NLMS Algorithm on echo cancelation

Creat random signal generator

```
sig_sam <- function(n) {

  stopifnot(length(n)== 1, class(n) == "numeric")
  stopifnot(n > 0)
  n <- ceiling(n)

  data <- data.frame()
  for (i in c(1:n)){
    sig <- sample(c(-1,1), 1, replace = TRUE)
    amp <- c(sample(c(0:1), 1, replace = TRUE), sample(c(0:1), 14, replace = TRUE))
    newsample <- sig * sum(amp * 2^c(14:0))
    data <- rbind(data, newsample)
  }
  colnames(data) <- "sig_far"
  return(data)
}
```

Generating “sig_close”, “echo” (some combination of L delay of “sig_close + Noise”)

We have (n + L) sample points with delay (L) We will use the rest n_test sample for corss validation

```
n <- 125                ##training data
L <- 25                 ##lags
n_test <- 3000          ##testing data

sig_far <- sig_sam(n + L + n_test)
par <- rnorm(L + 1)
echo <- data.frame(par[1] * sig_far)
for (i in c(1:L)) {
  echo <- cbind(echo, par[i + 1] * c(rep(NA,i),
                      sig_far[-((n + L + n_test - i + 1):(n + L + n_test)),]) + (rnorm(1)))
}
colnames(echo) <- par
echo_sum <- rowSums(echo[,(1:(L + 1))])
data <- data.frame(sig_far, echo, echo_sum)
train_data <- data[1:(n + L),]
```

The first column is sig_far (original signal)

followed by echo with different lag, with parameter (generated normal distribution) shown in the heading

The last column is sig_close (recieveing signal original + echo)

Imoritant!!: the “par” is the parameter of corresponding lag. It is the actual object we want to predict.

We will use above mini-data to test the echo-cancelation algorithm.

We apply echo-cancelation algorighm started at the (L+1) step

Our goal is to use sig_far to predict parameters of echos

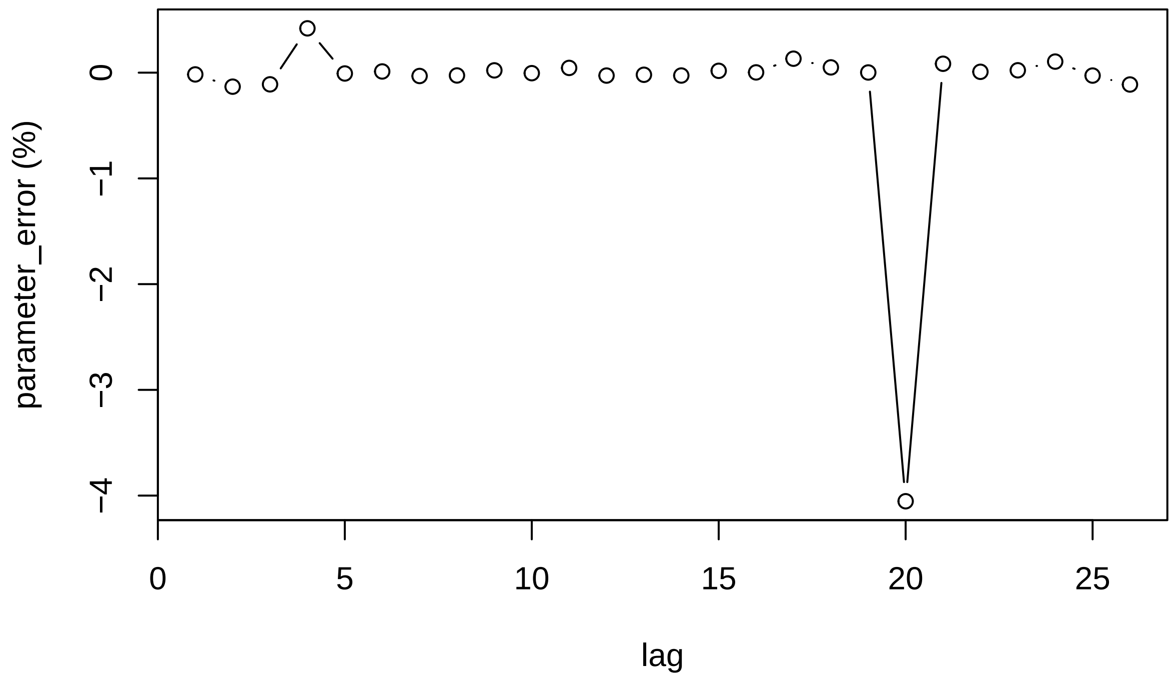
```
x <- train_data[,1]
y <- train_data[,length(train_data[1,])]

mu <- 1
gamma <- 0.01
h <- rep(0, L + 1)
p <- rep(0, L)
g <- rep(0, L)
e <- rep(0, L)
amp_h <- rep(0,L)
for (i in c((L + 1):(n + L))) {
  p[i] <- sum(x[(i):(i - L)] * x[(i):(i - L)])
  g[i] <- sum(h * x[(i):(i - L)])
  e[i] <- y[i] - g[i]
  dh <- (1 * mu / (gamma + p[i])) * e[i] * x[(i):(i - L)]
  h <- h + dh
  amp_h[i] <- sum(dh * dh)
}
sol <- list("p" = p, "g" = g, "e" = e, "amp_h" = amp_h, "h" = h)
sol$h
```

```
##  [1]  1.52516416 -0.93636255  0.43885702  0.50171227  0.98292608
##  [6] -2.40704166 -1.19240351 -0.77817083 -0.28104937  0.25780722
## [11]  0.38866679  0.13855688 -1.09129012 -2.13114265 -1.29943102
## [16] -1.15917303 -0.15320472  1.48100350  1.18634555 -0.11062513
## [21] -0.80637714  0.59995939  0.06512661  0.40157511 -0.63168884
## [26]  0.82226537
```

```
par
```

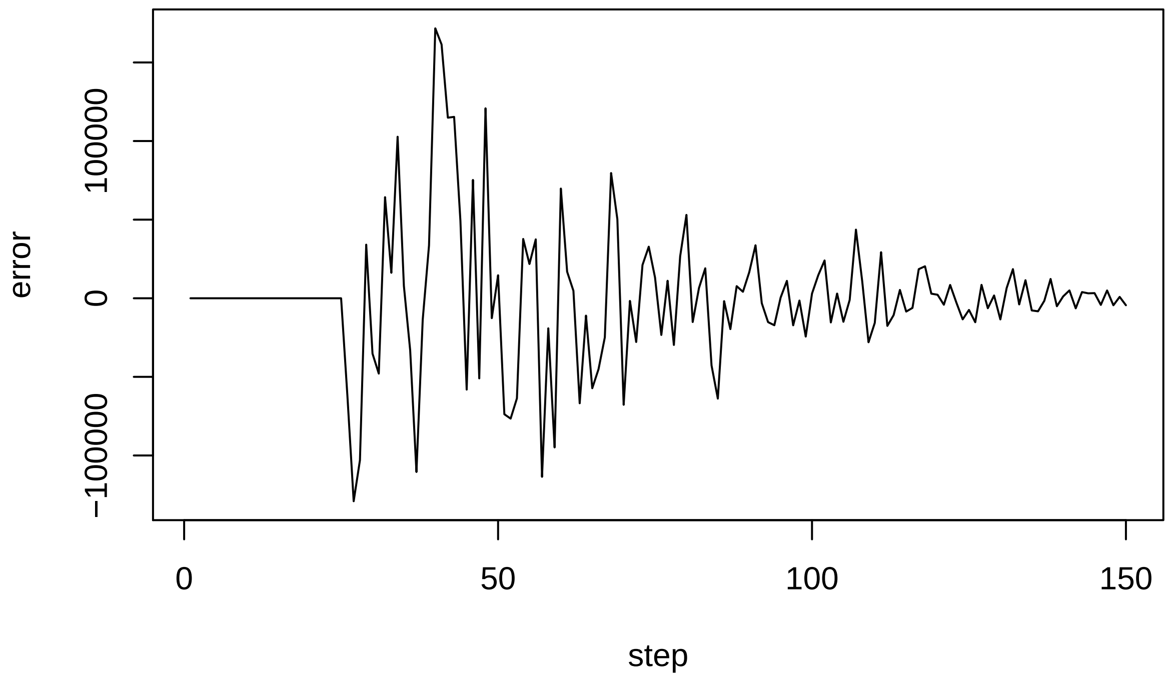
```
##  [1]  1.55094321 -0.82688562  0.49346582  0.35357207  0.99069757
##  [6] -2.43460931 -1.15583245 -0.75807931 -0.28725630  0.25909784
## [11]  0.37197812  0.14246981 -1.07000018 -2.07488152 -1.32233825
## [16] -1.16203970 -0.17652738  1.41083818  1.18444797 -0.02189171
## [21] -0.88038429  0.59517680  0.06370787  0.36357087 -0.61479724
## [26]  0.92611710
```

```
plot((sol$h-par) / abs(par), type = "b", ylab = "parameter_error (%)",xlab = "lag")
```

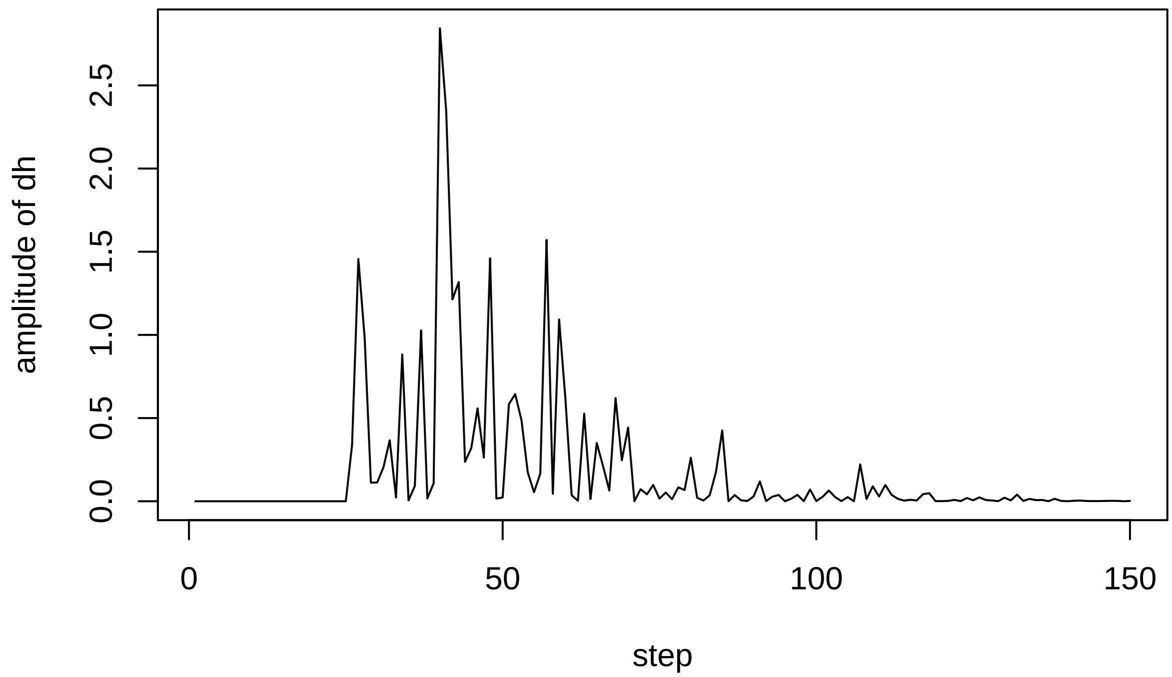


We can see “sol$h” predict “par” very actuarily. That means our training is pretty succesful.

```
plot(sol$e, type = "l", ylab = "error", xlab = "step")
```




```r
plot(sol$amp_h, type = "l", ylab = "amplitude of dh", xlab = "step")
```

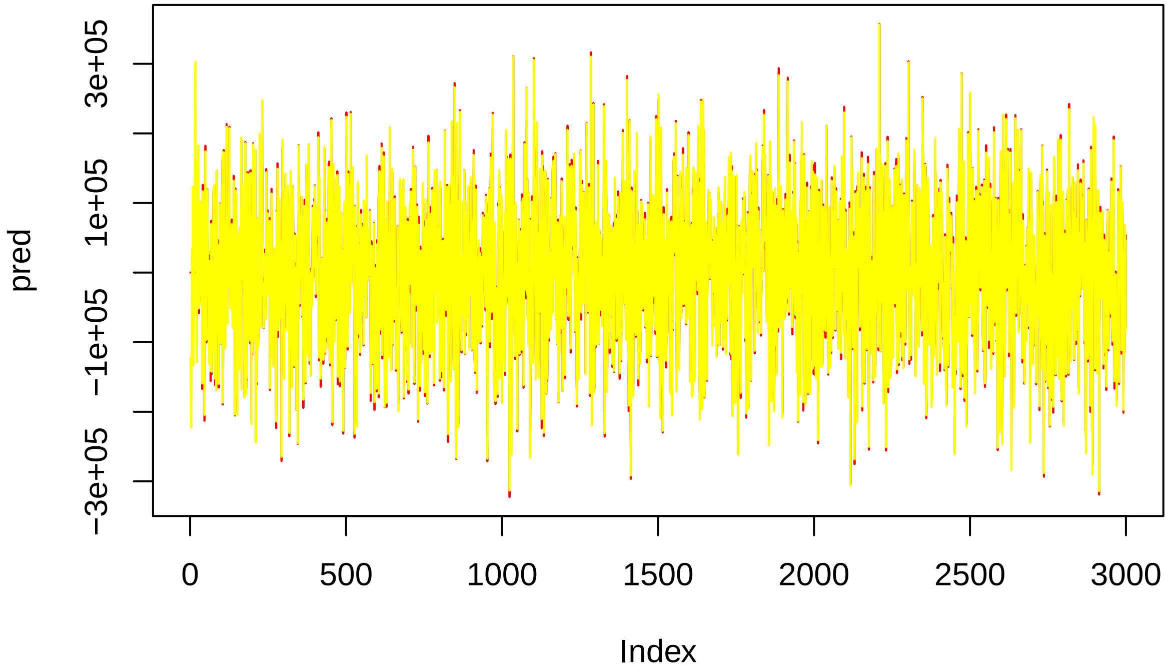


We can see the error of predicton converge to 0.

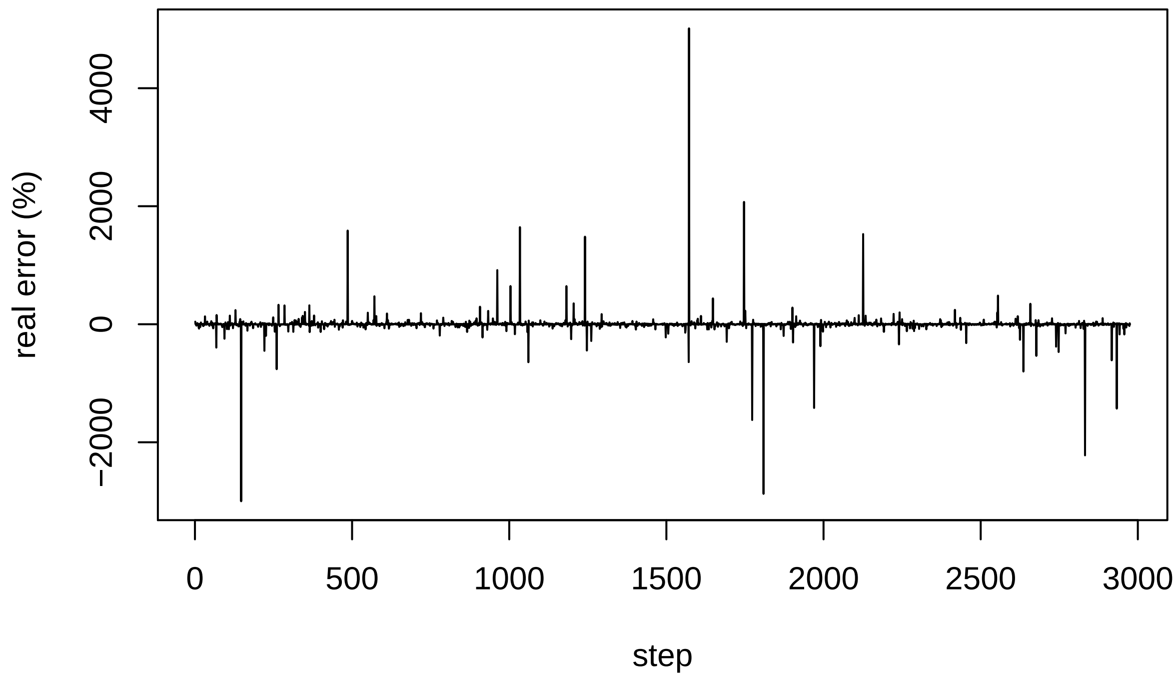
also the amplitude of parameter correction also converge to 0.

We we can test our result

```
test_data <- data[-(1:(n + L)),]
x <- test_data[,1]
y <- test_data[,length(test_data[1,])]

pred <- rep(0,L)
for (i in c((L + 1):(n_test))) {
  pred[i] <- sum(sol$h * x[(i):(i - L)])
}
real <- y

plot(pred,type = "l",col = "red")
lines(real,type = "l", col = "yellow")
```




```
error <- ((pred-real)/real * 100)[-(1:L)]
plot(error, type = "l", xlab = "step", ylab = "real error (%)")
```




```
sd(error)
```

```
## [1] 161.9746
```

```
mean(error)
```

```
## [1] -0.2420409
```

```
table((error <= 10^(-2)))
```

```
##
## FALSE  TRUE
##  1571  1404
```

standard deviation and mean of error

Over 99% of prediction are only 0.01% off.

We now show the algorithm step by step on a mini data

creat mini data

```
n <- 10
L <- 2
n_test <- 0

sig_far <- sig_sam(n + L + n_test)
par <- rnorm(L + 1)
echo <- data.frame(par[1] * sig_far)
```

```
for (i in c(1:L)) {
  echo <- cbind(echo, par[i + 1] * c(rep(NA,i),
                        sig_far[-((n + L + n_test - i + 1):(n + L + n_test)),]) + rnorm(1) * 10^15)
}
colnames(echo) <- par
echo_sum <- rowSums(echo[,(1:(L + 1))])
data <- data.frame(sig_far, echo, echo_sum)
train_data <- data[1:(n + L),]
x <- train_data[,1]
y <- train_data[,length(train_data[1,])]

print(x)
```

```
##  [1] -20814   7522  15125  -1998  18916  24716 -23857  20081  20225 -25410
## [11] -23599  26909
```

```
print(y)
```

```
##  [1]            NA            NA -1.364722e+15 -1.364722e+15 -1.364722e+15
##  [6] -1.364722e+15 -1.364722e+15 -1.364722e+15 -1.364722e+15 -1.364722e+15
## [11] -1.364722e+15 -1.364722e+15
```

Inertiallize parameters (all set to 0)

```
a <- 1
h <- rep(0, L + 1)
dh <- rep(0, L + 1)
p <- rep(0, L)
g <- rep(0, L)
e <- rep(0, L)
amp_h <- rep(0,L)
```

We started at step 3 (since it is lag 2 model)

$p[3] = x[3-0]^2 + x[3-1]^2 + x[3-2]^2$

`## [1] "p[3]="      "718568705"`

$g[3] = h_3[0]*x[3-0]+h_3[1]*x[3-1]+h_3[2]*x[3-2]$

`## [1] "g[3]=" "0"` $e[3] = y[3] - g[3]$

`## [1] "e[3]=" "-1364721522926384"`

$\Delta h_3[0] = 1 * a/p[3] * e[3] * x[3-0]$
$\Delta h_3[1] = 1 * a/p[3] * e[3] * x[3-1]$
$\Delta h_3[2] = 1 * a/p[3] * e[3] * x[3-2]$

```
## [1] "dh_3[0]=" "-28725733378.9141"
## [1] "dh_3[1]=" "-14285948196.773"
## [1] "dh_3[2]=" "39530407573.4689"
```

$h_4[1] = h_3[0] + \Delta h_3[0]$
$h_4[1] = h_3[1] + \Delta h_3[1]$
$h_4[2] = h_3[2] + \Delta h_3[2]$

`## [1] "h_4[0]=" "-28725733378.9141"`

```
## [1] "h_4[1]="
"-14285948196.773"
## [1] "h_4[2]="
"39530407573.4689"
```

We do one more step. We are now at step 4 $p[4] = x[4-0]^2 + x[4-1]^2 + x[4-2]^2$
`## [1] "p[4]="      "289338113"` $g[4] = h_4[0]*x[4-0]+h_4[1]*x[4-1]+h_4[2]*x[4-2]$
`## [1] "g[4]=" "138666774582512"` $e[4] = y[4] - g[4]$
`## [1] "e[4]=" "-1503388297525803"`
$\Delta h_4[0] = 1 * a/p[4] * e[4] * x[4-0]$
$\Delta h_4[1] = 1 * a/p[4] * e[4] * x[4-1]$
$\Delta h_4[2] = 1 * a/p[4] * e[4] * x[4-2]$

```
## [1] "dh_4[0]="
"10381521422.5046"
## [1] "dh_4[1]="
"-78588844602.2932"
## [1] "dh_4[2]="
"-39083986056.0958"
```

$h_5[1] = h_4[0] + \Delta h_4[0]$
$h_5[1] = h_4[1] + \Delta h_4[1]$
$h_5[2] = h_4[2] + \Delta h_4[2]$

```
## [1] "h_5[0]="
"-18344211956.4095"
## [1] "h_5[1]="
"-92874792799.0662"
## [1] "h_5[2]="
"446421517.37307"
```

The complete calculation

```
## $p
##  [1]          0          0  718568705  289338113  590572685  972687716
##  [7] 1537852161 1583283666 1381453635 1457965286 1611631526 1926675182
##
## $g
##  [1]  0.000000e+00  0.000000e+00  0.000000e+00  1.386668e+14 -1.546832e+14
##  [6] -3.029681e+15 -1.683530e+15  8.441661e+13 -1.058207e+14 -1.495989e+14
## [11]  1.512314e+15  2.764211e+15
##
## $e
##  [1]  0.000000e+00  0.000000e+00 -1.364722e+15 -1.503388e+15 -1.210038e+15
##  [6]  1.664959e+15  3.188081e+14 -1.449138e+15 -1.258901e+15 -1.215123e+15
## [11] -2.877036e+15 -4.128932e+15
##
## $amp_dh
## [1] 0 0
##
## $h
```

```
##   X3           4           5           6           7           8
## 1  0 -28725733379 -18344211956 -57101652671 -14795032469 -19740763898
## 2  0 -14285948197 -92874792799 -88781043246 -56402340068 -51278531626
## 3  0  39530407573    446421517 -30543549450 -33963545694 -30042119877
##             9          10          11          12          13
## 1 -38120378088 -56551159482 -35373518375   6754701662 -50912226944
## 2 -29442843377 -47742399430 -64598668276 -19237505891  31335976890
## 3 -52664028363 -30923452259 -47659706138 -83764763566 -29310244414
```