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

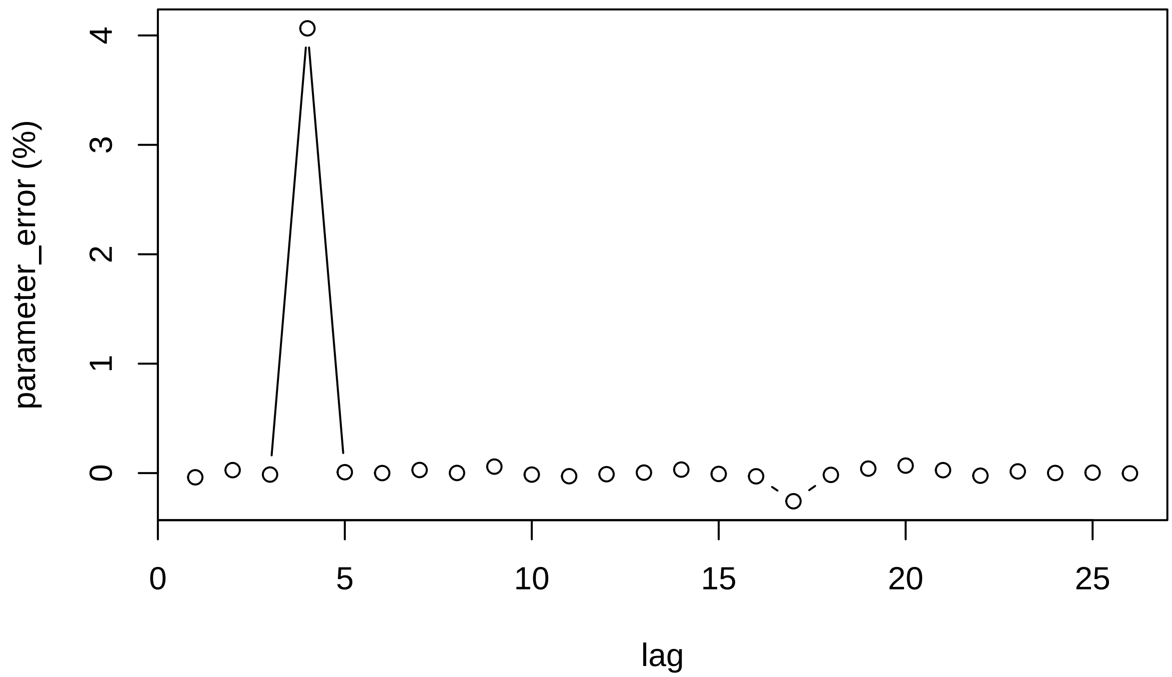
```r
x <- train_data[,1]
y <- train_data[,length(train_data[1,])]

mu <- 1
gamma <- 0.01
h <- rep(1, 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]  1.00668941  1.41336202 -2.25162161  0.03669472  0.40123357
##  [6] -1.33621665 -0.86079368 -1.59623870  0.39857835 -0.87163544
## [11] -0.51264700  0.64769399 -1.00789707 -0.18990166  1.80214166
## [16]  0.29167304 -0.17019626 -0.82528014  0.57607165 -0.08847831
## [21]  0.81384446  1.18357898 -0.39545046 -0.14099597 -2.58482352
## [26]  1.61941912
```

```r
par
```

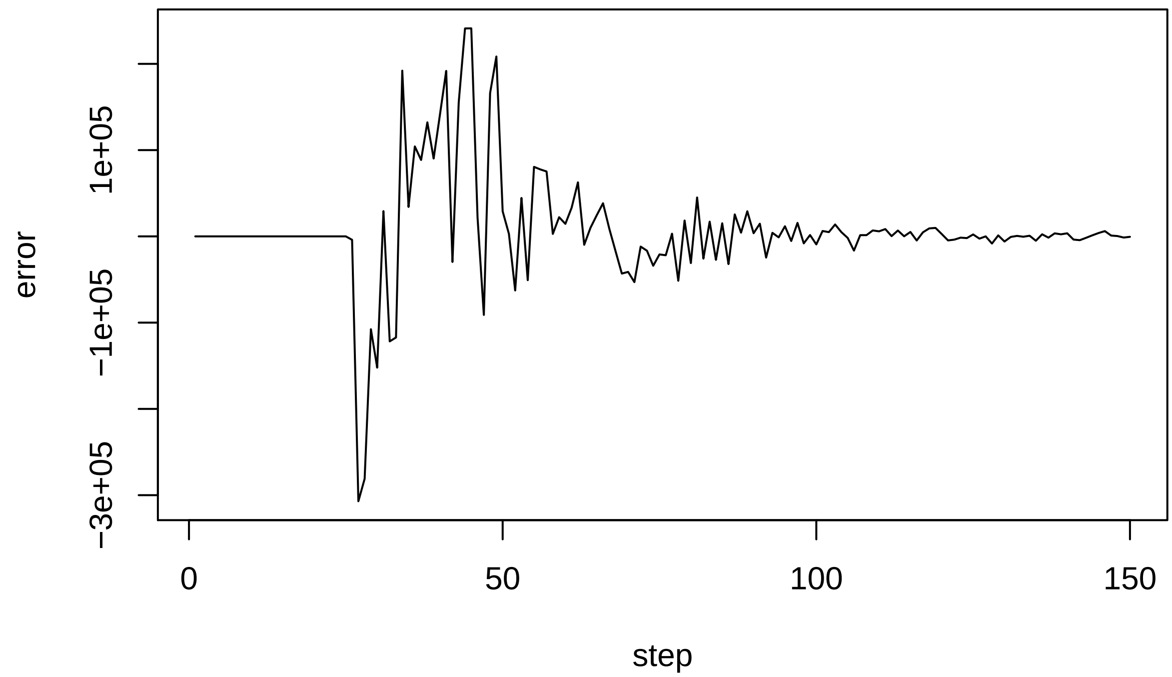
```
##  [1]  1.047178800  1.376265893 -2.221888360  0.007244976  0.398037639
##  [6] -1.336582806 -0.885792342 -1.598150871  0.376356090 -0.859578864
## [11] -0.498445914  0.654432256 -1.012230784 -0.196144602  1.816456331
## [16]  0.300538535 -0.135358973 -0.811944501  0.553497147 -0.094955845
## [21]  0.792687550  1.212692627 -0.401599532 -0.141111254 -2.595081197
## [26]  1.623365286
```

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

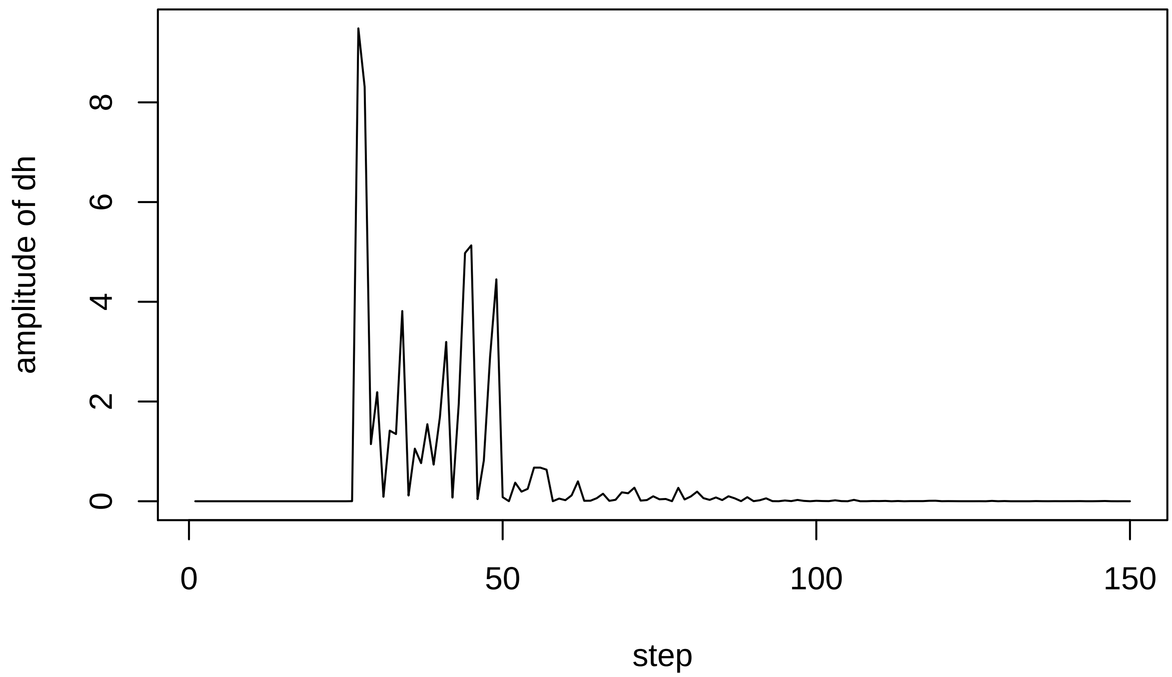


We can see “sol$h” predict “par” very actuarily. That means our training is pretty succesful.

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




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

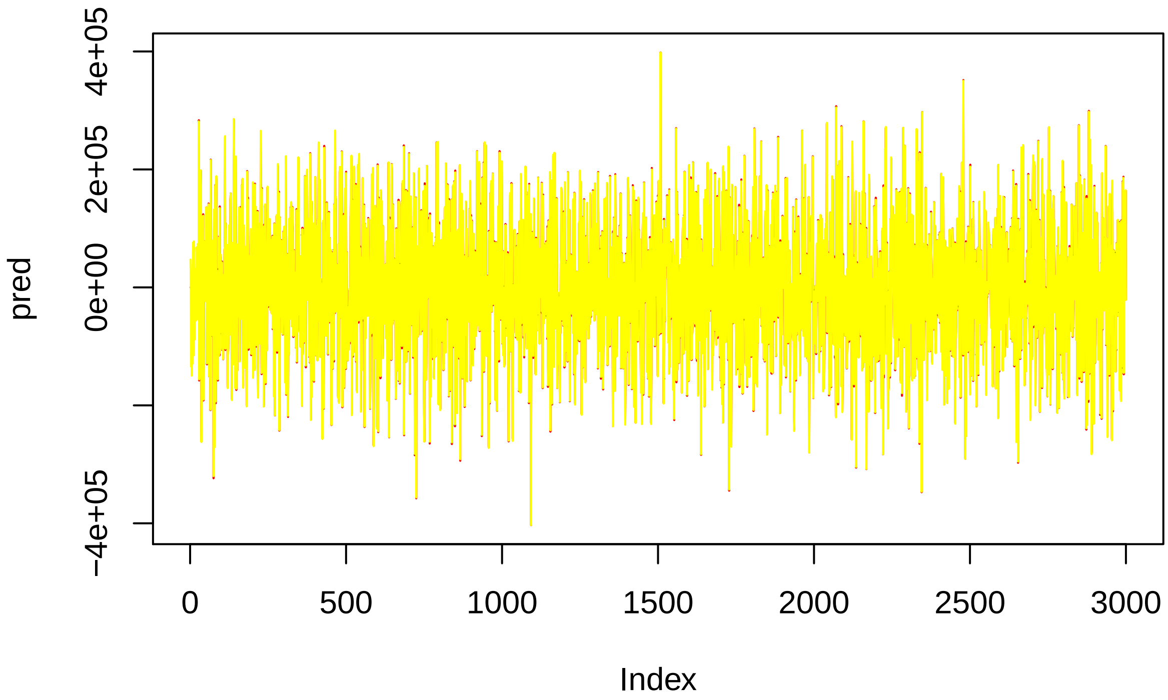


We can see the error of predicton converge to 0.

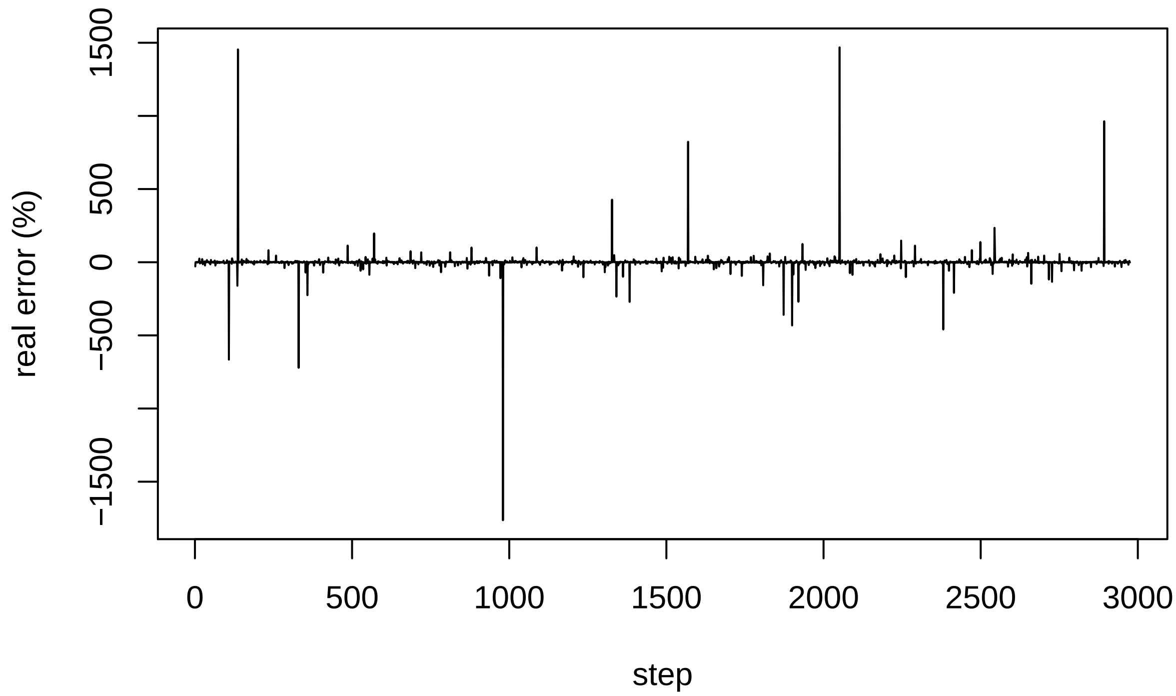
also the amplitude of parameter correction also converge to 0.

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] 62.53136
```

```
mean(error)
```

```
## [1] -0.2173333
```

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

```
##
## FALSE  TRUE
##  1482  1493
```

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)
```

```r
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]   6596  32485 -28554  27689 -15736 -19328 -20895 -31377  16986  32309
## [11]  32320  -6323
```

```r
print(y)
```

```
##  [1]            NA            NA -2.152419e+15 -2.152419e+15 -2.152419e+15
##  [6] -2.152419e+15 -2.152419e+15 -2.152419e+15 -2.152419e+15 -2.152419e+15
## [11] -2.152419e+15 -2.152419e+15
```

Inertiallize parameters (all set to 0)

```r
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]="      "1914113357"
```

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

```
## [1] "g[3]=" "0"
```

$e[3] = y[3] - g[3]$

```
## [1] "e[3]=" "-2152418828730694"
```

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

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

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

```
## [1] "dh_3[0]=" "32108948516.9797"
## [1] "dh_3[1]=" "-36529354646.4273"
## [1] "dh_3[2]=" "-7417196344.40002"
```

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

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

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

```
## [1] "h_4[0]=" "32108948516.9797"
```

```
## [1] "h_4[1]="
"-36529354646.4273"
## [1] "h_4[2]="
"-7417196344.40002"
```

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]="        "2637286862"
```

$g[4] =$
$h_4[0]*x[4-0]+h_4[1]*x[4-1]+h_4[2]*x[4-2]$

```
## [1] "g[4]="
"1691176244812903"
```

$e[4] = y[4] - g[4]$

```
## [1] "e[4]="
"-3843595073608213"
```

$\Delta h_4[0] = 1 * a/p[4] * e[4] * x[4-0]$
$\Delta h_4[1] = 1 * a/p[4] * e[4] * x[4-1]$
$\Delta h_4[2] = 1 * a/p[4] * e[4] * x[4-2]$

```
## [1] "dh_4[0]="
"-40354087197.1848"
## [1] "dh_4[1]="
"41614742526.939"
## [1] "dh_4[2]="
"-47343801603.5446"
```

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

```
## [1] "h_5[0]="
"-8245138680.20506"
## [1] "h_5[1]="
"5085387880.51167"
## [1] "h_5[2]="
"-54760997947.9447"
```

The complete calculation

```
## $p
##  [1]          0          0 1914113357 2637286862 1829633333 1387874001
##  [7] 1057794305 1794688738 1709641350 2316911806 2376978077 2128434210
##
## $g
##  [1]  0.000000e+00  0.000000e+00  0.000000e+00  1.691176e+15  1.834200e+15
##  [6]  5.724592e+14 -1.282706e+14 -3.201738e+15  2.028470e+15  2.013193e+15
## [11]  2.402376e+15  1.499771e+15
##
## $e
##  [1]  0.000000e+00  0.000000e+00 -2.152419e+15 -3.843595e+15 -3.986619e+15
##  [6] -2.724878e+15 -2.024148e+15  1.049319e+15 -4.180889e+15 -4.165612e+15
## [11] -4.554794e+15 -3.652190e+15
##
## $amp_dh
## [1] 0 0
##
```

```
## $h
##   X3           4           5           6           7            8
## 1  0  32108948517  -8245138680  26042299218  63989867383 103973612338
## 2  0 -36529354646   5085387881 -55246644909 -24351418676  12633784603
## 3  0  -7417196344 -54760997948   7455798046 -46907313238 -16795601794
##              9          10          11          12          13
## 1  85628105670 44089228515 -13999631880 -75931610040 -65081944490
## 2    416894465 77148635223  46609238068 -15301661757 -70759702335
## 3 -28096298256 23001949605  79415152508  46866372155  -8572793471
```