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

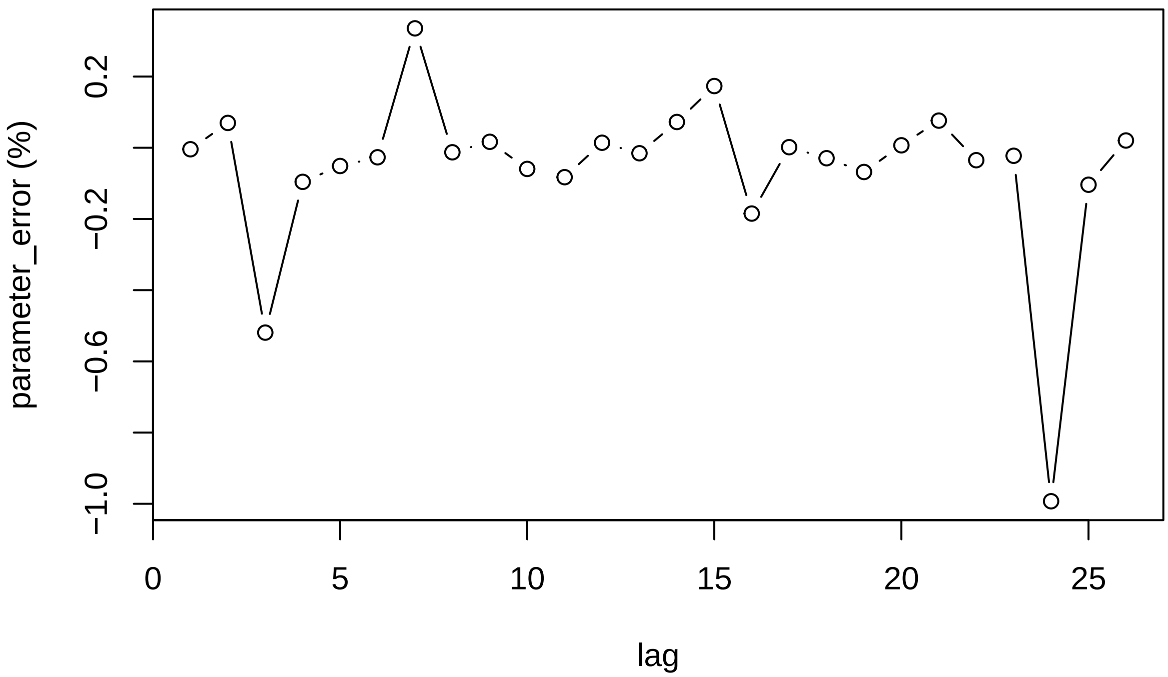
```r
x <- train_data[,1]
y <- train_data[,length(train_data[1,])]

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] -0.4099105348  0.5227149868  0.0301696986  0.9773686611  1.2558462415
##  [6] -0.8474031116  0.1971413597 -0.3108016281  3.0354760797  0.5449812995
## [11]  0.4900135417 -1.1027027307 -0.5465820826  1.0027257630  0.3928007292
## [16]  0.3527398398 -1.5030191363 -0.9042114360  0.5869378846 -0.5257007130
## [21] -0.4716530619  0.3855780454  1.8123371741  0.0005059254  0.5327017985
## [26]  0.6048158202
```

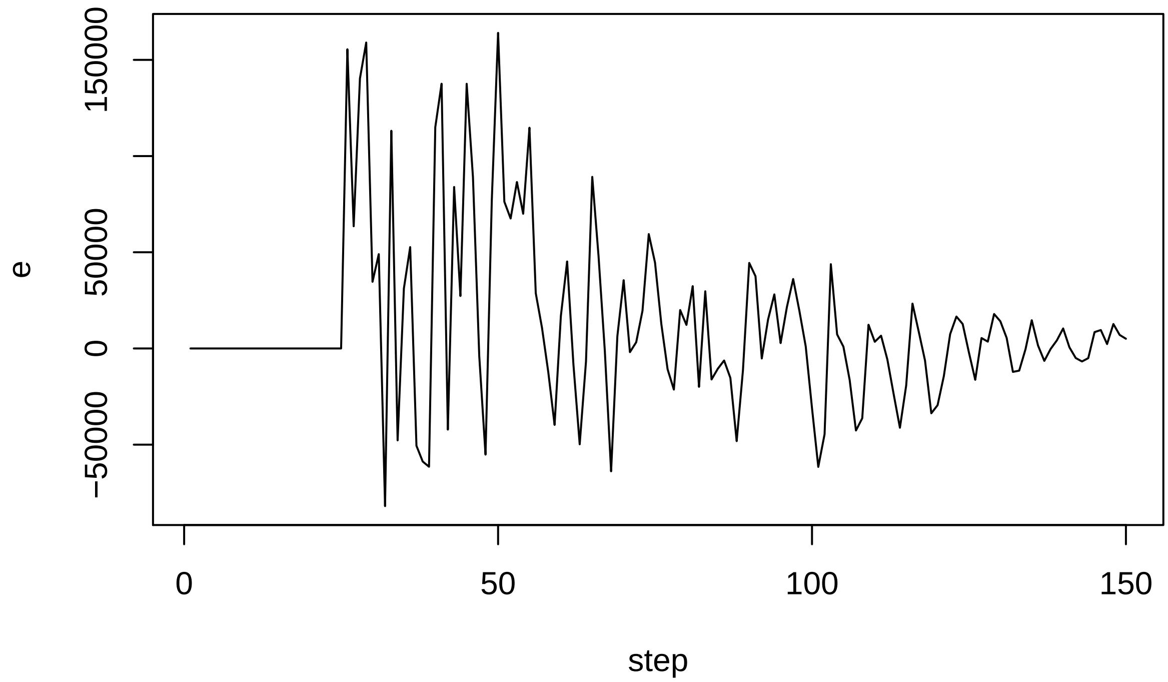
```r
par
```

```
##  [1] -0.40820271  0.48862425  0.06274898  1.08087518  1.32362569
##  [6] -0.82532949  0.14762341 -0.30690658  2.98558090  0.57944065
## [11]  0.53412805 -1.11846092 -0.53829116  0.93506493  0.33477489
## [16]  0.43273156 -1.50559510 -0.87847032  0.62979571 -0.52937446
## [21] -0.51061752  0.39950139  1.85403004  0.07116088  0.59443489
## [26]  0.59273107
```

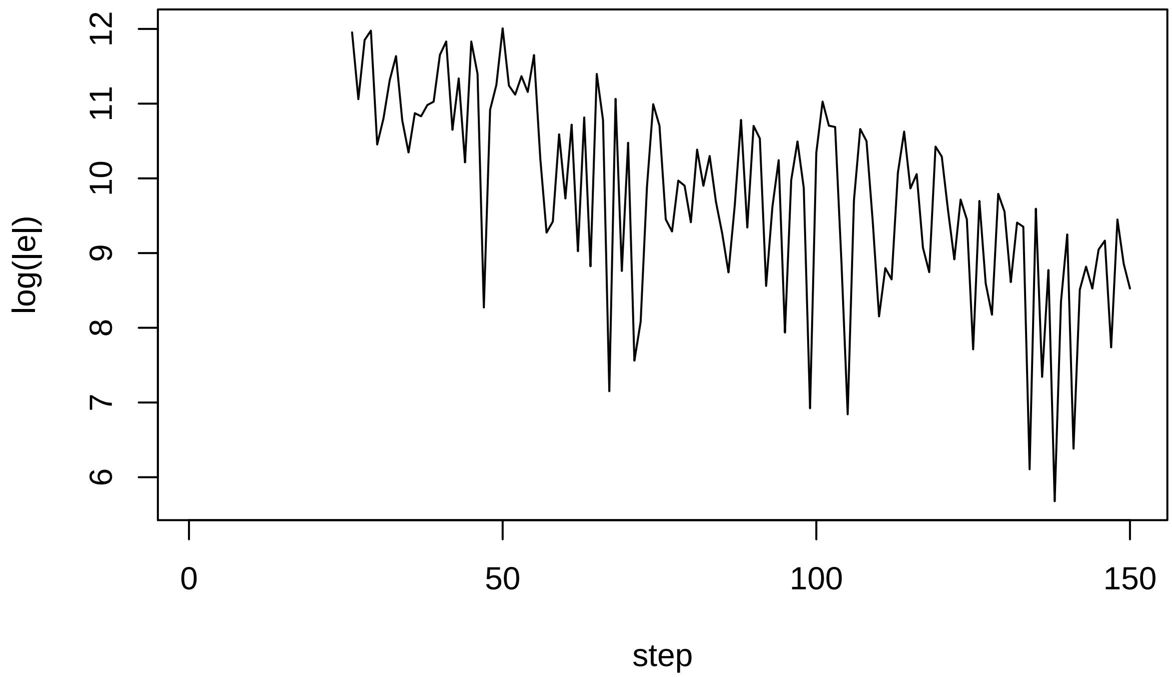
```r
plot((sol$h-par) / abs(par), type = "b", ylab = "parameter_error (%)",xlab = "lag")
```




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

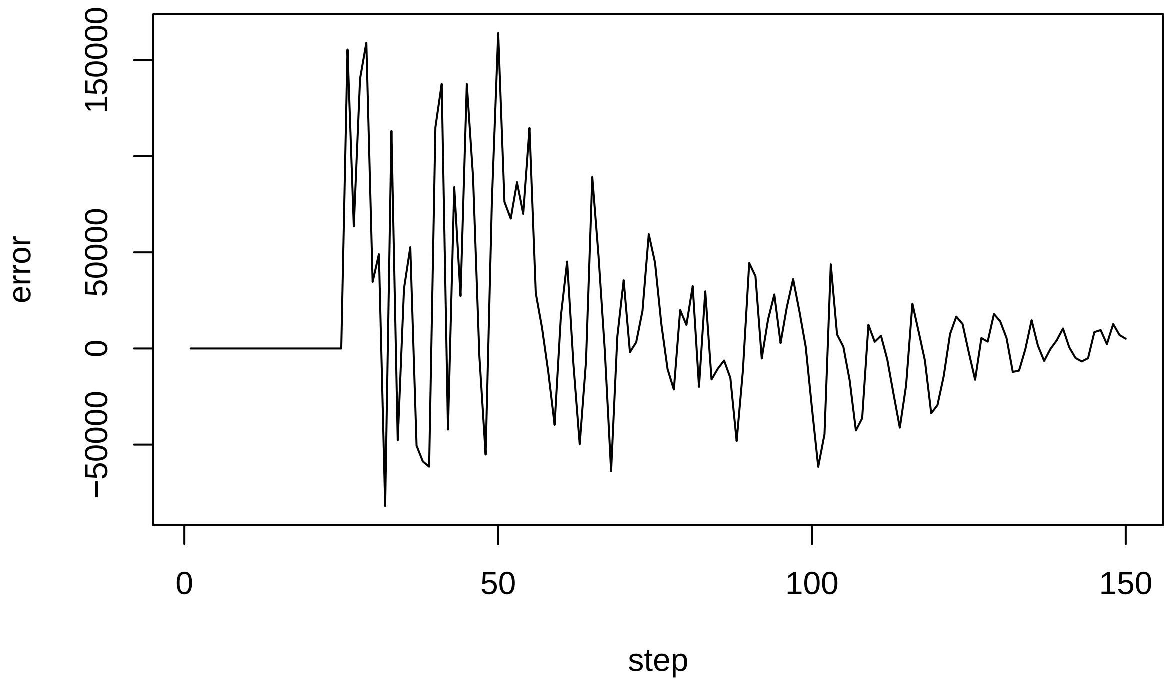



```
plot(log(abs(e)),type = "l", ylab = "log(|e|)", xlab = "step")
```

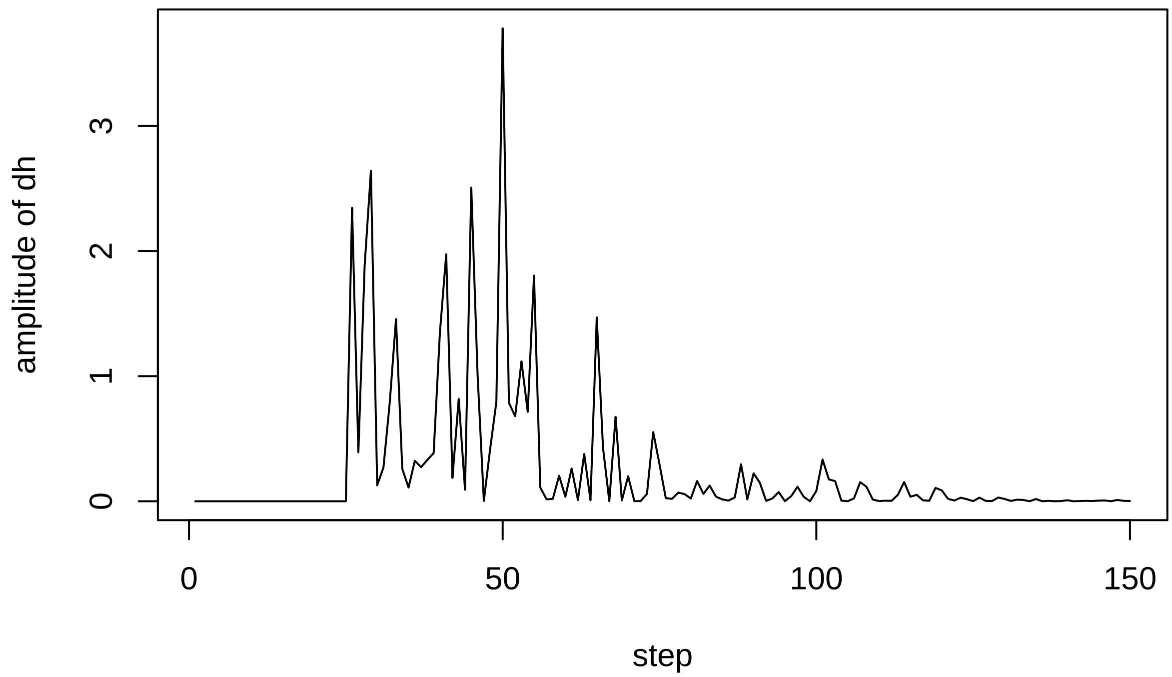


We can see “sol$h” predict “par” very actuarily. That means our training is pretty succesful.

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




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

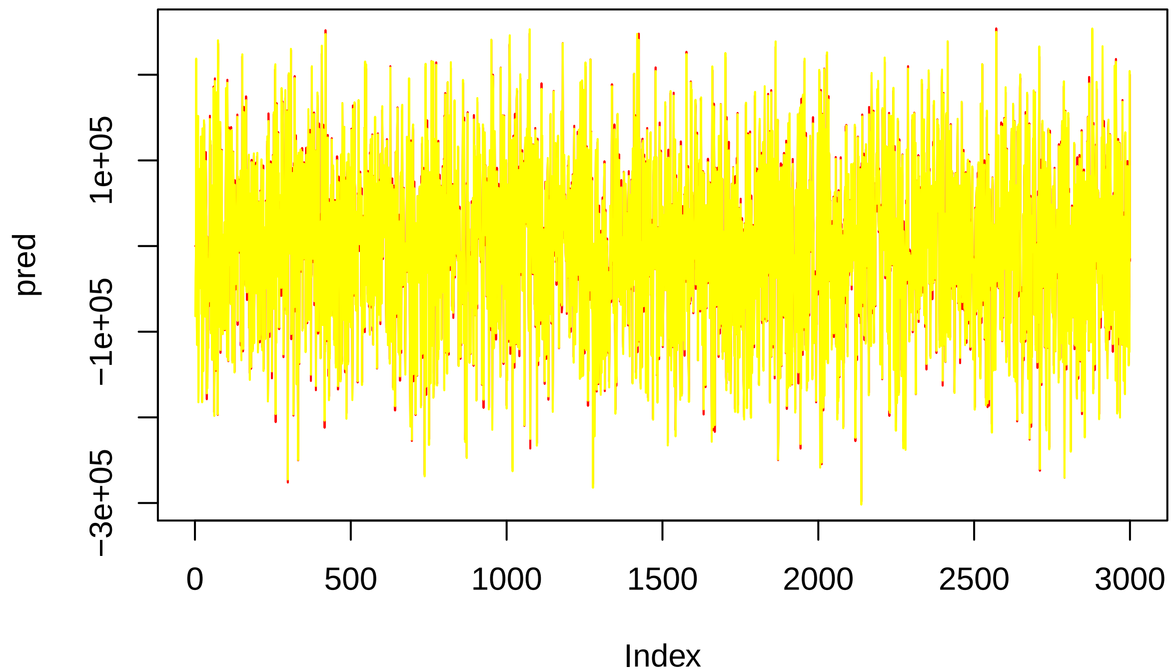


We can see the error of predicton converge to 0.

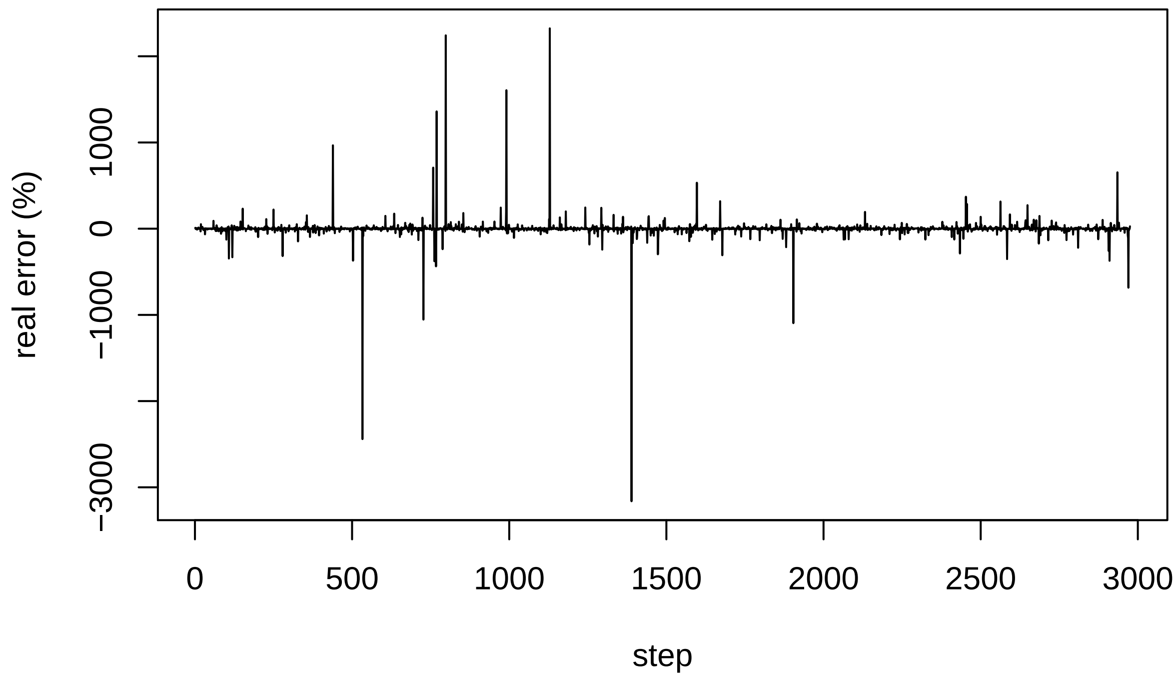
also the amplitude of parameter correction also converge to 0.

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")
```




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




```r
sd(error)
```

```
## [1] 115.8282
```

```r
mean(error)
```

```
## [1] -0.5439006
```

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

```
##
## FALSE  TRUE
##  1339  1636
```

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

```r
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]   9395 -31532  24242 -27310   3503  31390 -30718  23788 -31152 -29898
## [11]  26036  -7407
```

```r
print(y)
```

```
##  [1]            NA            NA -1.535849e+15 -1.535849e+15 -1.535849e+15
##  [6] -1.535849e+15 -1.535849e+15 -1.535849e+15 -1.535849e+15 -1.535849e+15
## [11] -1.535849e+15 -1.535849e+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]="      "1670207613"
```

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

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

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

```
## [1] "e[3]=" "-1535848878866804"
```

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

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

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

```
## [1] "dh_3[0]=" "-22291868526.8195"
## [1] "dh_3[1]=" "28995429353.5052"
## [1] "dh_3[2]=" "-8639225509.83703"
```

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

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

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

```
## [1] "h_4[0]=" "-22291868526.8195"
```

```
## [1] "h_4[1]="
"28995429353.5052"
## [1] "h_4[2]="
"-8639225509.83703"
```

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]="       "2327777688"
```
$g[4] =$
$h_4[0]*x[4-0]+h_4[1]*x[4-1]+h_4[2]*x[4-2]$
```
## [1] "g[4]="
"1584110186631295"
```
$e[4] = y[4] - g[4]$
```
## [1] "e[4]="
"-3119959065661256"
```
$\Delta h_4[0] = 1 * a/p[4] * e[4] * x[4-0]$
$\Delta h_4[1] = 1 * a/p[4] * e[4] * x[4-1]$
$\Delta h_4[2] = 1 * a/p[4] * e[4] * x[4-2]$
```
## [1] "dh_4[0]="
"36604046220.7613"
## [1] "dh_4[1]="
"-32491954906.0306"
## [1] "dh_4[2]="
"42262862886.6"
```
$h_5[1] = h_4[0] + \Delta h_4[0]$
$h_5[1] = h_4[1] + \Delta h_4[1]$
$h_5[2] = h_4[2] + \Delta h_4[2]$
```
## [1] "h_5[0]="
"14312177693.9418"
## [1] "h_5[1]="
"-3496525552.52541"
## [1] "h_5[2]="
"33623637376.763"
```

The complete calculation

```
## $p
##  [1]          0          0 1670207613 2327777688 1345781673 1743439209
##  [7] 1941198633 2494796568 2479911572 2430206452 2542210804 1626627349
##
## $g
##  [1]  0.000000e+00  0.000000e+00  0.000000e+00  1.584110e+15  9.607299e+14
##  [6]  7.204122e+14  2.430154e+15  1.901385e+15  1.394596e+15 -5.539207e+14
## [11]  1.032284e+15 -1.712019e+14
##
## $e
##  [1]  0.000000e+00  0.000000e+00 -1.535849e+15 -3.119959e+15 -2.496579e+15
##  [6] -2.256261e+15 -3.966003e+15 -3.437234e+15 -2.930445e+15 -9.819282e+14
## [11] -2.568133e+15 -1.364647e+15
##
## $amp_dh
## [1] 0 0
##
## $h
```

```
##   X3           4           5            6            7            8
## 1  0 -22291868527 14312177694   7813712454 -32809462601  29949527367
## 2  0  28995429354 -3496525553  47166644787  42633258925 -21498673333
## 3  0  -8639225510 33623637377 -11348042424  23995025775  16838155475
##             9          10          11          12          13
## 1  -2824658637 33986824669 46067151623 19765672018 25979719675
## 2  20823398537 -7286243376  5300763958 35503622970 13660914074
## 3 -26409771995  9888865145   277291876 31746937469 56829642592
```