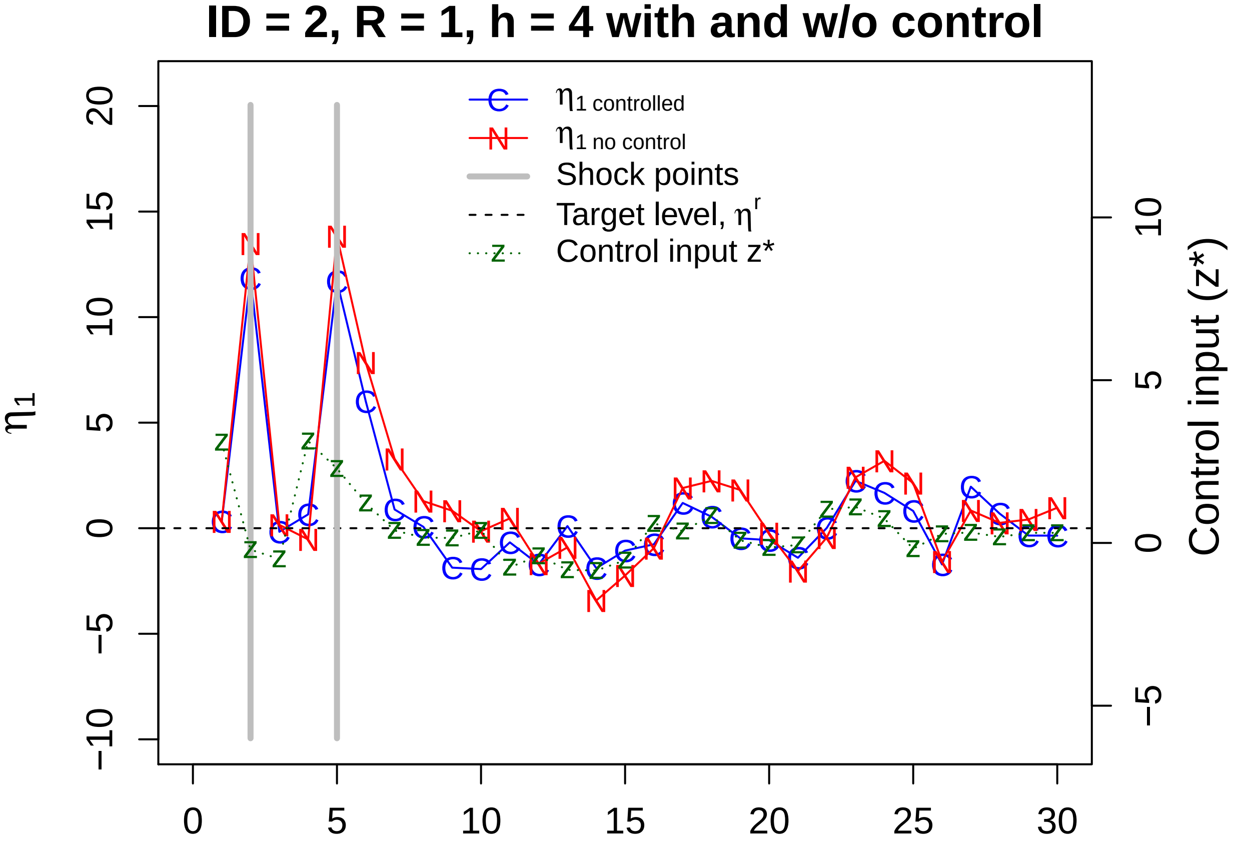

ID = 3, R = 0.1, h = 4 with and w/o control
η1 controlled
η1 no control
Shock points
Target level, η^r
Control input z*
η1
Control input (z*)
−10
−5
0
5
10
15
20
−5
0
5
10
0
5
10
15
20
25
30

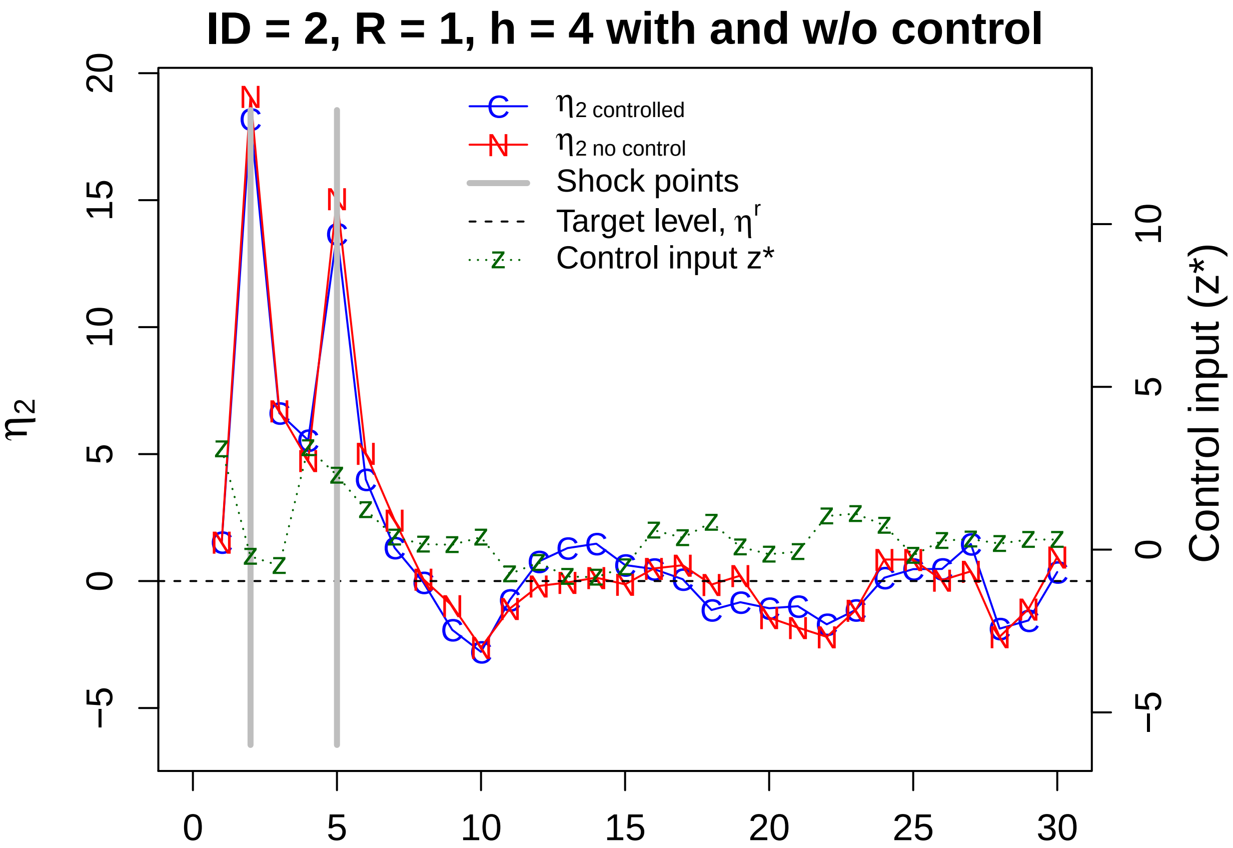

ID = 3, R = 0.1, h = 4 with and w/o control
η2 controlled
η2 no control
Shock points
Target level, η^r
Control input z*
η2
Control input (z*)
−5
0
5
10
15
20
−5
0
5
10
0
5
10
15
20
25
30

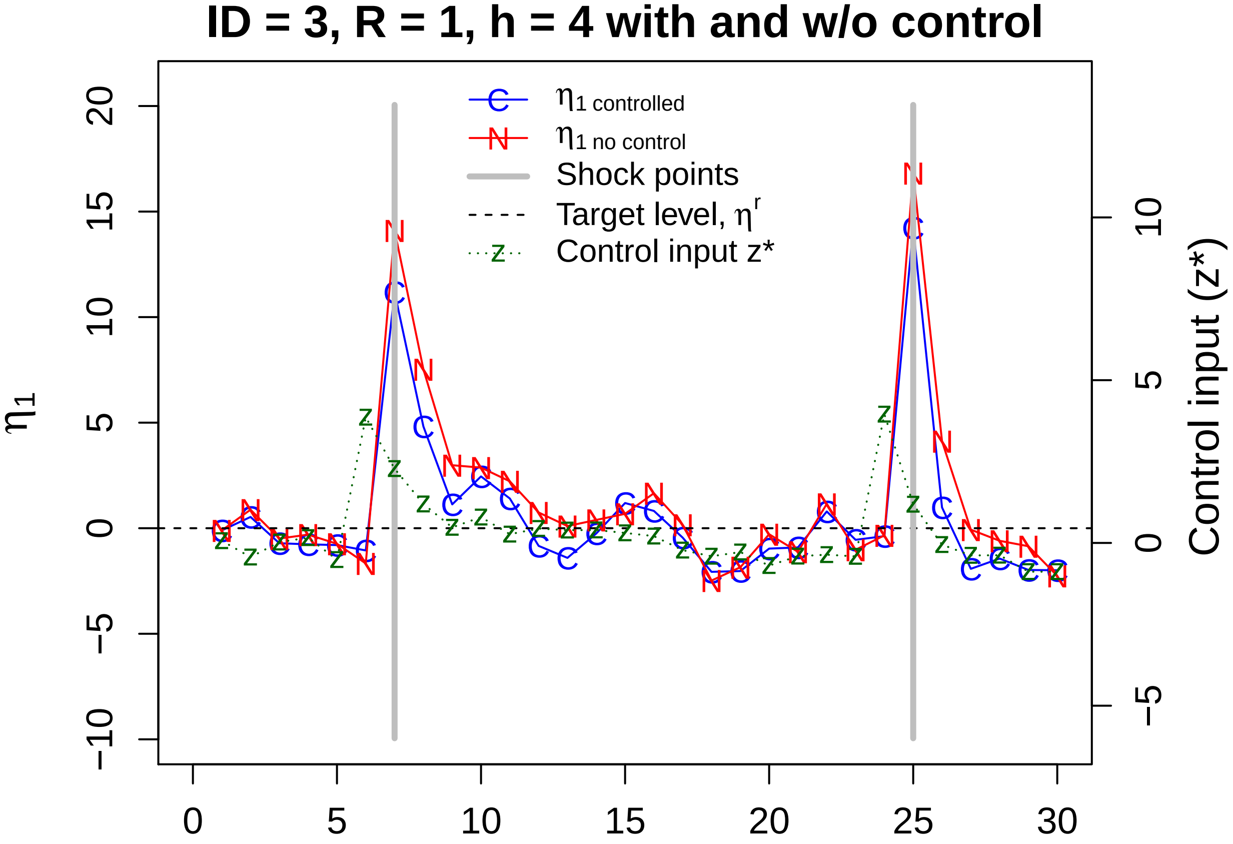

Simulated LV
LV 1
15
10
5
0
−5
0
10
20
30
Simulated LV
LV 2
15
10
5
0
−5
0
10
20
30
Simulated data
Observed y1
15
10
5
0
−5
0
10
20
30
Simulated data
Observed y2
10
0
0
10
20
30

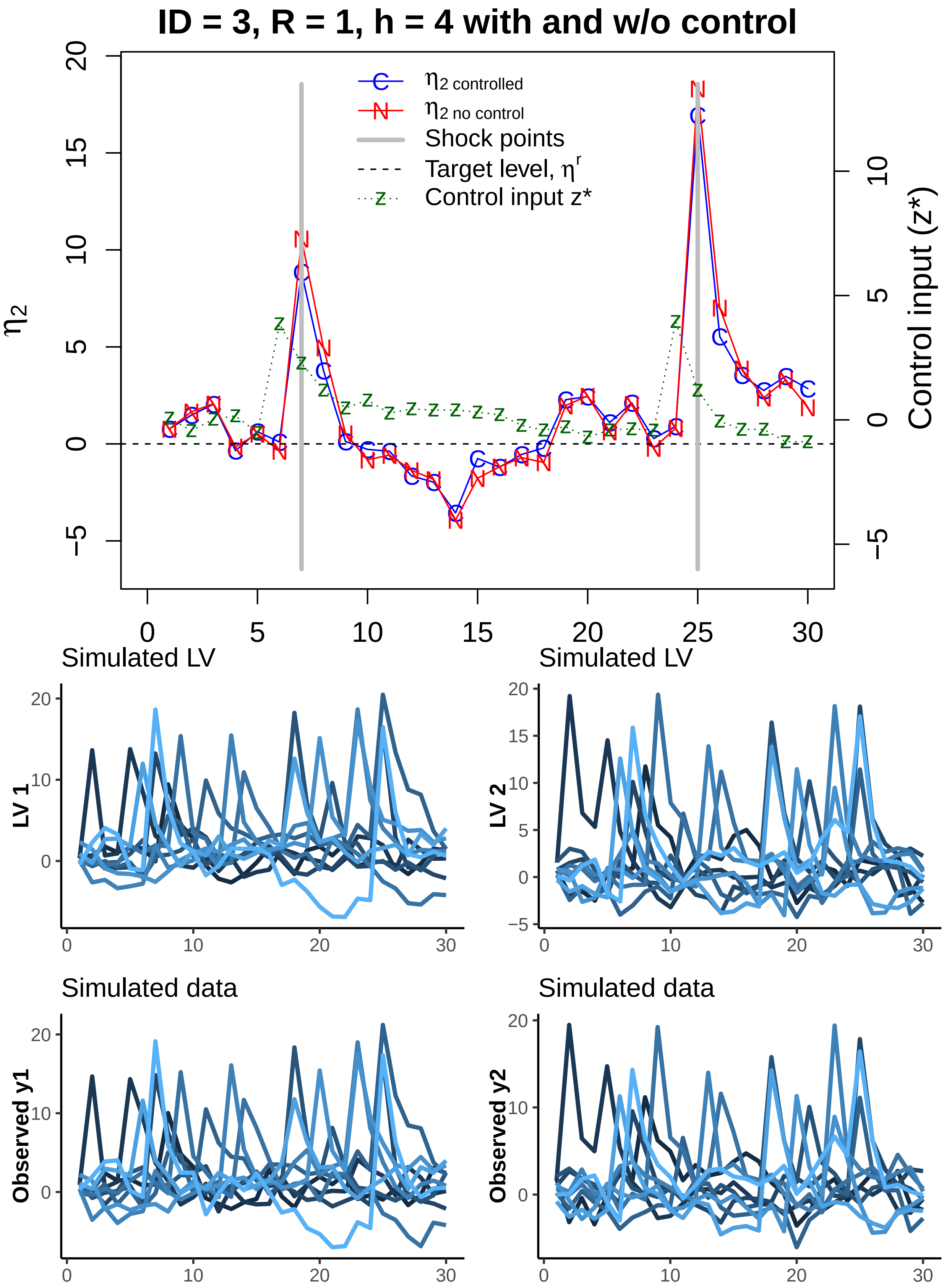

ID = 1, R = 1, h = 4 with and w/o control
η_1 controlled
η_1 no control
Shock points
Target level, η^r
Control input z*
η_1
20
15
10
5
0
−5
−10
Control input (z*)
10
5
0
−5
0
5
10
15
20
25
30

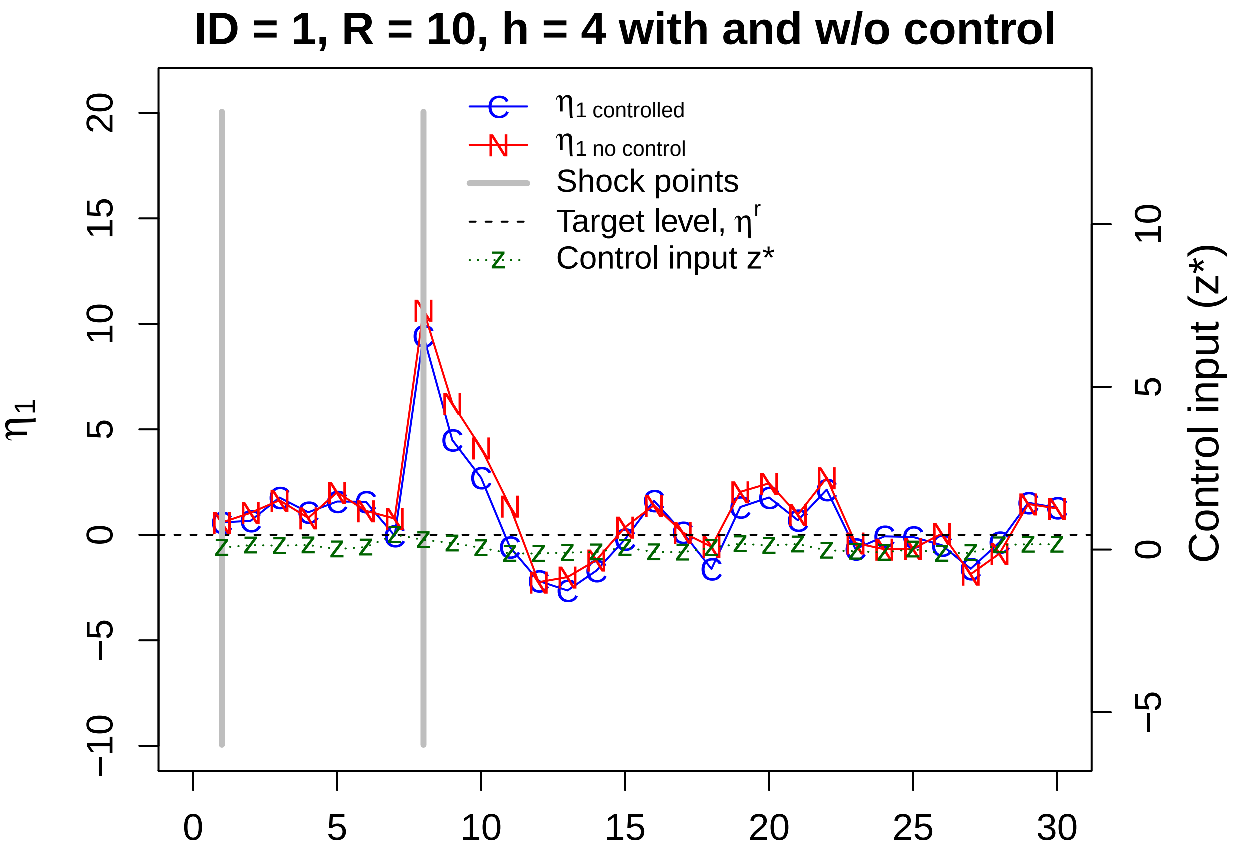

ID = 1, R = 1, h = 4 with and w/o control
η_2 controlled
η_2 no control
Shock points
Target level, η^r
Control input z*
η_2
Control input (z*)
20
15
10
5
0
−5
10
5
0
−5
0
5
10
15
20
25
30

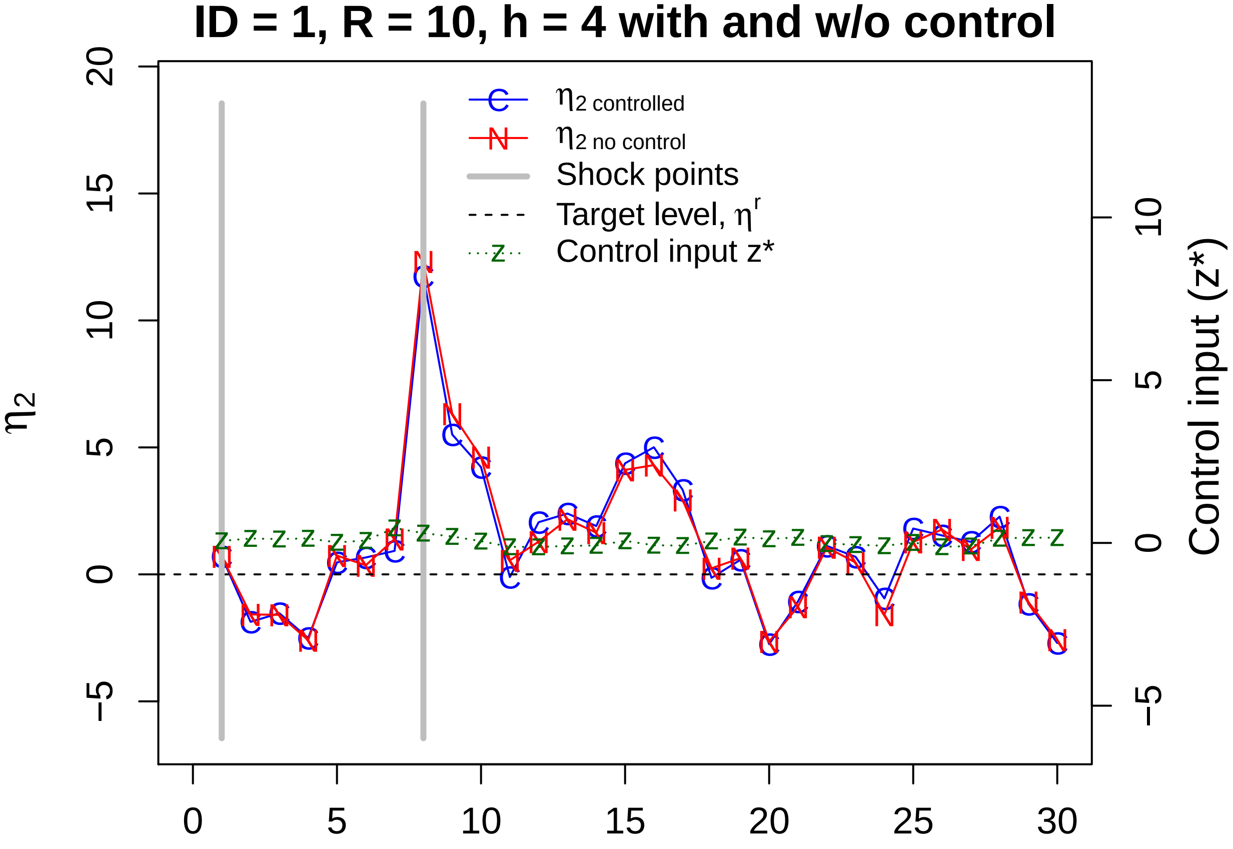

ID = 2, R = 1, h = 4 with and w/o control
η_1 controlled
η_1 no control
Shock points
Target level, η^r
Control input z*
η_1
Control input (z*)
20
15
10
5
0
−5
−10
10
5
0
−5
0
5
10
15
20
25
30

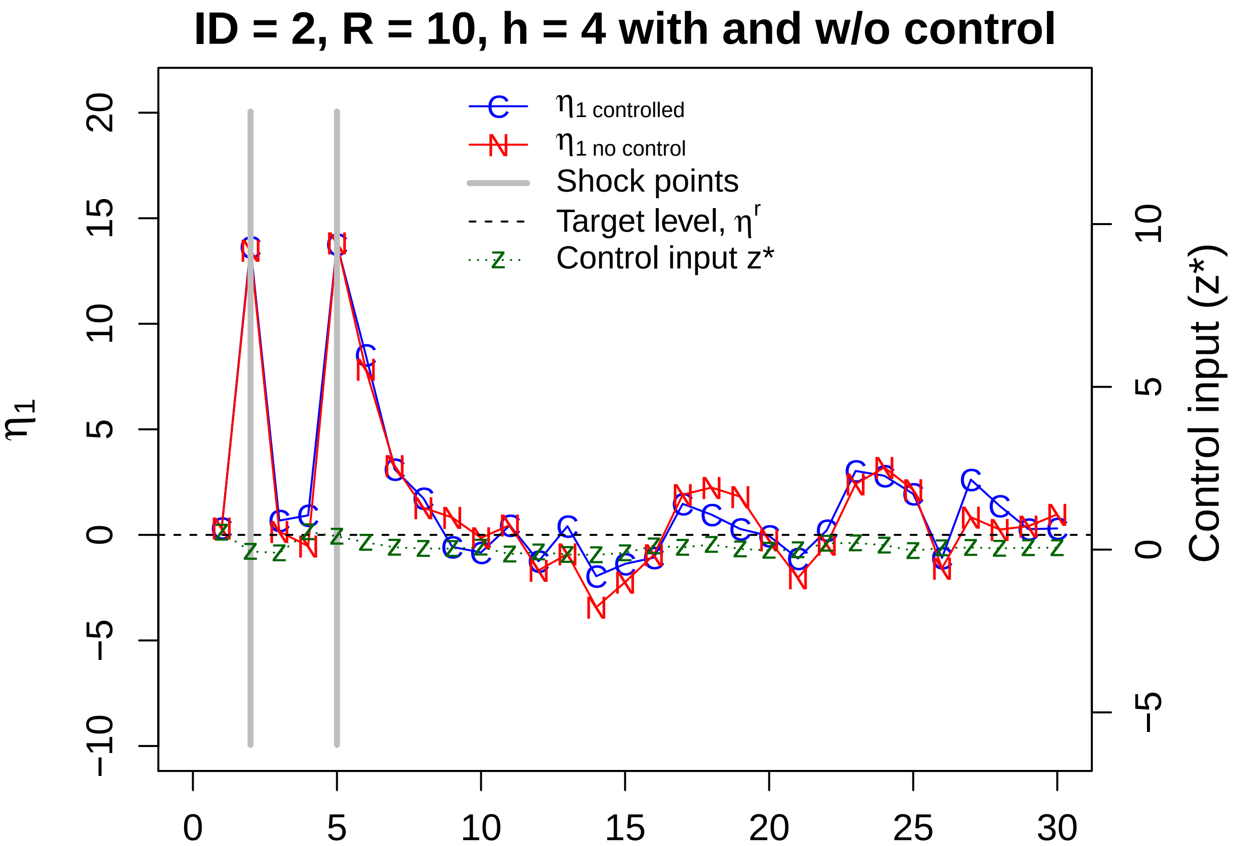

ID = 2, R = 1, h = 4 with and w/o control
η_2 controlled
η_2 no control
Shock points
Target level, η^r
Control input z*
η_2
Control input (z*)
20
15
10
5
0
−5
10
5
0
−5
0
5
10
15
20
25
30

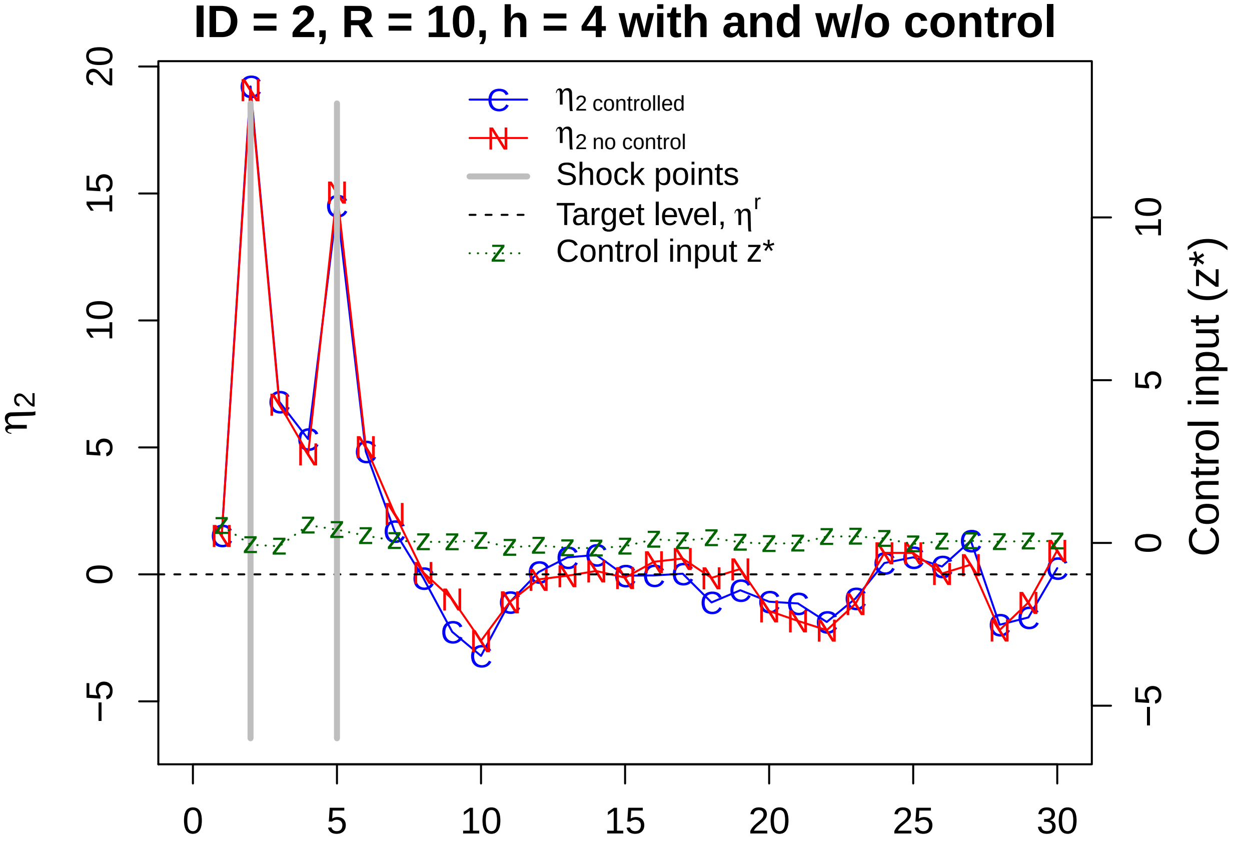

ID = 3, R = 1, h = 4 with and w/o control
η_1 controlled
η_1 no control
Shock points
Target level, η^r
Control input z*
η_1
Control input (z*)
20
15
10
5
0
−5
−10
10
5
0
−5
0
5
10
15
20
25
30

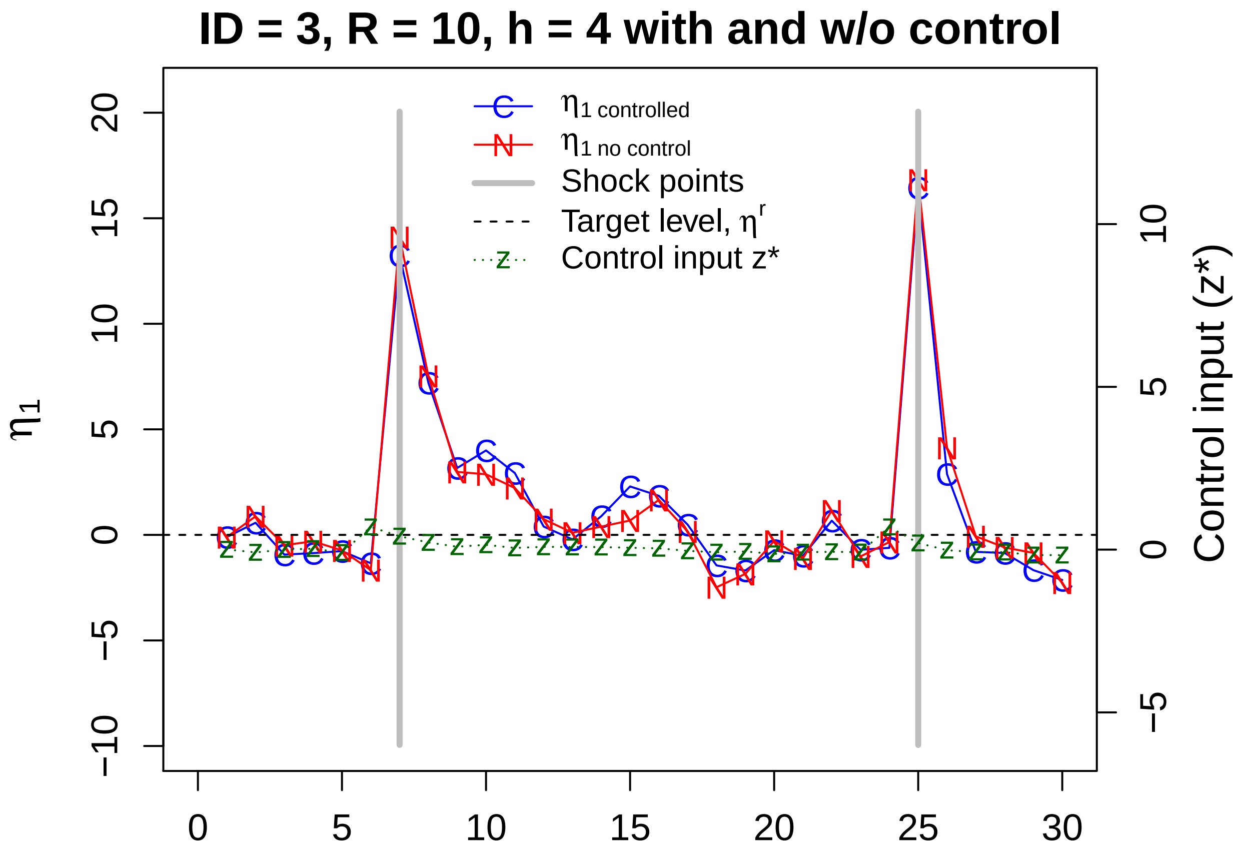

ID = 3, R = 1, h = 4 with and w/o control
η_2
Control input (z*)
η_2 controlled
η_2 no control
Shock points
Target level, η^r
Control input z*
−5
0
5
10
15
20
0
5
10
15
20
25
30
Simulated LV
LV 1
Simulated LV
LV 2
Simulated data
Observed y1
Simulated data
Observed y2

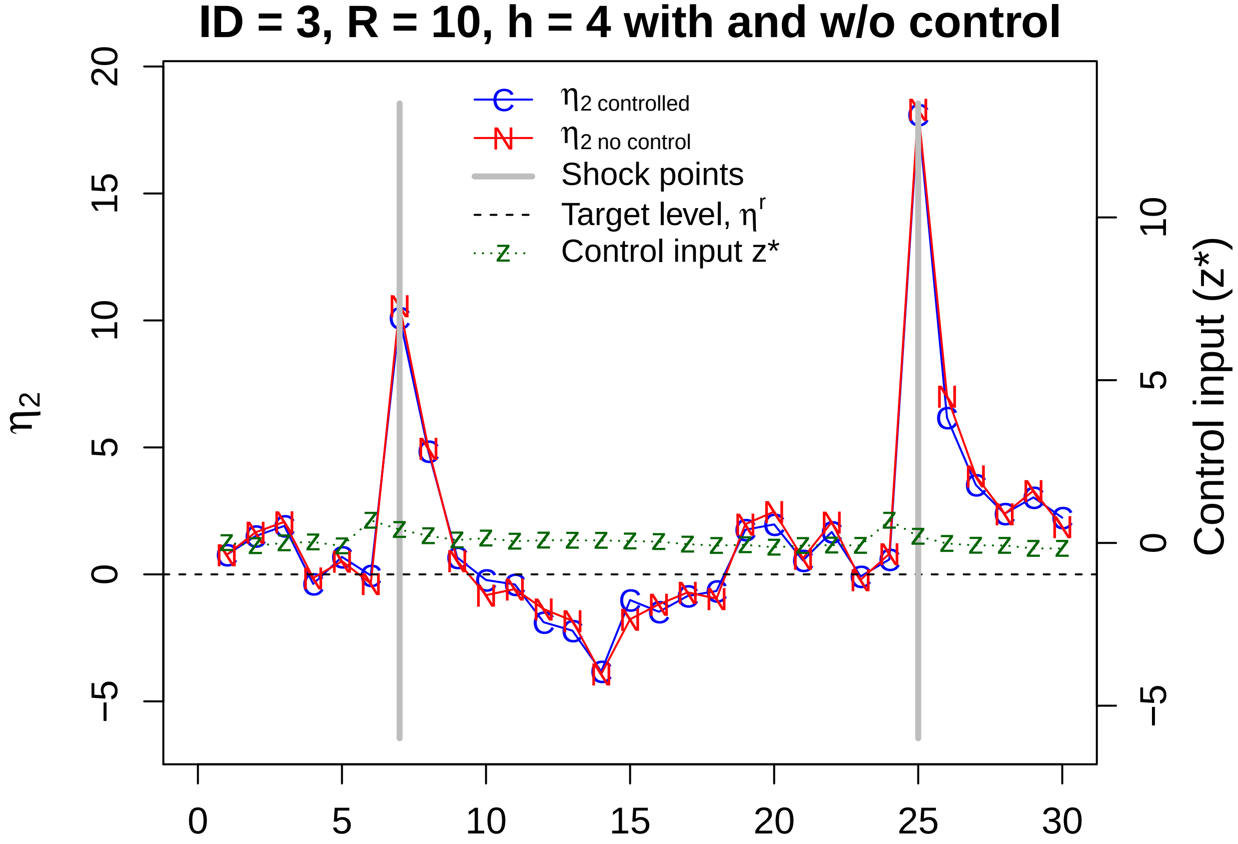

ID = 1, R = 10, h = 4 with and w/o control
η1
Control input (z*)
η1 controlled
η1 no control
Shock points
Target level, η^r
Control input z*
−10
−5
0
5
10
15
20
0
5
10
15
20
25
30
−5
0
5
10


ID = 1, R = 10, h = 4 with and w/o control
η2
Control input (z*)
η2 controlled
η2 no control
Shock points
Target level, η^r
Control input z*
−5
0
5
10
15
20
0
5
10
15
20
25
30
−5
0
5
10


ID = 2, R = 10, h = 4 with and w/o control
η_1 controlled
η_1 no control
Shock points
Target level, η^r
Control input z*
η_1
Control input (z*)
20
15
10
5
0
−5
−10
10
5
0
−5
0
5
10
15
20
25
30


ID = 2, R = 10, h = 4 with and w/o control
η_2 controlled
η_2 no control
Shock points
Target level, η^r
Control input z*
η_2
Control input (z*)
20
15
10
5
0
−5
10
5
0
−5
0
5
10
15
20
25
30


ID = 3, R = 10, h = 4 with and w/o control
η_1 controlled
η_1 no control
Shock points
Target level, η^r
Control input z*
η_1
Control input (z*)
20
15
10
5
0
−5
−10
10
5
0
−5
0
5
10
15
20
25
30


ID = 3, R = 10, h = 4 with and w/o control
η_2 controlled
η_2 no control
Shock points
Target level, η^r
Control input z*
η_2
Control input (z*)
20
15
10
5
0
−5
10
5
0
−5
0
5
10
15
20
25
30

```
colnames(X_dev) = c(t(outer(paste0("R",Rvalues),c("True_NoShock","True_Shock",
                   "True_zstar","Filtered_old","Filtered_zstar"), FUN=paste0)))
colnames(costBenefitMatrix) = c("R","cost","benefit")
```

Step 3. Inspecting graphical output and summary statistics

```
#Calculate root mean squared deviations from target values to compare
#X_NoControl and X_Control to eta_target and see which ones are closer
RMS_Dev = cbind(Rvalues,matrix(sqrt(colMeans(X_dev,na.rm=noTime)),ncol=5,byrow=TRUE))
colnames(RMS_Dev) = c("R","True_NoShock","True_Shock",
                   "True_zstar","Filtered_old","Filtered_zstar")
RMS_Dev
```

```
##              R True_NoShock True_Shock True_zstar Filtered_old Filtered_zstar
## [1,]  0.00001     1.403636   4.556103   3.391992     4.003549       3.852520
## [2,]  0.00100     1.403636   4.556103   3.391918     4.003549       3.852717
## [3,]  0.10000     1.403636   4.556103   3.406835     4.003549       3.869622
## [4,]  1.00000     1.403636   4.556103   3.785998     4.003549       3.932429
## [5,] 10.00000     1.403636   4.556103   4.428659     4.003549       3.988060
```

```
costBenefitMatrix
```

```
##              R         cost    benefit
## [1,]  0.00001   0.02820733 0.44600872
## [2,]  0.00100   2.81278024 0.44603268
## [3,]  0.10000 217.66360491 0.44111768
## [4,]  1.00000 564.36824619 0.30960486
## [5,] 10.00000 187.12506758 0.05517544
```

Notice that with the control input (under “True_zstar”), the system generally shows smaller root mean squared deviations from the target level than when no control input is applied (see “True_Shock”) regardless of the R values used, even though The root mean squared deviation values are still slightly higher than if no shocks are applied to the system. Similarly, the filtered estimates with control input (“Filtered_zstar”) also have smaller root mean squared deviation values than the filtered estimates with no control (“Filtered_old”) across all R values, even though the root mean squared deviation and relative benefit are notably lower at $R = 10$, suggesting that slightly more liberal application of the input (i.e., assigning a lower cost value to the input) is necessary in this case to counteract the effects of the shocks.

Verify from the plots over time that when the control input values are applied, the individual trajectories within the system all return to their target baseline (= 0) more efficaciously than if no control input is fed into the system.