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 <- 512                ##training data
L <- 16                 ##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, sd =
}
colnames(echo) <- par
echo_sum <- rowSums(echo[,(1:(L + 1))])
data <- data.frame(sig_far,echo,echo_sum)
train_data <- data[1:(n + L),]
```

Lets view the data first:

```
tail(data)
```

```
##      sig_far X.1.75109380645825 X.0.222919974050394 X.1.78216882480844
## 3523   18250         -31957.462          -1719.7002          14145.254
## 3524   15757         -27591.985          -4069.9454         -13733.213
## 3525   27638         -48396.731          -3514.2059         -32522.619
## 3526   -5604           9813.130          -6162.7182         -28079.672
## 3527   -4332           7585.738           1247.5876         -49253.620
## 3528   16073         -28145.331            964.0334           9989.236
##      X0.203799497261961 X.1.09031445450597 X1.45170316207217
## 3523           3615.910         -19148.951          26216.61
```

```
## 3524          -1617.253         -19345.208          25496.57
## 3525           1570.782           8651.887          25757.88
## 3526           3719.440          -8403.902         -11518.96
## 3527           3211.368         -19899.088          11190.03
## 3528           5632.710         -17180.934          26495.34
##      X.1.27829634356687 X.0.196533790531312 X.0.443965271028611
## 3523          -27910.659            1080.914            4250.544
## 3524          -23082.534           -4290.747            2439.610
## 3525          -22448.499           -3548.439           -9694.849
## 3526          -22678.592           -3450.958           -8017.992
## 3527           10145.501           -3486.334           -7797.785
## 3528           -9850.889            1560.260           -7877.699
##      X.0.325920751666866 X0.0235382725213436 X.0.901848885110049
## 3523           -5215.555           -607.4232           15828.736
## 3524            3122.802            374.7347           23192.332
## 3525            1793.371           -227.4684          -14438.215
## 3526           -7114.695           -131.4558            8634.687
## 3527           -5883.693            511.8922            4956.046
## 3528           -5722.036            422.9882          -19693.288
##      X0.00794326195374663 X0.14190530682478 X.1.13193646762845
## 3523             251.51265         -2327.467          -19549.24
## 3524            -138.90662          4481.859           18549.48
## 3525            -203.76335         -2492.929          -35766.50
## 3526             127.67719         -3651.586           19869.31
## 3527             -75.54322          2269.555           29111.57
## 3528             -43.14265         -1360.950          -18119.61
##      X1.65529091335472 X0.236003433810751    echo_sum
## 3523         -17291.14           3801.361  -56536.754
## 3524          28590.21          -2464.530    9913.272
## 3525         -27123.57           4077.013 -158526.853
## 3526          52305.57          -3866.390   -8607.106
## 3527         -29053.64           7458.234  -37762.169
## 3528         -42569.09          -4141.570 -109639.964
```

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)
```

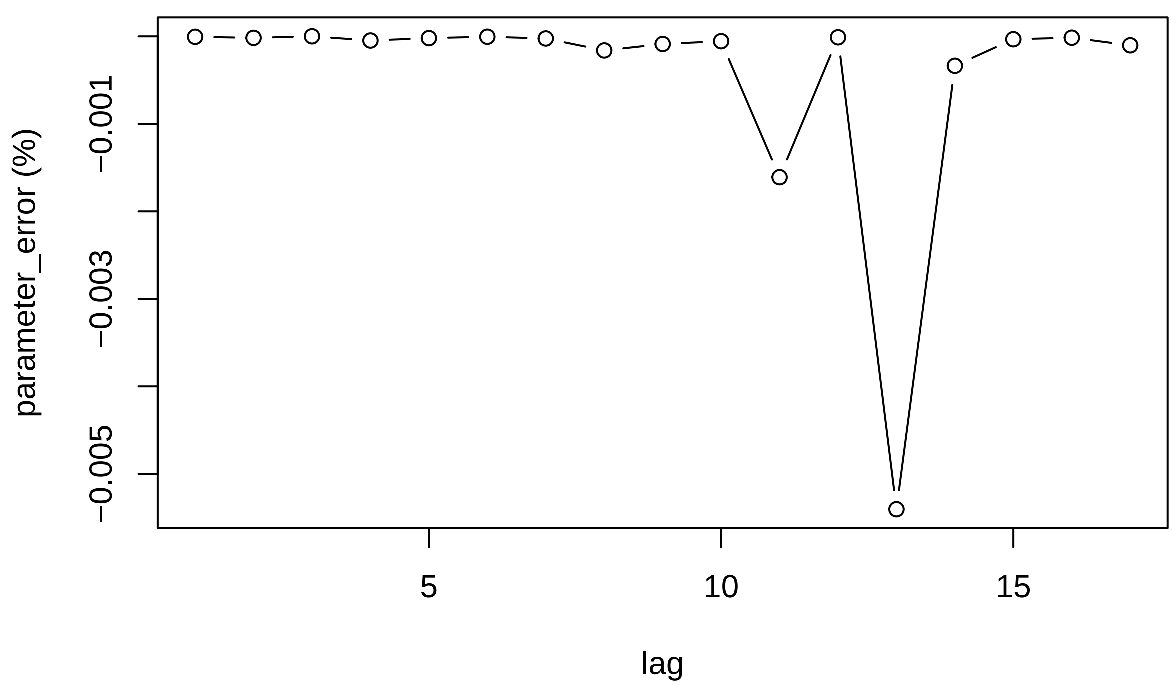
```
e <- rep(0, L)
amp_h <- rep(0,L)
for (i in c((L + 1):(n + L))) {
  p[i] <- sum(x[(i):(i - L)] * x[(i):(i - L)])
  g[i] <- sum(h * x[(i):(i - L)])
  e[i] <- y[i] - g[i]
  dh <- (1 * mu / (gamma + p[i])) * e[i] * x[(i):(i - L)]
  h <- h + dh
  amp_h[i] <- sum(dh * dh)
}
sol <- list("p" = p, "g" = g,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

```
##  [1] -1.751102273 -0.222923785 -1.782167801  0.203789838 -1.090336381
##  [6]  1.451696992 -1.278327006 -0.196565419 -0.444003964 -0.325938842
## [11]  0.023500390 -0.901858677  0.007900338  0.141857614 -1.131973540
## [16]  1.655265641  0.235979277
```

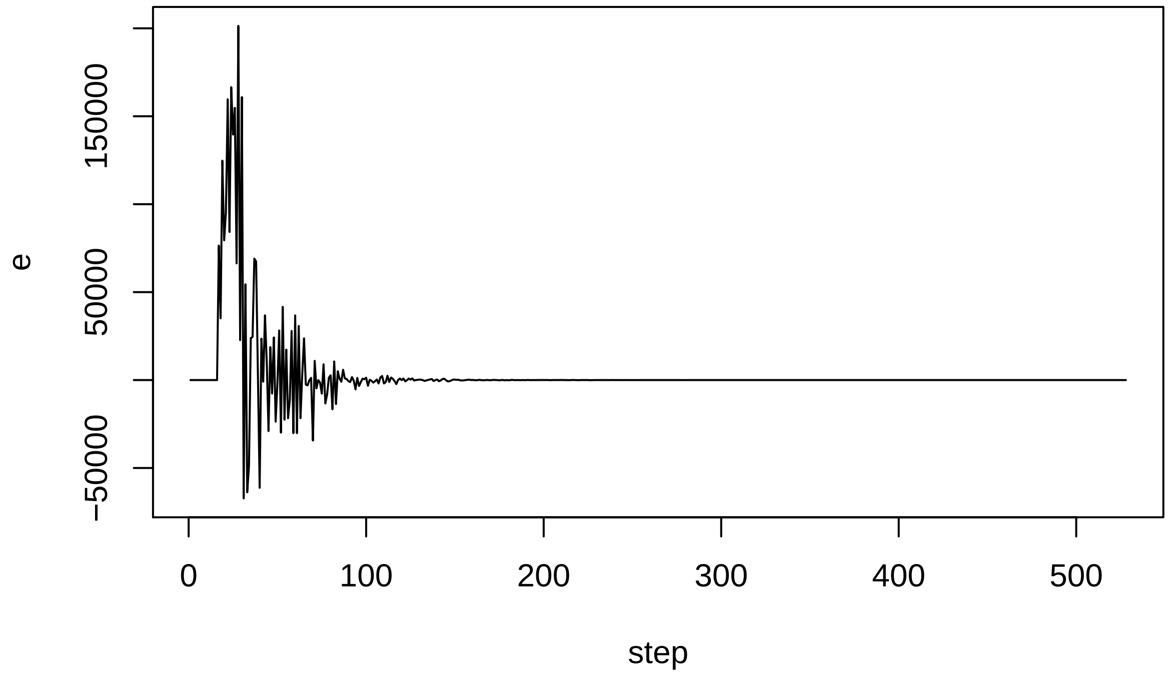
```
par
```

```
##  [1] -1.751093806 -0.222919974 -1.782168825  0.203799497 -1.090314455
##  [6]  1.451703162 -1.278296344 -0.196533791 -0.443965271 -0.325920752
## [11]  0.023538273 -0.901848885  0.007943262  0.141905307 -1.131936468
## [16]  1.655290913  0.236003434
```

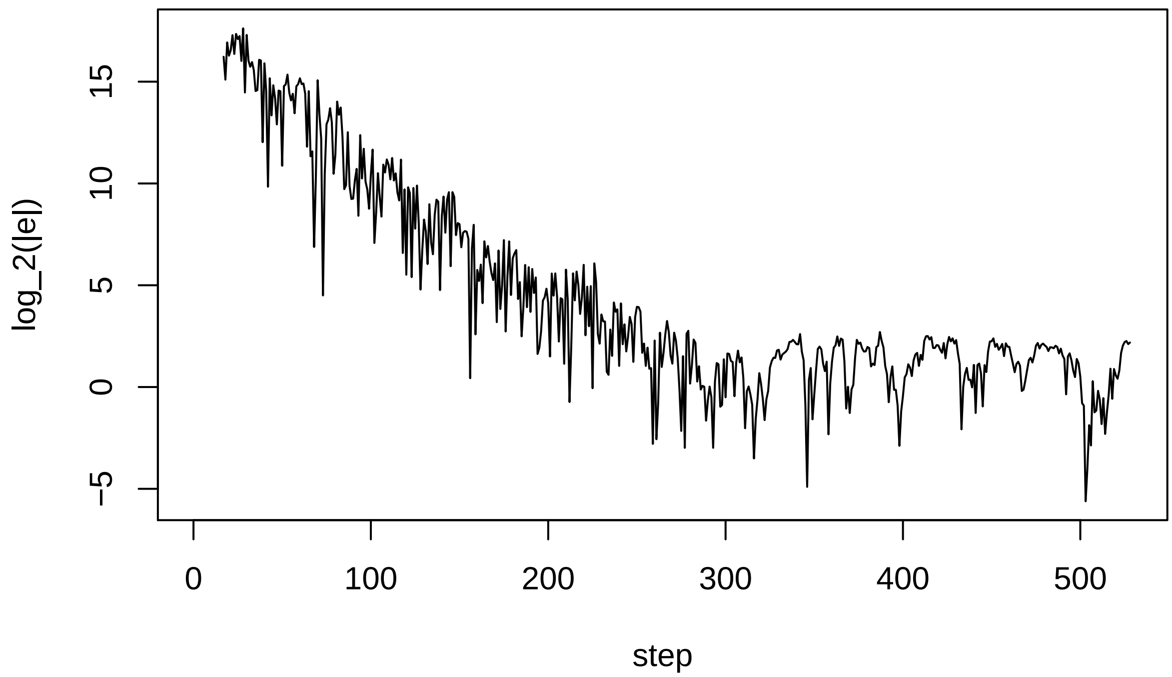
```
plot((sol$h-par) / abs(par), type = "b", ylab = "parameter_error (%)",xlab = "lag")
```



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

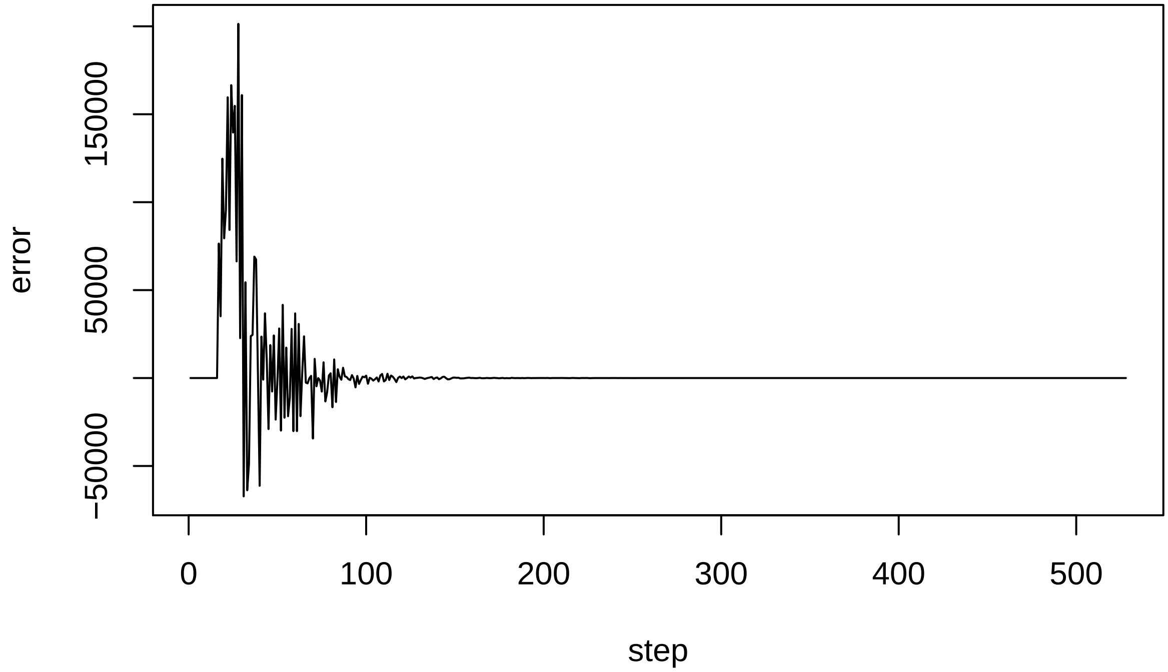



```r
plot(log(abs(e),base = 2),type = "l", ylab = "log_2(|e|)", xlab = "step")
```

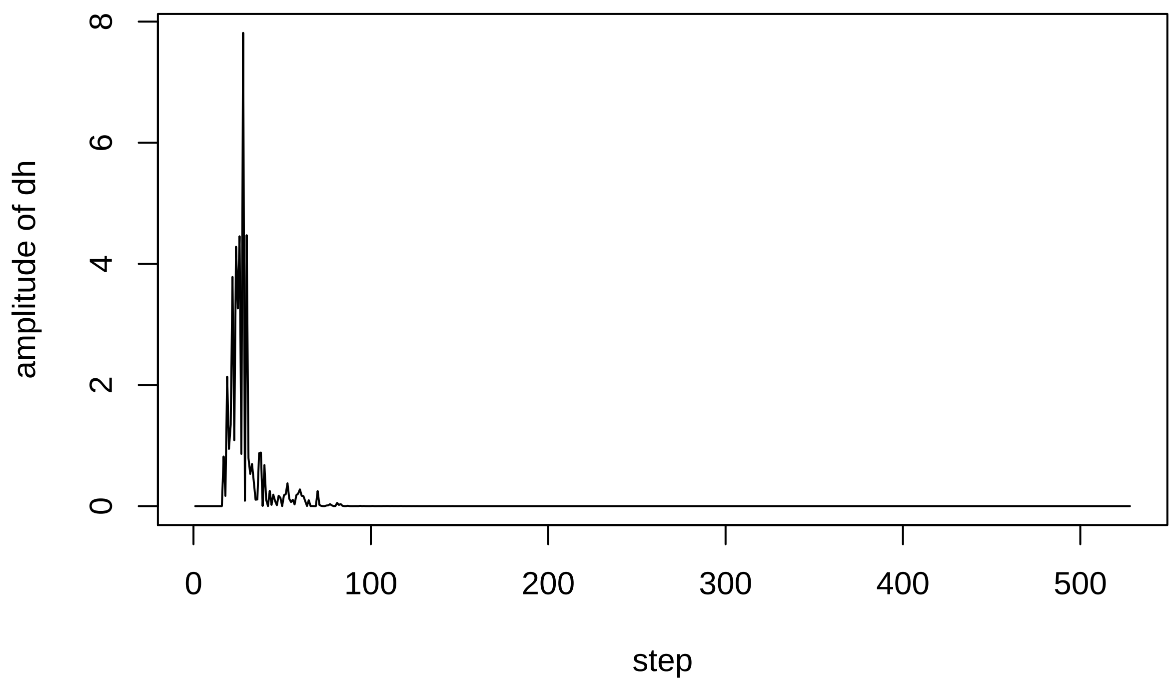


We can see “sol$h” predict “par” very actuarily. That means our training is pretty succesful.

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




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



We can see the error of predicton converge to 0.

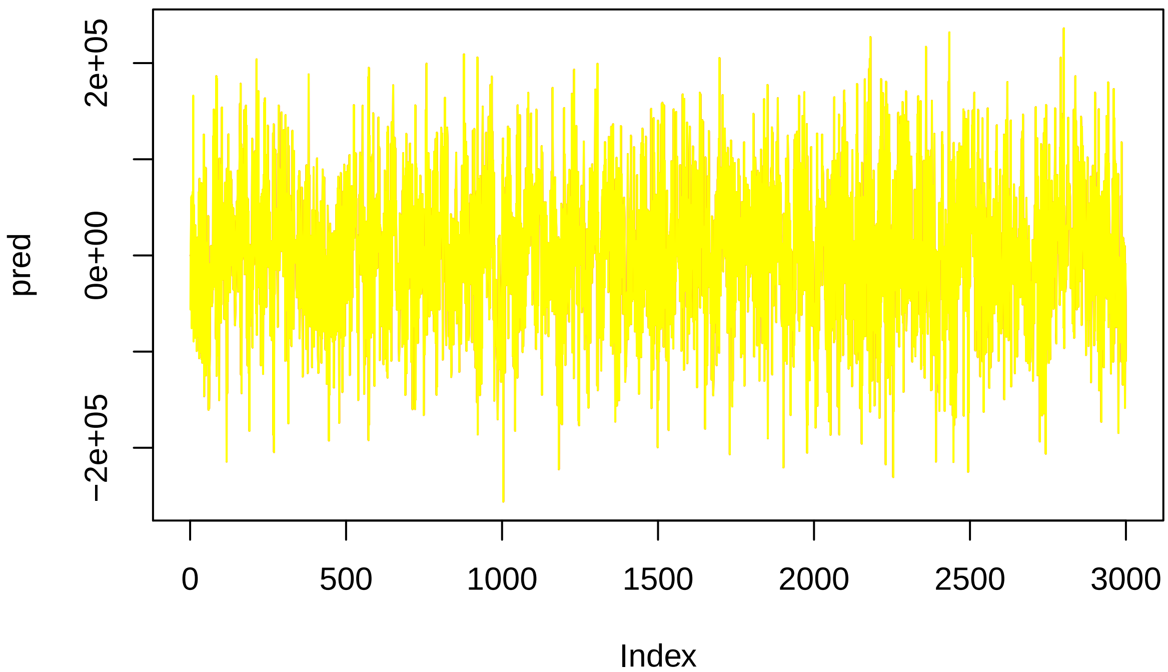
also the amplitude of parameter correction also converge to 0.

We we can test our result

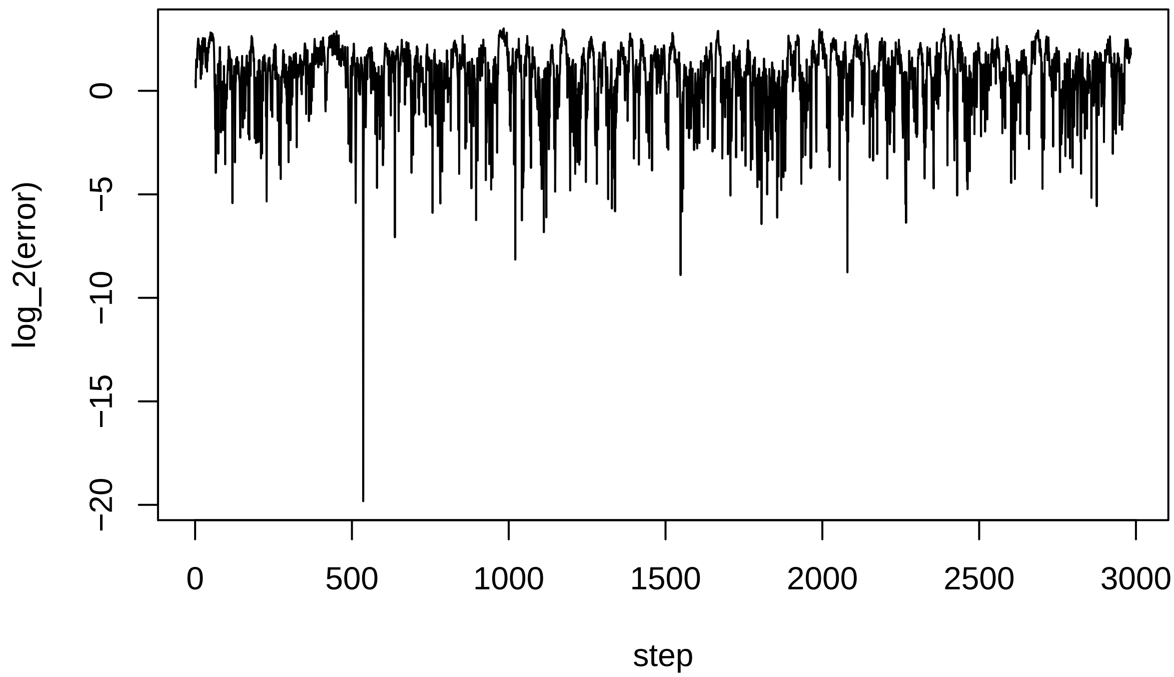
```
test_data <- data[-(1:(n + L)),]
x <- test_data[,1]
y <- test_data[,length(test_data[1,])]

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)")
```




```r
sd(error)
```

```
## [1] 2.236632
```

```r
mean(error)
```

```
## [1] -1.660712
```

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

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

```
##
## FALSE  TRUE
##  1465  1519
```

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

```
##
## FALSE  TRUE
##   455  2529
```

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

```
##
## FALSE  TRUE
##     1  2983
```

In 16 bit nonideal data. With 512 samples, we are expecting:

30% of the predictions are offed by 1 digit 70% 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] -27554  13797  21811  17660  20411   1935   1406  30321 -24387   5511
## [11]  -7178 -22257
```

```
print(y)
```

```
##  [1]            NA            NA -2.582508e+13 -2.582508e+13 -2.582508e+13
##  [6] -2.582508e+13 -2.582508e+13 -2.582508e+13 -2.582508e+13 -2.582508e+13
## [11] -2.582508e+13 -2.582508e+13
```

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]="      "1425299846"`

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

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

`## [1] "e[3]=" "-25825078594751.9"`

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

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

```
## [1] "dh_3[0]="
"-395194590.675718"
## [1] "dh_3[1]="
"-249988527.236389"
## [1] "dh_3[2]="
"499252292.489053"
```

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

```
## [1] "h_4[0]="
"-395194590.675718"
## [1] "h_4[1]="
"-249988527.236389"
## [1] "h_4[2]="
"499252292.489053"
```

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]="      "977952530"` $g[4] = h_4[0]*x[4-0]+h_4[1]*x[4-1]+h_4[2]*x[4-2]$
`## [1] "g[4]="`
`"-5543452359414.6"` $e[4] = y[4] - g[4]$

```
## [1] "e[4]="
"-20281626204496.7"
```

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

```
## [1] "dh_4[0]="
"-366248368.692713"
## [1] "dh_4[1]="
"-452335400.314653"
## [1] "dh_4[2]="
"-286134130.399398"
```

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

```
## [1] "h_5[0]="
"-761442959.368431"
## [1] "h_5[1]="
"-702323927.551042"
## [1] "h_5[2]="
"213118162.089655"
```

The complete calculation

```
## $p
##  [1]          0          0 1425299846  977952530 1204204242  732228746
##  [7]  422329982  925084102 1516065646 1544459931  676620574  577268854
##
## $g
```

```
##  [1]  0.000000e+00  0.000000e+00  0.000000e+00 -5.543452e+12 -2.329653e+13
##  [6] -1.369346e+13 -5.817960e+12 -3.113624e+13 -1.895257e+13 -4.711385e+12
## [11]  3.628849e+13  1.539320e+13
## 
## $e
##  [1]  0.000000e+00  0.000000e+00 -2.582508e+13 -2.028163e+13 -2.528546e+12
##  [6] -1.213162e+13 -2.000712e+13  5.311165e+12 -6.872513e+12 -2.111369e+13
## [11] -6.211357e+13 -4.121828e+13
## 
## $amp_dh
## [1] 0 0
## 
## $h
##   X3          4          5          6           7           8           9
## 1  0 -395194591 -761442959 -804301264  -836360484  -902967189  -728885890
## 2  0 -249988527 -702323928 -739405778 -1077576700 -1169243823 -1161171586
## 3  0  499252292  213118162  167320186  -125271969 -1092206156 -1081096782
##           10          11          12         13
## 1  -618336605  -693675279   -34736985 1554462267
## 2 -1298620426  -965235516 -1471143698 -958618550
## 3 -1087470354 -1501976602   736742842  343245197
```