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

Lets view the data first:

```
tail(data)
```

```
##      sig_far X.2.2183141328483 X.1.37516468567918 X1.89451585575662
## 3062  -26292         58323.915         -28795.949        -18643.931
## 3063    3474         -7706.423          36155.830         39671.162
## 3064   23798        -52791.440          -4777.322        -49810.611
## 3065   11040        -24490.188         -32726.169          6581.548
## 3066    1905         -4225.888         -15181.818         45085.688
## 3067  -30197         66986.432          -2619.689         20915.455
##      X.1.15342654420583   echo_sum
## 3062           7808.698   18692.73
```

```
## 3063          11350.871   79471.44
## 3064         -24152.752 -131532.12
## 3065          30325.891  -20308.92
## 3066          -4007.004   21670.98
## 3067         -27449.245   57832.95
```

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

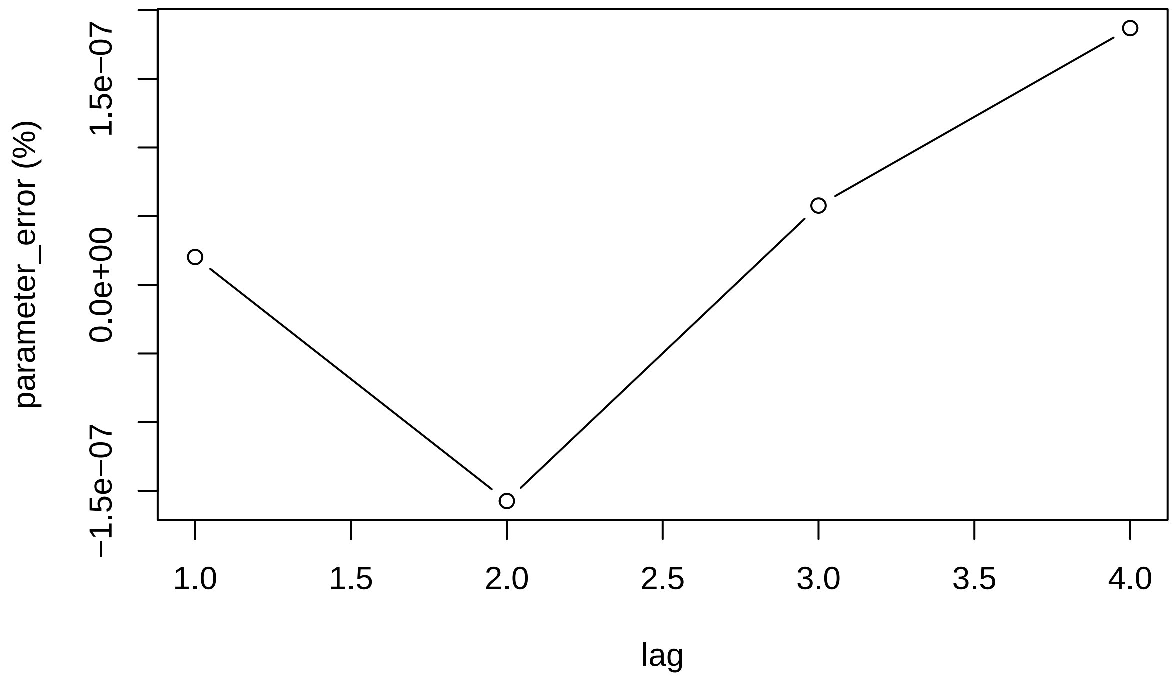
```r
x <- train_data[,1]
y <- train_data[,length(train_data[1,])]

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] -2.218314 -1.375165  1.894516 -1.153426
```

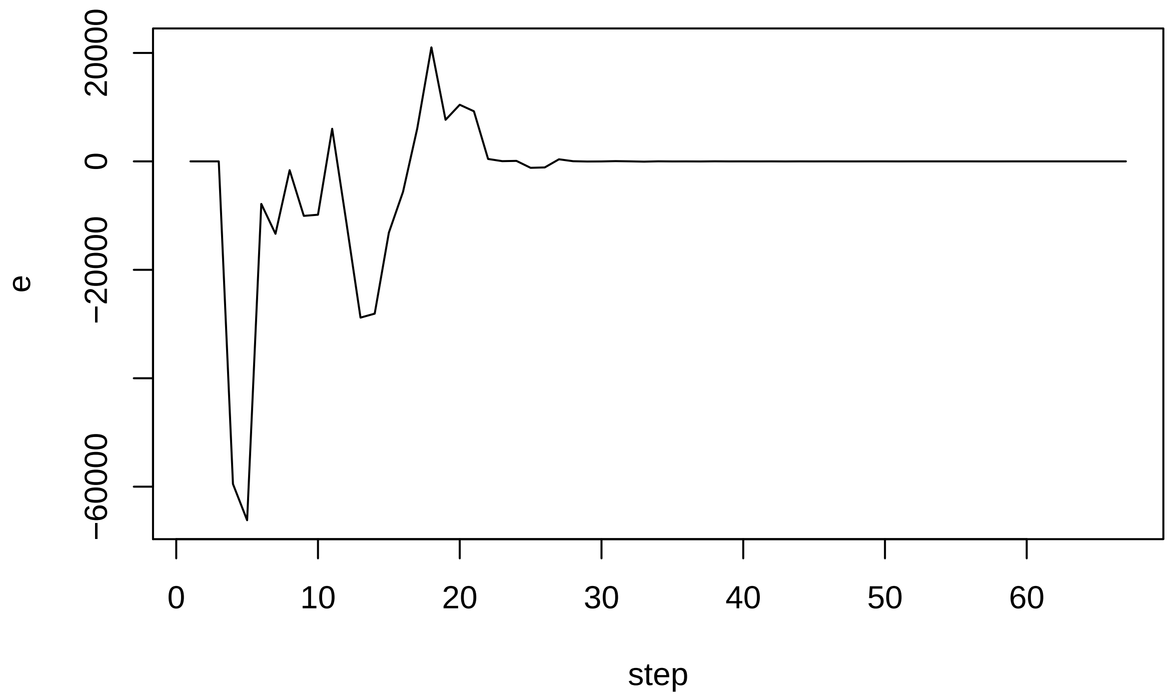
```r
par
```

```
## [1] -2.218314 -1.375165  1.894516 -1.153427
```

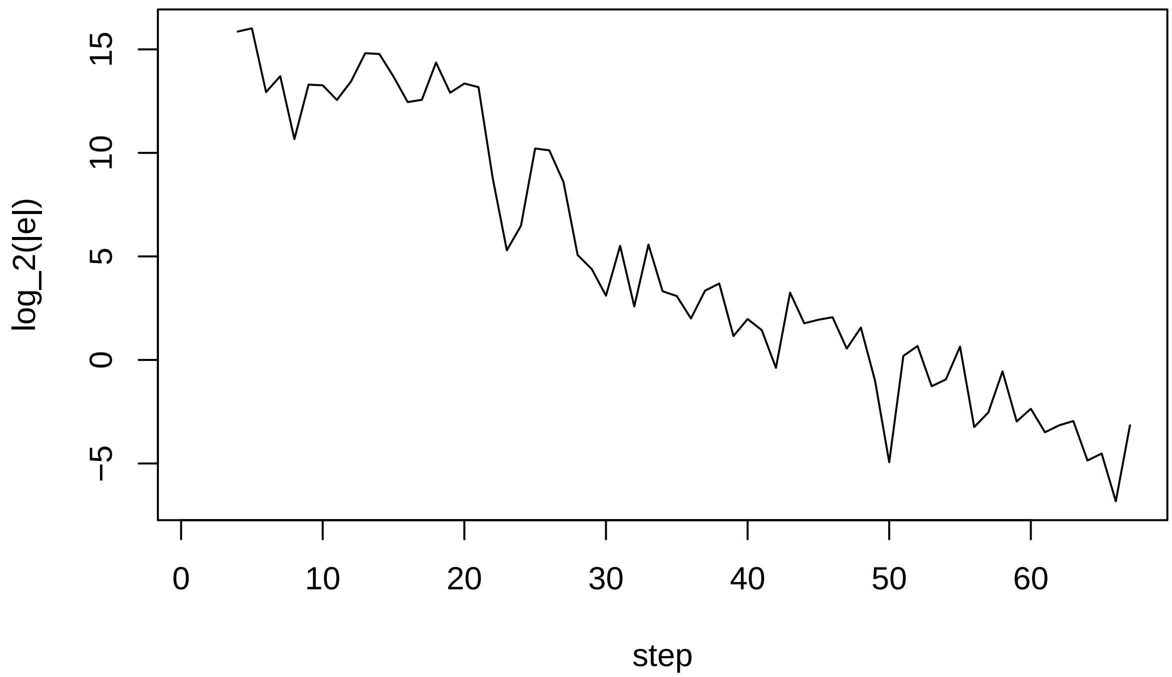
```r
plot((sol$h-par) / abs(par), type = "b", ylab = "parameter_error (%)",xlab = "lag")
```




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

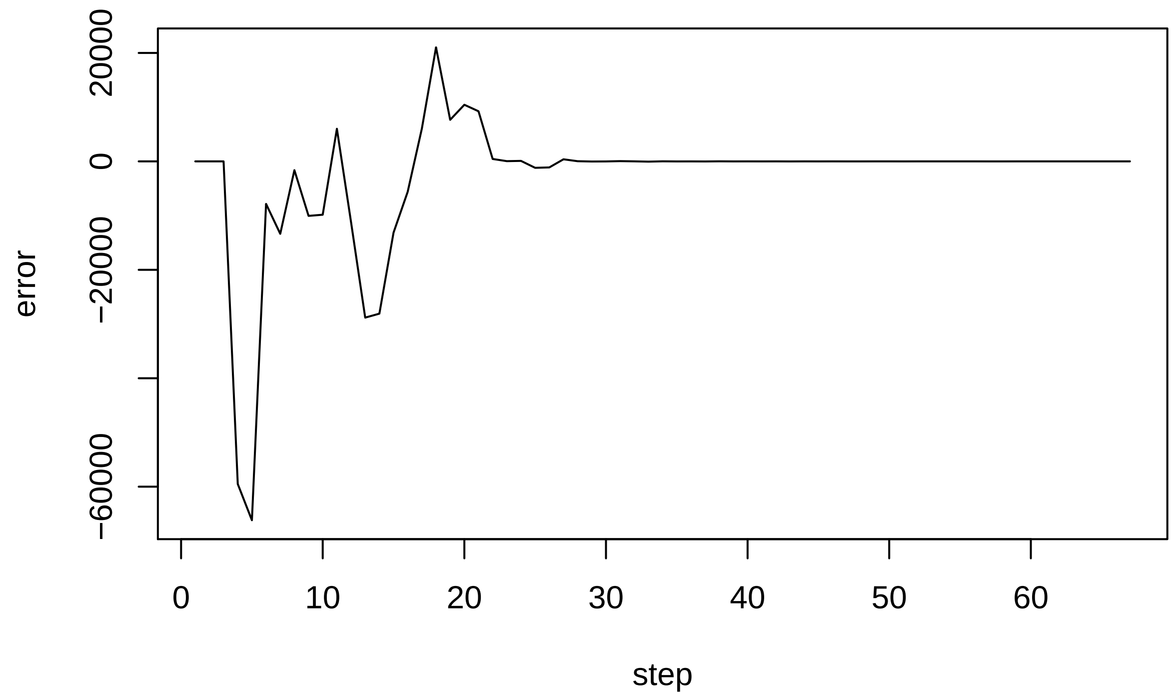



```
plot(log(abs(e),base = 2),type = "l", ylab = "log_2(|e|)", xlab = "step")
```

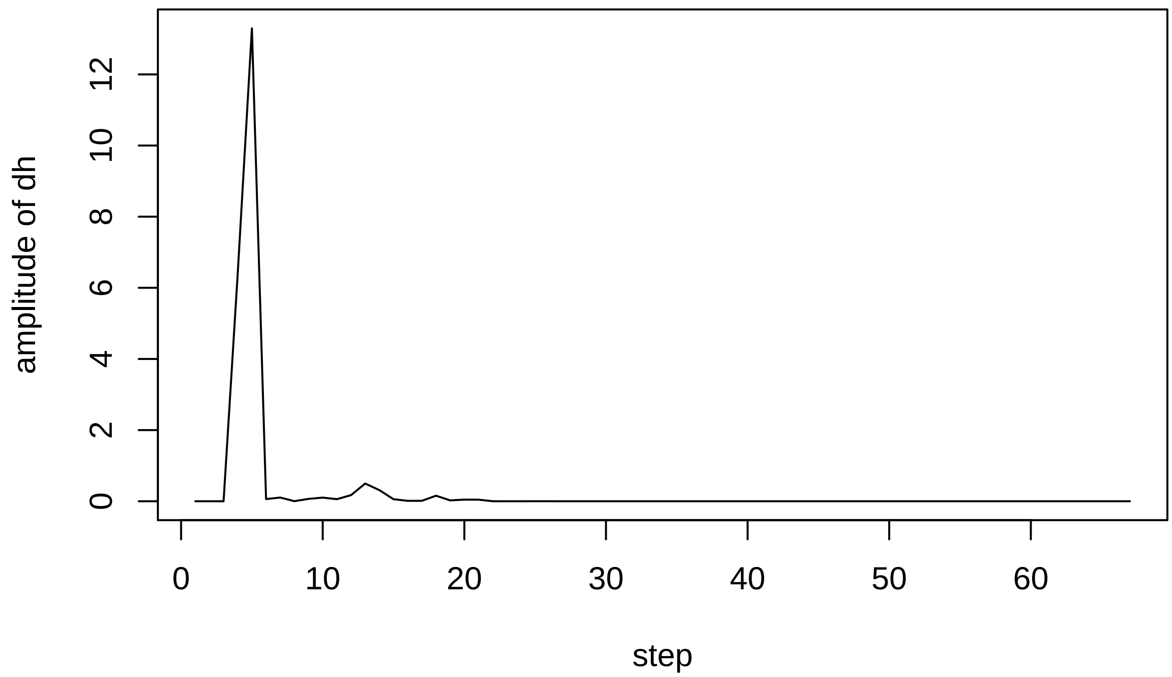


We can see "sol$h" predict "par" very actuarily. That means our training is pretty succesful.

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




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

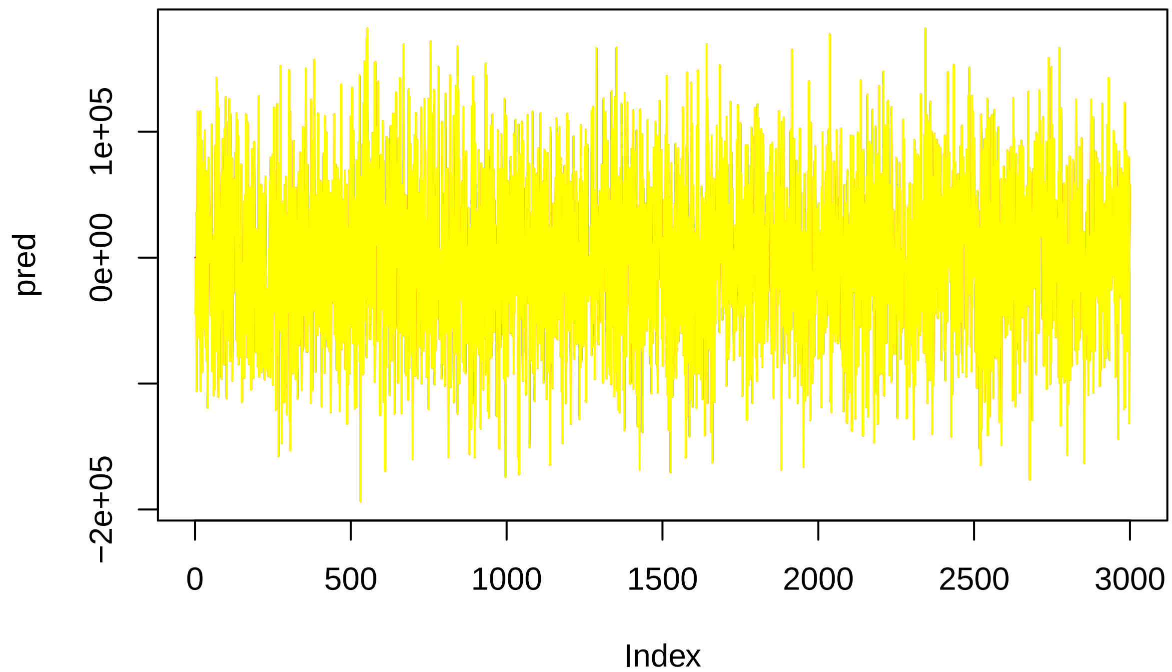


We can see the error of predicton converge to 0.

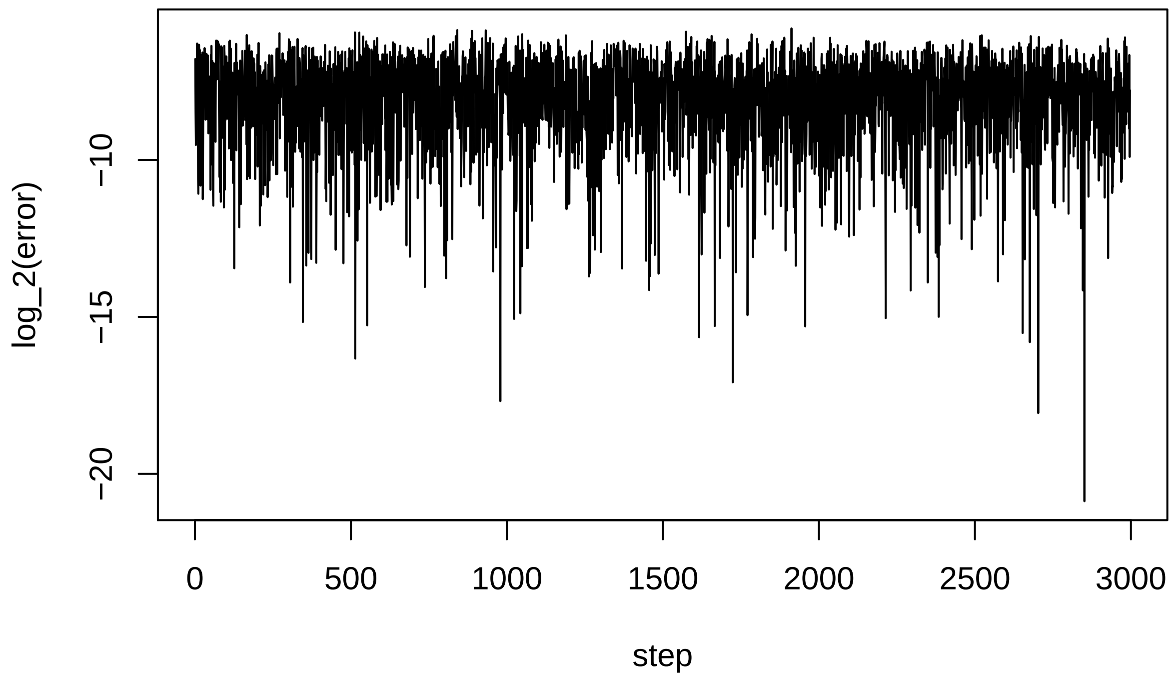
also the amplitude of parameter correction also converge to 0.

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




```r
sd(error)
```

```
## [1] 0.006202468
```

```r
mean(error)
```

```
## [1] 2.090453e-05
```

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

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

```
##
## TRUE
## 2997
```

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

```
##
## TRUE
## 2997
```

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

```
##
## TRUE
## 2997
```

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]   8946  13320 -10712 -28941 -20315 -14251  14586 -17096  25402  -2950
## [11]   -596 -24685
```

```
print(y)
```

```
##  [1]            NA            NA -9.924005e+14 -9.924005e+14 -9.924005e+14
##  [6] -9.924005e+14 -9.924005e+14 -9.924005e+14 -9.924005e+14 -9.924005e+14
## [11] -9.924005e+14 -9.924005e+14
```

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]="      "372200260"`
$g[3] = h_3[0]*x[3-0]+h_3[1]*x[3-1]+h_3[2]*x[3-2]$
`## [1] "g[3]=" "0"` $e[3] = y[3] - g[3]$
`## [1] "e[3]=" "-992400514584278"`
$\Delta h_3[0] = 1 * a/p[3] * e[3] * x[3-0]$

$\Delta h_3[1] = 1 * a/p[3] * e[3] * x[3-1]$
$\Delta h_3[2] = 1 * a/p[3] * e[3] * x[3-2]$
[1] "dh_3[0]="
"28561490828.1547"
[1] "dh_3[1]="
"-35515221978.2506"
[1] "dh_3[2]="
"-23852790977.2845"
$h_4[1] = h_3[0] + \Delta h_3[0]$
$h_4[1] = h_3[1] + \Delta h_3[1]$
$h_4[2] = h_3[2] + \Delta h_3[2]$
[1] "h_4[0]="
"28561490828.1547"
[1] "h_4[1]="
"-35515221978.2506"
[1] "h_4[2]="
"-23852790977.2845"

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]=" "1129750825"
$g[4] =$
$h_4[0]*x[4-0]+h_4[1]*x[4-1]+h_4[2]*x[4-2]$
[1] "g[4]="
"-763878224044034" $e[4] = y[4] - g[4]$
[1] "e[4]="
"-228522290524534"
$\Delta h_4[0] = 1 * a/p[4] * e[4] * x[4-0]$
$\Delta h_4[1] = 1 * a/p[4] * e[4] * x[4-1]$
$\Delta h_4[2] = 1 * a/p[4] * e[4] * x[4-2]$
[1] "dh_4[0]="
"5854090533.69847"
[1] "dh_4[1]="
"2166788217.30341"
[1] "dh_4[2]="
"-2694325901.27721"
$h_5[1] = h_4[0] + \Delta h_4[0]$
$h_5[1] = h_4[1] + \Delta h_4[1]$
$h_5[2] = h_4[2] + \Delta h_4[2]$
[1] "h_5[0]="
"34415581361.8531"
[1] "h_5[1]="
"-33348433760.9472"
[1] "h_5[2]="
"-26547116878.5617"

The complete calculation

```
## $p
##  [1]          0          0  372200260 1129750825 1365027650 1453371707
##  [7]  828541622  708115613 1150286216  946237320  654319320  618406941
##
```

```
## $g
##  [1]  0.000000e+00  0.000000e+00  0.000000e+00 -7.638782e+14  5.503572e+14
##  [6] -3.867549e+14  8.603215e+14 -5.593070e+14  1.274330e+15  1.270162e+15
## [11]  1.613269e+15  8.773696e+13
##
## $e
##  [1]  0.000000e+00  0.000000e+00 -9.924005e+14 -2.285223e+14 -1.542758e+15
##  [6] -6.056456e+14 -1.852722e+15 -4.330936e+14 -2.266731e+15 -2.262562e+15
## [11] -2.605670e+15 -1.080137e+15
##
## $amp_dh
## [1] 0 0
##
## $h
##   X3            4            5            6           7           8
## 1  0  28561490828  34415581362  57375645955 63314289111 30698180539
## 2  0 -35515221978 -33348433761   -639242060  7826376893 39693383834
## 3  0 -23852790977 -26547116879 -14440387274 -2380161917 43046702585
##             9          10          11           12           13
## 1 41154336758 -8902332217 -1848542783    524884186  43640819462
## 2 30772379236 64461412100  3722307284  15469974327  16510974830
## 3 51762816429 23019938952 63898442059 -37258926831 -32106323669
```