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 <- 64                  ##training data
L <- 3                   ##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, mean = 0, s
}
colnames(echo) <- par
echo_sum <- rowSums(echo[,(1:(L + 1))])
for (k in c(20:length(echo_sum))) {
  echo_sum[k] <- min(echo_sum[k],32768)
  echo_sum[k] <- max(echo_sum[k],-32768)
}

data <- data.frame(sig_far,echo,echo_sum)
train_data <- data[1:(n + L),]
```

Lets view the data first:

```
tail(data)
```

```
##      sig_far X.0.10871009643313 X0.763874581752673 X0.171951606507629
## 3062   28165         -3061.8199        -20916.4138          4368.6025
## 3063    5904          -641.8244         21514.5276         -4708.3789
## 3064    7630          -829.4580          4509.9155          4843.0170
```

```
## 3065    -436           47.3976          5828.3631          1015.2023
## 3066    7632          -829.6755          -333.0493          1311.9908
## 3067  -15535          1688.8113          5829.8908           -74.9709
##      X0.495555180376978   echo_sum
## 3062           7922.936 -11686.695
## 3063          12590.075  28754.399
## 3064         -13569.292  -5045.817
## 3065          13957.312  20848.275
## 3066           2925.758   3075.024
## 3067           3781.086  11224.817
```

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

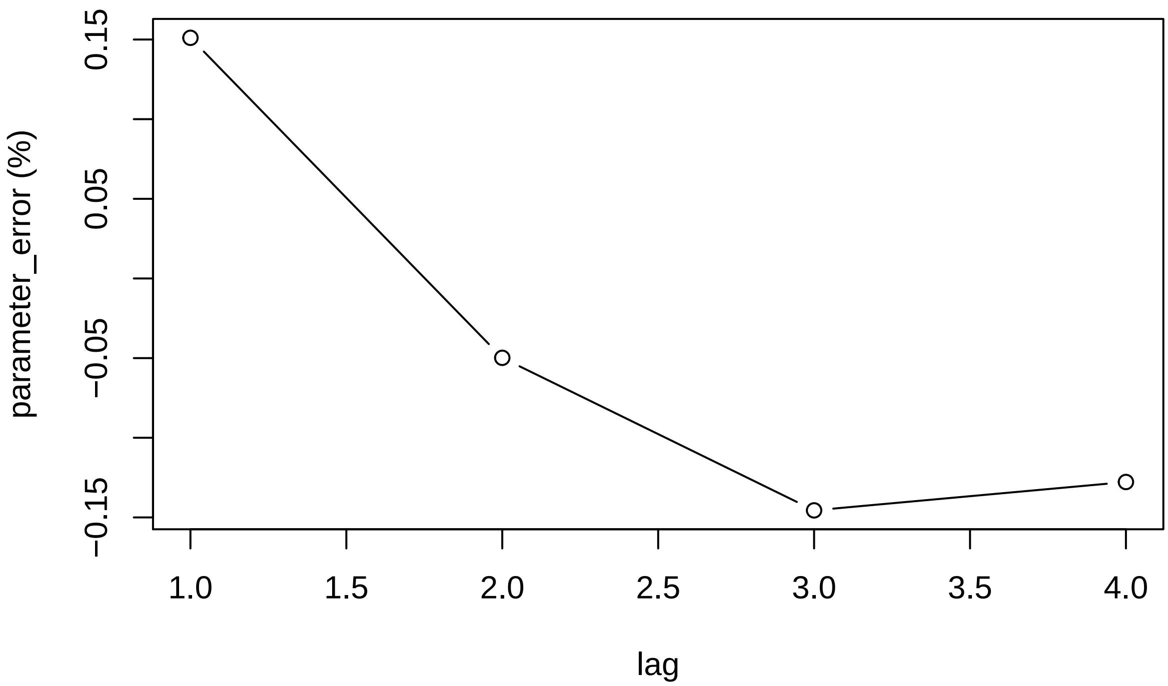
```r
x <- train_data[,1]
y <- train_data[,length(train_data[1,])]

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] -0.09228739  0.72582812  0.14692061  0.43224019
```

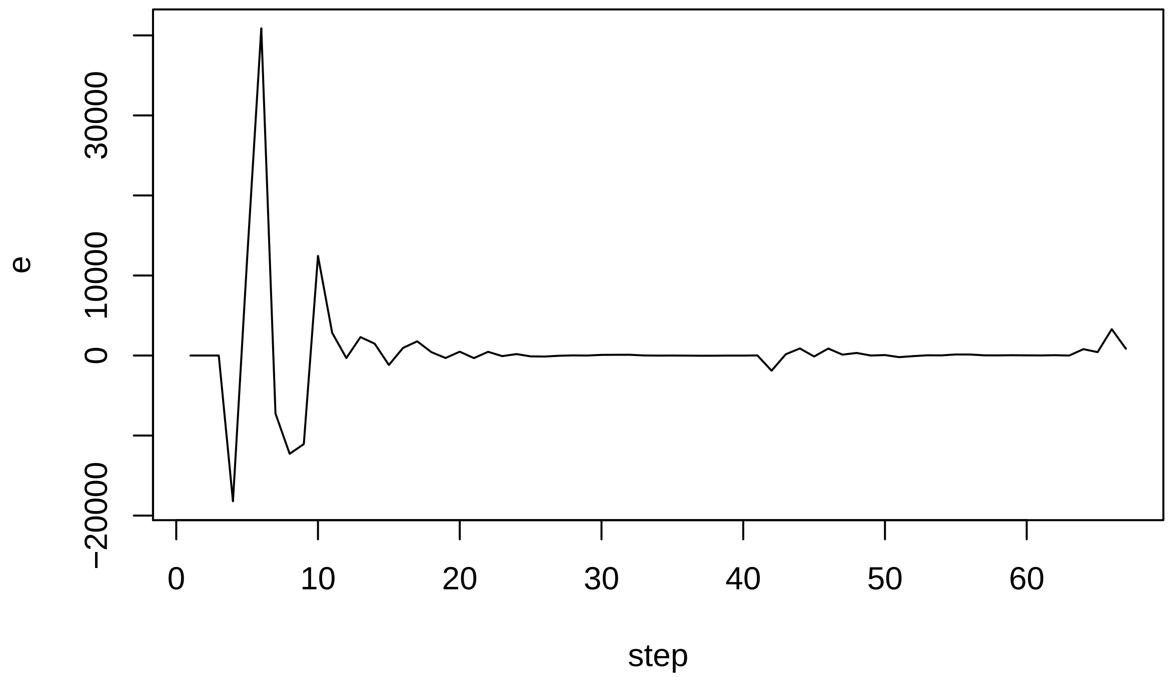
```r
par
```

```
## [1] -0.1087101  0.7638746  0.1719516  0.4955552
```

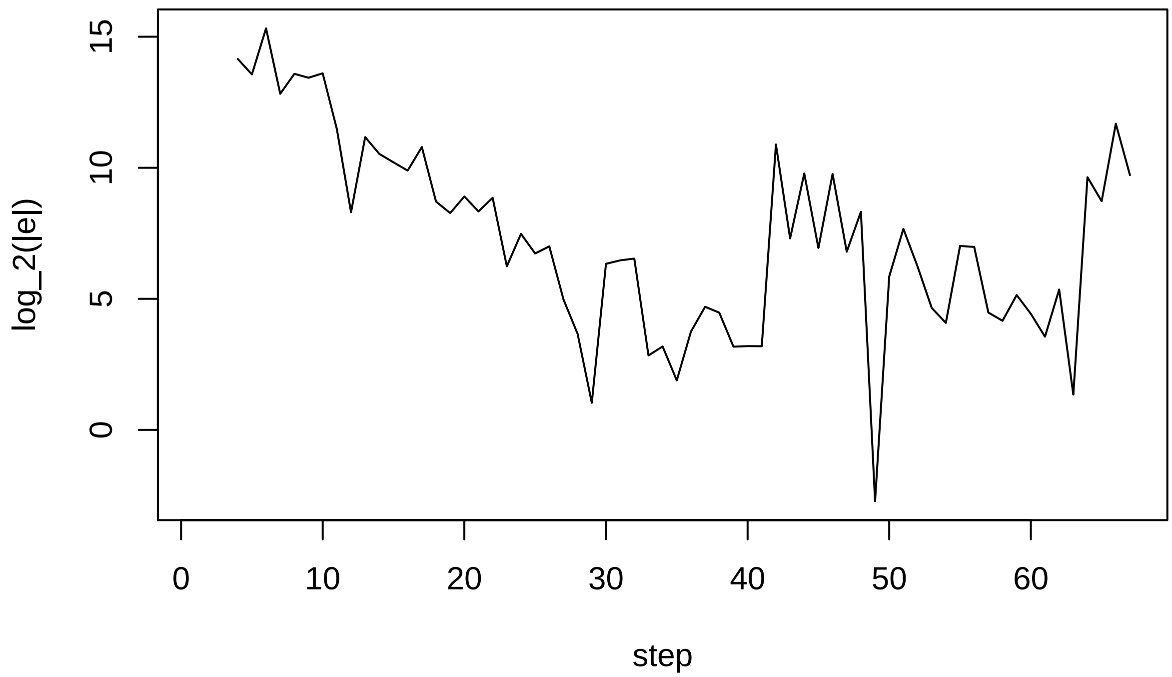
```r
plot((sol$h-par) / abs(par), type = "b", ylab = "parameter_error (%)",xlab = "lag")
```




```r
plot(e,type = "l", ylab = "e", xlab = "step")
```

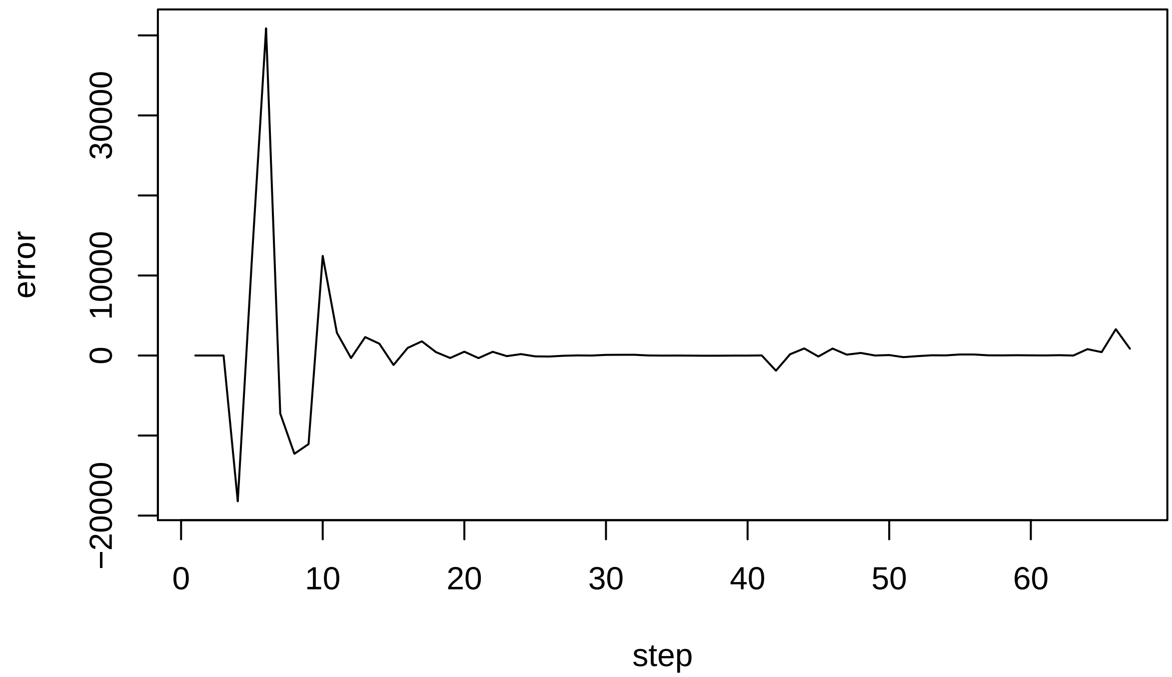



```
plot(log(abs(e),base = 2),type = "l", ylab = "log_2(|e|)", xlab = "step")
```

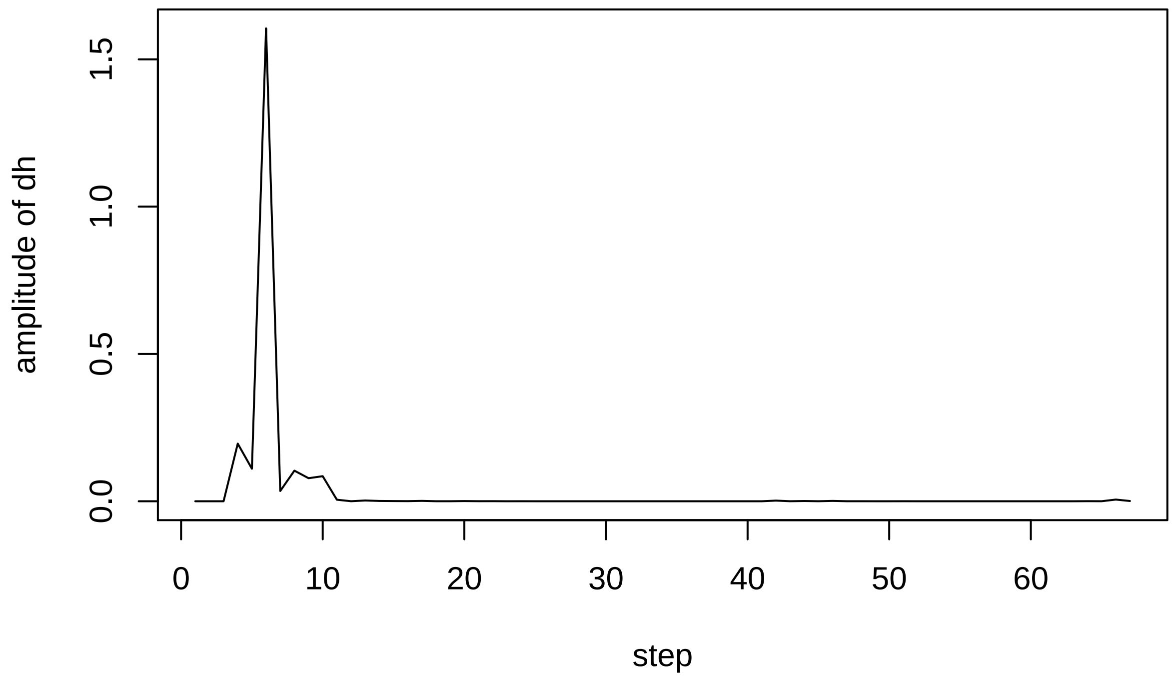


We can see “sol$h” predict “par” very actuarily. That means our training is pretty succesful.

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




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

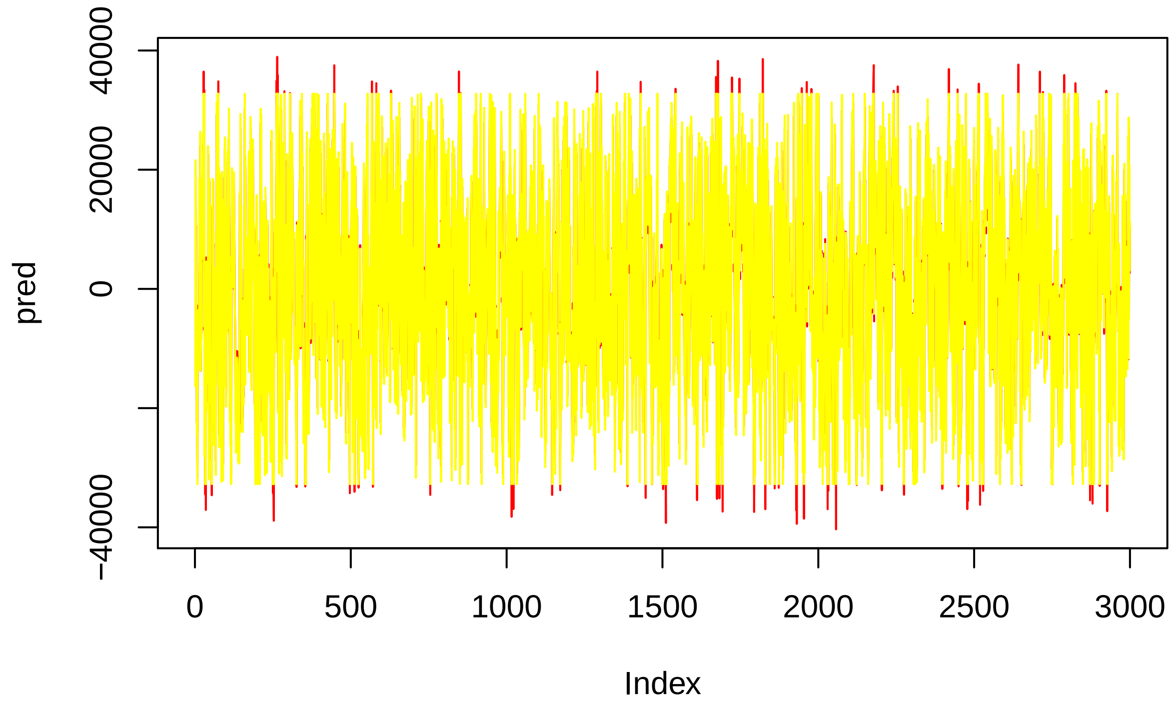


We can see the error of predicton converge to 0.

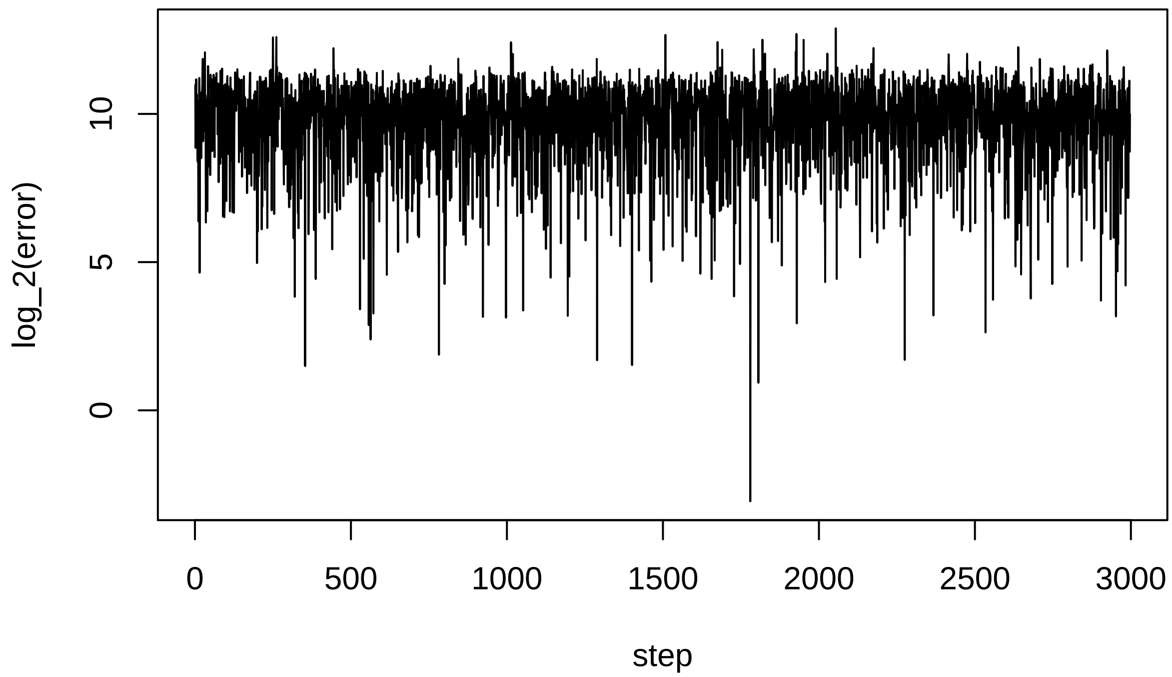
also the amplitude of parameter correction also converge to 0.

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)[-(1:L)]
plot(log(abs(error), base = 2), type = "l", xlab = "step", ylab = "log_2(error)")
```




```
sd(error)
```

```
## [1] 1443.319
```

```
mean(error)
```

```
## [1] -43.1414
```

!!Important!! expecting proformer on 16 bit datas.

```
table((log(abs(error), base = 2) <= 1))
```

```
##
## FALSE  TRUE
##  2995     2
```

```
table((log(abs(error), base = 2) <= 2))
```

```
##
## FALSE  TRUE
##  2990     7
```

```
table((log(abs(error), base = 2) <= 3))
```

```
##
## FALSE  TRUE
##  2986    11
```

In 16 bit nonideal data. With 64 samples, we are expecting:

99% of the predictions are offed by 1 digit 99% of the predictions are offed by 2 digits 99% of the predictions are offed by 3 digits

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]  -9067   5123   7811   7394  -8680 -25253 -18407  19449 -20845 -15775
## [11]  21533  24145
```

```
print(y)
```

```
##  [1]            NA            NA -1.632555e+15 -1.632555e+15 -1.632555e+15
##  [6] -1.632555e+15 -1.632555e+15 -1.632555e+15 -1.632555e+15 -1.632555e+15
## [11] -1.632555e+15 -1.632555e+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]="      "169467339"
```

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

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

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

```
## [1] "e[3]="
"-1632554850485153"
```

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

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

```
## [1] "dh_3[0]="
"-75246864749.1982"
## [1] "dh_3[1]="
"-49352155691.9911"
## [1] "dh_3[2]="
"87346475826.5242"
```

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

```
## [1] "h_4[0]="
"-75246864749.1982"
## [1] "h_4[1]="
"-49352155691.9911"
## [1] "h_4[2]="
"87346475826.5242"
```

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]="      "141928086"` $g[4] = h_4[0]*x[4-0]+h_4[1]*x[4-1]+h_4[2]*x[4-2]$
`## [1] "g[4]="`
`"-494389010406431"` $e[4] = y[4] - g[4]$

```
## [1] "e[4]="
"-1138165840085739"
```

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

```
## [1] "dh_4[0]="
"-59294805269.1555"
## [1] "dh_4[1]="
"-62638859069.1606"
## [1] "dh_4[2]="
"-41082943926.6816"
```

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

```
## [1] "h_5[0]="
"-134541670018.354"
## [1] "h_5[1]="
"-111991014761.152"
## [1] "h_5[2]="
"46263531899.8426"
```

The complete calculation

```
## $p
##  [1]          0          0  169467339  141928086  191025357  767727645
##  [7] 1051874058 1354795259 1151595275 1061628251 1147034739 1295501739
##
## $g
```

```
##  [1]  0.000000e+00  0.000000e+00  0.000000e+00 -4.943890e+14  7.011246e+14
##  [6]  2.112402e+15  3.036936e+15  5.491161e+15 -2.178582e+15 -7.206436e+14
## [11] -7.886663e+14  2.184338e+15
##
## $e
##  [1]  0.000000e+00  0.000000e+00 -1.632555e+15 -1.138166e+15 -2.333679e+15
##  [6] -3.744957e+15 -4.669491e+15 -7.123716e+15  5.460269e+14 -9.119112e+14
## [11] -8.438886e+14 -3.816893e+15
##
## $amp_dh
## [1] 0 0
##
## $h
##   X3            4             5             6             7            8
## 1  0 -75246864749 -134541670018  -28501624983   94681911535 176394476119
## 2  0 -49352155692 -111991014761 -202320518562 -159979683929 -47876277973
## 3  0  87346475827   46263531900  -49160292037  -85228049096 -46695693982
##             9          10          11          12           13
## 1 74128726327 64245106054 77795423492 61953311557  -9184285298
## 2 48910480032 58132189348 76037505798 87643381234  24201431635
## 3 86088355558 77360708719 60654520202 75990461468 122467810417
```