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(1:9), 1, replace = TRUE), sample(c(0:9), 14, replace = TRUE))
    newsample <- sig * sum(amp * 10^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 <- 1000
L <- 50
n_test <- 3000

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

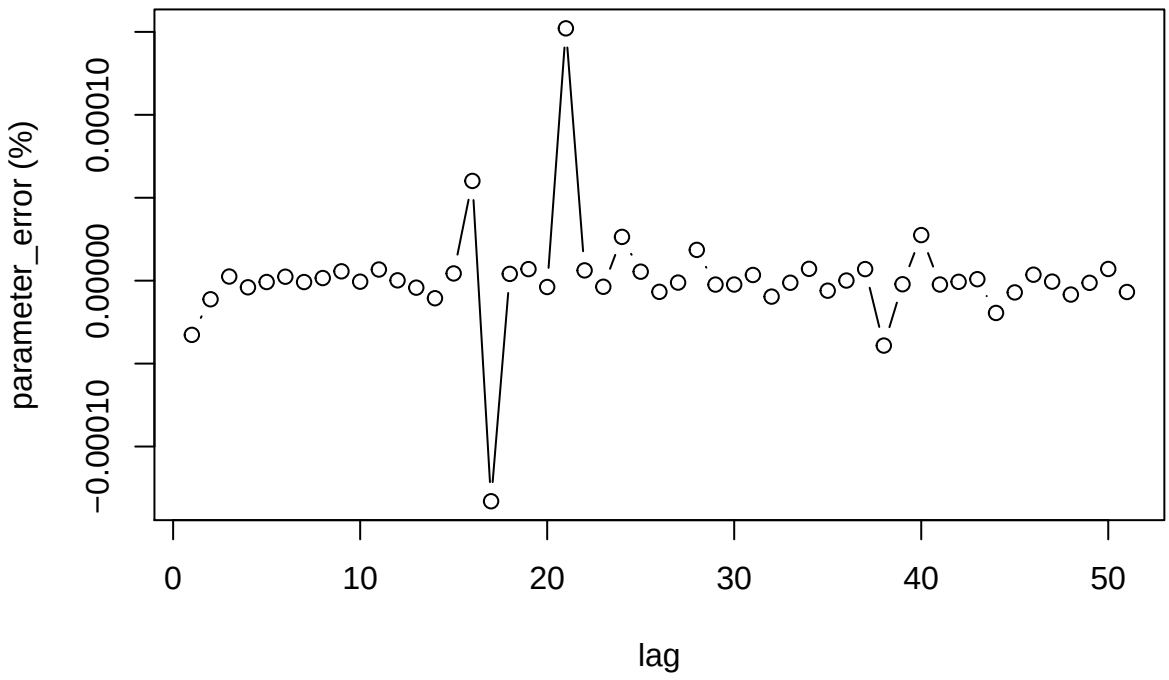
```
x <- train_data[,1]
y <- train_data[,length(train_data[1,])]

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]  0.07744853  0.68474339  1.64696390  1.73041930  0.73678824
##  [6]  1.43024851  1.26254305  0.28630975 -0.33932344  1.71178170
## [11] -0.05145589 -1.15720935  0.15696563  0.90879141 -1.27328620
## [16] -0.07602769 -0.10833445  1.38312636 -1.27700692 -2.96296715
## [21] -0.03323423 -0.59433690 -2.05322727  0.15017505 -0.71909301
## [26]  0.93385374  1.19817690  0.49154871  1.65767130  0.96395402
## [31] -1.77226639 -0.61822109  0.40790313  1.19177495 -1.16494220
## [36] -0.70852958 -1.07750822 -0.08072924  1.19643616  0.24621418
## [41]  1.15581954 -1.52696321 -1.11227330 -0.18365562  0.20740946
## [46]  0.62777994  1.26619613  0.25707580  1.05860830  0.34535718
## [51]  0.20251200
```

```
par
```

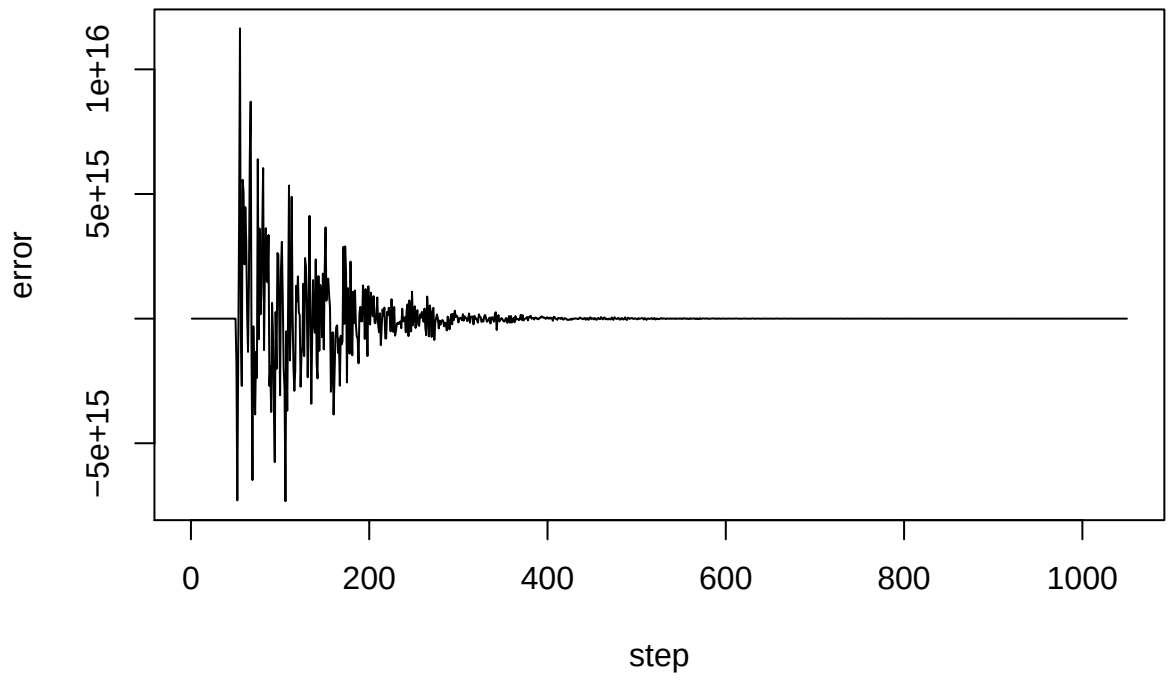
```
##  [1]  0.07745106  0.68475107  1.64695979  1.73042622  0.73678884
##  [6]  1.43024510  1.26254409  0.28630929 -0.33932535  1.71178273
## [11] -0.05145624 -1.15720958  0.15696630  0.90880103 -1.27329176
## [16] -0.07603226 -0.10832005  1.38312070 -1.27701583 -2.96295584
## [21] -0.03323928 -0.59434058 -2.05321972  0.15017109 -0.71909691
## [26]  0.93386003  1.19817828  0.49153959  1.65767519  0.96395626
## [31] -1.77227248 -0.61821515  0.40790364  1.19176645 -1.16493516
## [36] -0.70852965 -1.07751574 -0.08072608  1.19643873  0.24620741
## [41]  1.15582222 -1.52696215 -1.11227430 -0.18365204  0.20741093
## [46]  0.62777767  1.26619685  0.25707795  1.05860967  0.34535474
## [51]  0.20251338
```

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

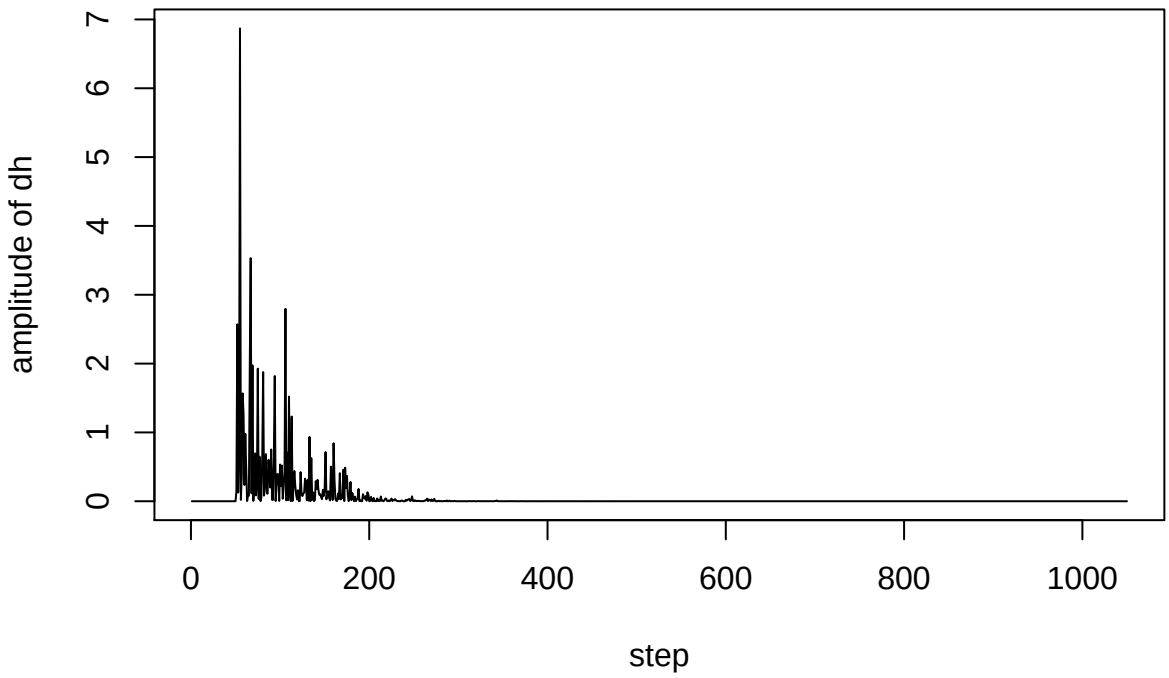


We can see “sol$h” predict “par” very actuarily. That means our training is pretty succesful.

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




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

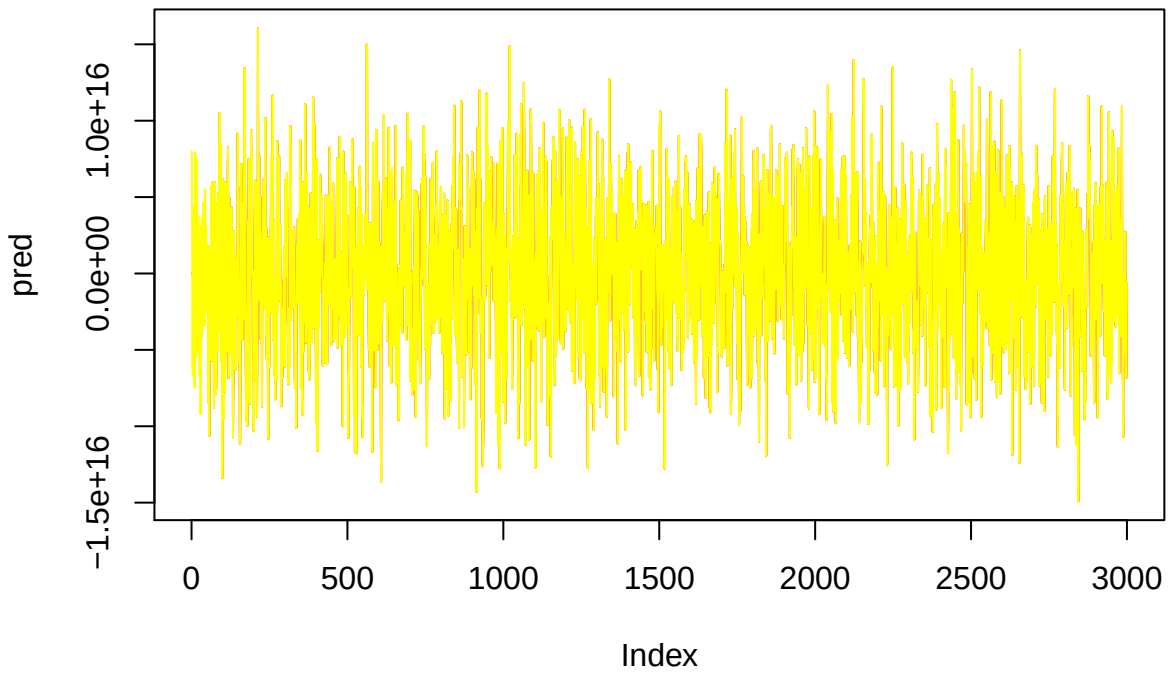


We can see the error of predicton converge to 0.

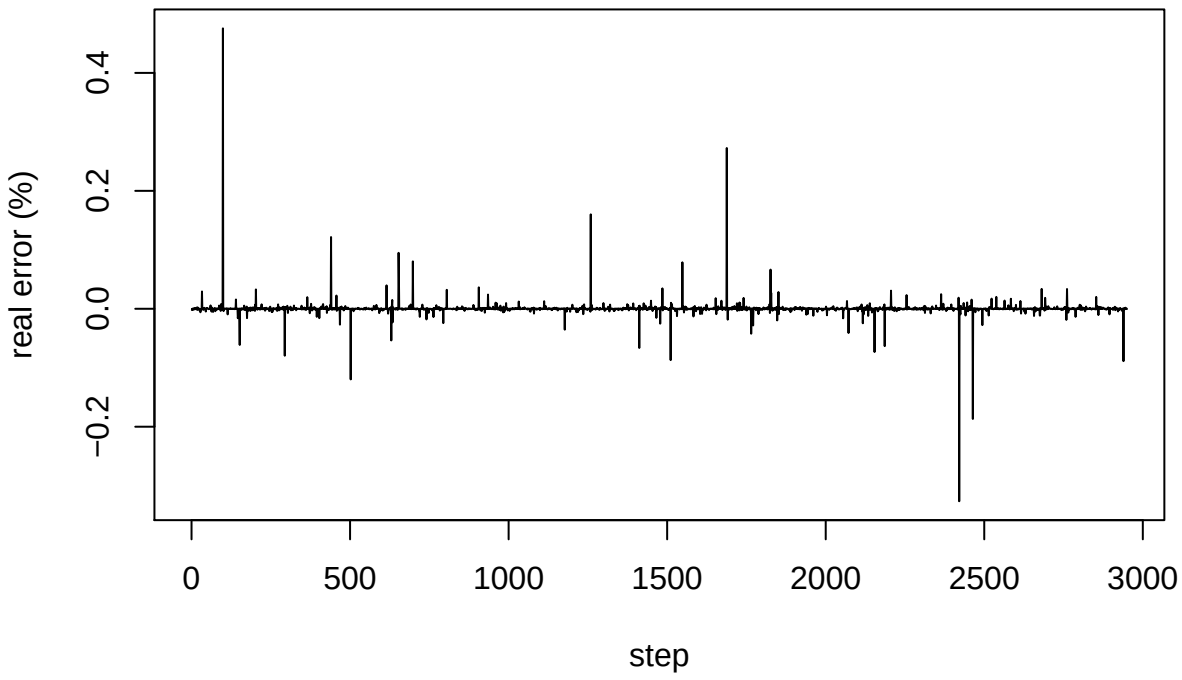
also the amplitude of parameter correction also converge to 0.

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] 0.01440683
```

```
mean(error)
```

```
## [1] 0.000127637
```

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

```
##
## FALSE  TRUE
##    47  2903
```

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]  3.200280e+14  9.697110e+14  5.748560e+14 -5.290216e+14  5.271174e+14
##  [6]  8.919979e+14  2.864493e+14 -6.936308e+14  7.120439e+14 -2.481667e+14
## [11] -5.229367e+14 -8.683361e+14
```

```
print(y)
```

```
##  [1]            NA            NA -3.190515e+15 -1.787614e+15 -6.630158e+14
##  [6] -2.239101e+15 -2.890009e+15 -1.075992e+15 -3.906024e+14 -1.349724e+15
## [11] -2.033081e+14  1.168011e+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]="
"1.37321679255697e+30"
```
$g[3] = h_3[0]*x[3-0]+h_3[1]*x[3-1]+h_3[2]*x[3-2]$
```
## [1] "g[3]=" "0"
```
$e[3] = y[3] - g[3]$
```
## [1] "e[3]="
"-3190514563171288"
```
$\Delta h_3[0] = 1 * a/p[3] * e[3] * x[3-0]$
$\Delta h_3[1] = 1 * a/p[3] * e[3] * x[3-1]$
$\Delta h_3[2] = 1 * a/p[3] * e[3] * x[3-2]$
```
## [1] "dh_3[0]="
"-1.33561319069311"
## [1] "dh_3[1]="
"-2.25301430031335"
## [1] "dh_3[2]="
"-0.743549079099084"
```
$h_4[1] = h_3[0] + \Delta h_3[0]$
$h_4[1] = h_3[1] + \Delta h_3[1]$
$h_4[2] = h_3[2] + \Delta h_3[2]$
```
## [1] "h_4[0]="
"-1.33561319069311"
```

```
## [1] "h_4[1]="
"-2.25301430031335"
## [1] "h_4[2]="
"-0.743549079099084"
```

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]="
"1.55066273869014e+30"
```

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

```
## [1] "g[4]="
"-1309618259722735"
```

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

```
## [1] "e[4]="
"-477995468636570"
```

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

```
## [1] "dh_4[0]="
"0.163072172525457"
## [1] "dh_4[1]="
"-0.17720072816138"
## [1] "dh_4[2]="
"-0.298915717032074"
```

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

```
## [1] "h_5[0]="
"-1.17254101816766"
## [1] "h_5[1]="
"-2.43021502847473"
## [1] "h_5[2]="
"-1.04246479613116"
```

The complete calculation

```
## $p
##  [1] 0.000000e+00 0.000000e+00 1.373217e+30 1.550663e+30 8.881761e+29
##  [6] 1.353377e+30 1.155566e+30 1.358837e+30 1.070183e+30 1.049717e+30
## [11] 8.420561e+29 1.089057e+30
##
## $g
##  [1]  0.000000e+00  0.000000e+00  0.000000e+00 -1.309618e+15  6.830240e+13
##  [6] -1.682564e+15 -3.222153e+15 -2.712161e+14 -6.008320e+13  2.848312e+14
## [11]  9.496238e+14  2.331716e+15
##
## $e
##  [1]  0.000000e+00  0.000000e+00 -3.190515e+15 -4.779955e+14 -7.313182e+14
##  [6] -5.565376e+14  3.321444e+14 -8.047757e+14 -3.305192e+14 -1.634555e+15
## [11] -1.152932e+15 -1.163705e+15
##
## $amp_dh
## [1] 0 0
```

```
##
## $h
##   X3          4         5         6         7         8         9
## 1  0 -1.3356132 -1.172541 -1.606566 -1.973375 -1.891040 -1.480235
## 2  0 -2.2530143 -2.430215 -1.994622 -2.211384 -1.954997 -2.124648
## 3  0 -0.7435491 -1.042465 -1.515797 -1.298252 -1.146743 -1.675032
##          10         11         12        13
## 1 -1.700146 -1.3137155 -0.5977176  0.330137
## 2 -1.910424 -3.0191756 -2.6793891 -2.120609
## 3 -1.763500 -0.6834202 -1.6583411 -1.393164
```