Gradient_Descent_and_Back_Propogation.R

kriti

Sat Nov 03 21:49:54 2018

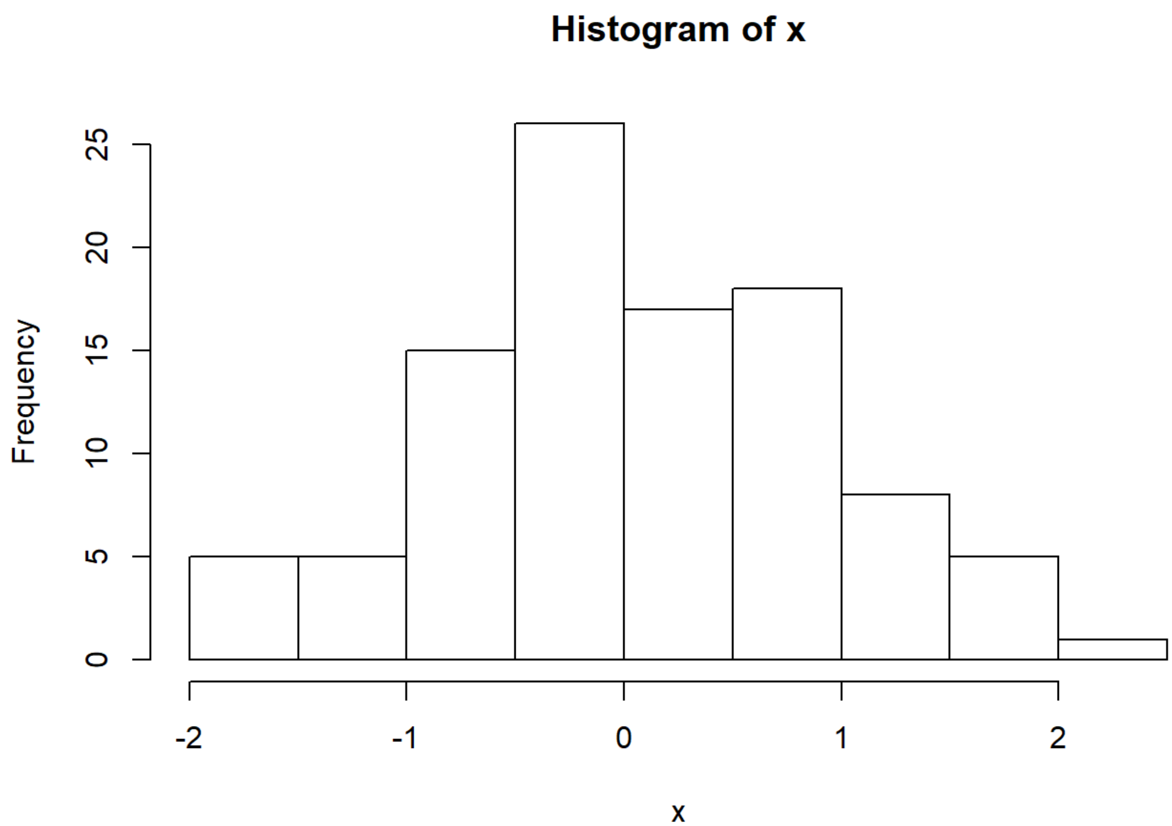
```
###understanding cost function and back propogation of errors

library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

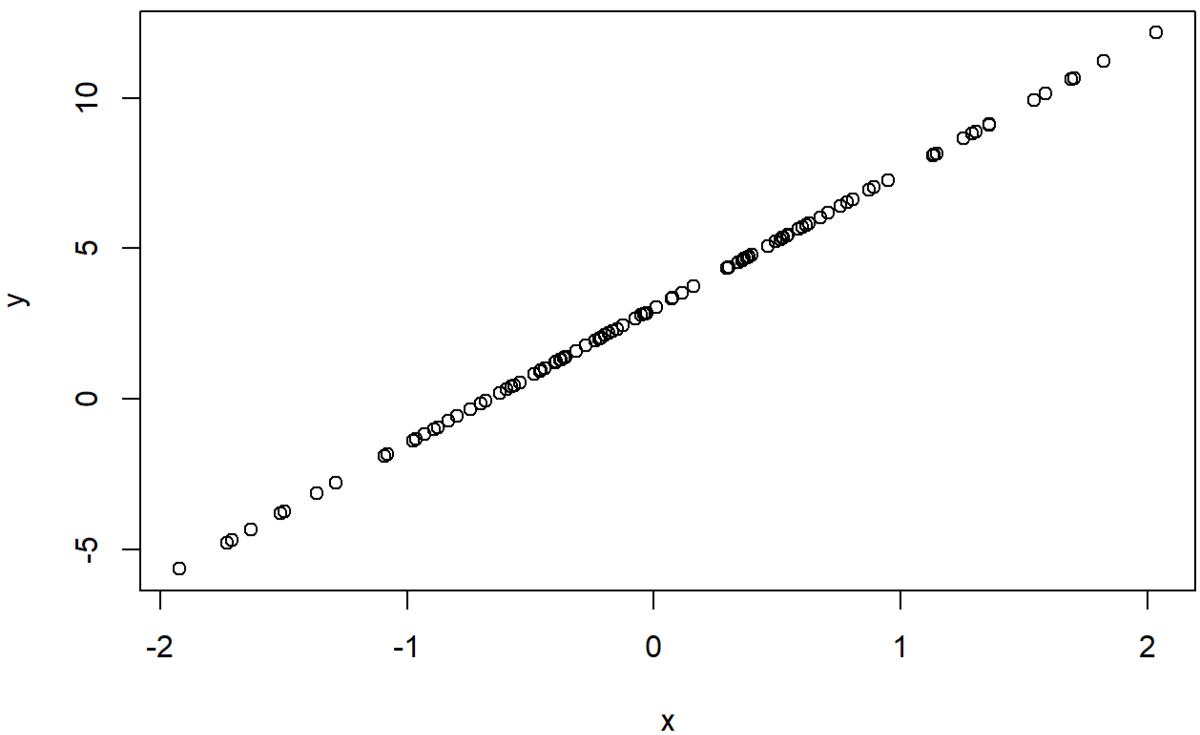
```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

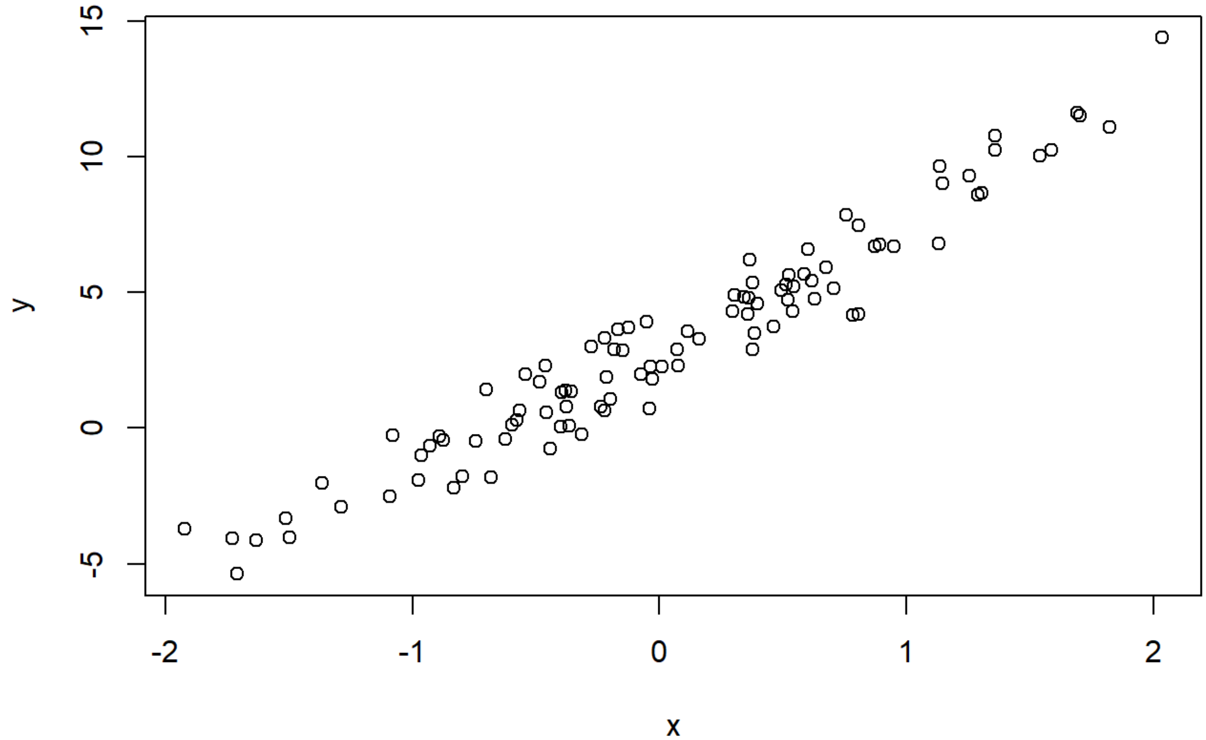
```
library(ggplot2)


##we will run 100 iterations to update the weights


##the weight and bias in a neural network can be modelled like the slope and intercept in a linear re
gression
##just like in a linear regression we try to update the beta values after each iteration, in a neural
 network
##we update the weights in a similar way

n=100
intercept=3
slope=4.5


x=rnorm(n)
hist(x)
```




```
y=x*slope+intercept
plot(x=x,y=y)
```




```
##adding noise
y=x*slope+intercept+rnorm(100)
plot(x=x,y=y)
```




```
##we add a columns of 1s to facilitate matrix mul

## its imp to notice that we want one out for each row
# of data,so essentially the row of matrix one and column of
# matrix 2 should be one.Hence since col of mat1=row of mat2
# for matrix mul hence we add another row of 1's in the data
x_b<-cbind(x,rep(1,n))
head(x_b)
```

```
##                x
## [1,] -1.4968709 1
## [2,]  2.0355967 1
## [3,]  1.1461531 1
## [4,]  0.3039515 1
## [5,]  0.4949068 1
## [6,] -0.3958989 1
```

```
##creating a dataframe
data<-data.frame(x=x_b,y=y)
data
```

```
##              x.x x.V2           y
## 1   -1.496870888    1 -4.03906278
## 2    2.035596737    1 14.38659355
## 3    1.146153104    1  9.03047109
## 4    0.303951495    1  4.89377923
## 5    0.494906755    1  5.08713731
## 6   -0.395898862    1  1.32734895
## 7    0.375597595    1  5.35595300
## 8   -0.577810016    1  0.31318230
## 9    0.893397990    1  6.76529397
## 10  -0.380008023    1  1.39871745
## 11  -0.357027710    1  1.34409871
## 12   0.009639976    1  2.25811867
## 13   1.540456189    1 10.04739810
## 14  -0.797293121    1 -1.78503446
## 15  -0.624245126    1 -0.40695245
## 16  -0.181903360    1  2.88909865
## 17   0.949147924    1  6.69302747
## 18  -0.366860714    1  0.09544085
## 19   1.288300348    1  8.60190244
## 20  -0.073802780    1  1.97212918
## 21  -1.634027117    1 -4.14141346
## 22  -0.123427222    1  3.70881671
## 23   0.364782093    1  6.21597170
## 24  -0.966146811    1 -1.02145629
## 25  -1.290545651    1 -2.89493794
## 26  -1.514009960    1 -3.31883608
## 27   0.542343311    1  5.22220973
## 28   1.255756488    1  9.31507520
## 29   0.460602619    1  3.75431205
## 30  -1.731658593    1 -4.06907539
## 31  -0.166958647    1  3.63296924
## 32  -1.366072992    1 -2.03163279
## 33   0.872536580    1  6.69154365
## 34   1.135311334    1  9.64666226
## 35  -0.041428738    1  0.73279572
## 36  -0.275325909    1  2.99167283
## 37   1.589155117    1 10.24174535
## 38  -0.223390365    1  0.65752024
## 39   0.707787567    1  5.13729319
## 40  -0.566538379    1  0.63381017
## 41   1.357883868    1 10.24890141
## 42   1.305104745    1  8.67446355
## 43  -1.092030224    1 -2.50333290
## 44  -0.457279757    1  0.56224997
## 45  -0.050783085    1  3.92656043
## 46  -0.483100104    1  1.70250676
## 47  -1.923479947    1 -3.72216254
## 48  -0.876061335    1 -0.42632108
## 49  -0.314151333    1 -0.22284664
## 50   0.384494847    1  3.49189467
## 51   0.630399668    1  4.75763437
## 52   0.076010719    1  2.29805845
## 53   0.783954787    1  4.17602028
## 54  -0.442257709    1 -0.76016634
## 55   1.130085740    1  6.81460930
```

```
## 56   0.617862224    1  5.44210419
## 57  -0.702086902    1  1.42097336
## 58   0.804946905    1  7.47715744
## 59   0.523758280    1  5.62592500
## 60   0.586711487    1  5.69205661
## 61  -0.461080382    1  2.30385968
## 62   0.361594005    1  4.80782737
## 63  -1.079538302    1 -0.26227779
## 64  -0.977102188    1 -1.90608314
## 65   0.294670124    1  4.30821877
## 66   0.112992073    1  3.57552873
## 67   0.511993941    1  5.28513768
## 68   1.359297179    1 10.79447545
## 69   0.395124629    1  4.59389869
## 70  -0.890957517    1 -0.29844304
## 71   0.602680848    1  6.59883674
## 72   0.070465483    1  2.91141830
## 73  -0.401519797    1  0.03851908
## 74  -0.929177244    1 -0.65067484
## 75  -0.831373558    1 -2.21721978
## 76   0.805025831    1  4.20709408
## 77   0.521993733    1  4.71231766
## 78   0.538438833    1  4.29498775
## 79  -0.030217794    1  1.80130241
## 80  -0.680781130    1 -1.80711180
## 81  -0.215103177    1  1.89448539
## 82  -0.544381272    1  1.99181858
## 83   0.160722764    1  3.27471360
## 84   1.824658382    1 11.08406509
## 85  -0.037768465    1  2.26001572
## 86  -0.196833948    1  1.06068617
## 87   0.342446225    1  4.85032391
## 88   0.356618960    1  4.18685231
## 89  -0.744108798    1 -0.49091736
## 90   1.693654668    1 11.63947667
## 91   0.673389450    1  5.93178738
## 92  -0.376670160    1  0.80059107
## 93  -1.710998906    1 -5.37228665
## 94   0.376146187    1  2.88479574
## 95  -0.222982160    1  3.33560950
## 96  -0.150352581    1  2.85792469
## 97  -0.238628940    1  0.77854094
## 98   0.755413855    1  7.86769189
## 99   1.703566825    1 11.51056699
## 100 -0.595217149    1  0.13009541
```

```
##defining the learning rate and no of iterations

learning_rate=0.05
n_iterations=100


##though the value of theta zero and theta one is 3 and 4.5
## we start the model with very high values which are not
## at all close to the original values
theta=matrix(c(20,20))
theta
```

```
##      [,1]
## [1,]   20
## [2,]   20
```

```
##creating vectors that store the values of the parameters and
## the errors

b0<-vector("numeric",length=n_iterations)
b1<-vector("numeric",length=n_iterations)
sse<-vector("numeric",length=n_iterations)


res<-x_b%*%theta-y

grad=(t(x_b)%*%res)*2/n
grad
```

```
##       [,1]
## x 24.85373
##   35.57653
```

```
grad*learning_rate
```

```
##       [,1]
## x 1.242686
##   1.778827
```

```
theta=theta-grad*learning_rate
theta
```

```
##       [,1]
## x 18.75731
##   18.22117
```

```
ss=(y-(x_b%*%theta))**2
head(ss)
```

```
##             [,1]
## [1,]   33.83796
## [2,] 1765.42039
## [3,]  941.84267
## [4,]  362.09171
## [5,]  502.52894
## [6,]   89.63972
```

```
###we see that after one iteration the value of weights have come down

##This we will repeat for N interations inside a loop

##reinitializing theta

theta=matrix(c(20,20))
theta
```

```
##      [,1]
## [1,]   20
## [2,]   20
```

```
for(i in 1:n_iterations)
{
  res<-(x_b%*%theta)-y
  gradient<-(t(x_b)%*%res)*2/n
  theta<-theta-(learning_rate*gradient)
  sse[i]<-sum((y-(x_b%*%theta))**2)
  b0[i]<-theta[2]
  b1[i]<-theta[1]
}

model_i<-data.frame(model_iter=1:n_iterations,sse=sse,b0=b0,b1=b1)
model_i
```

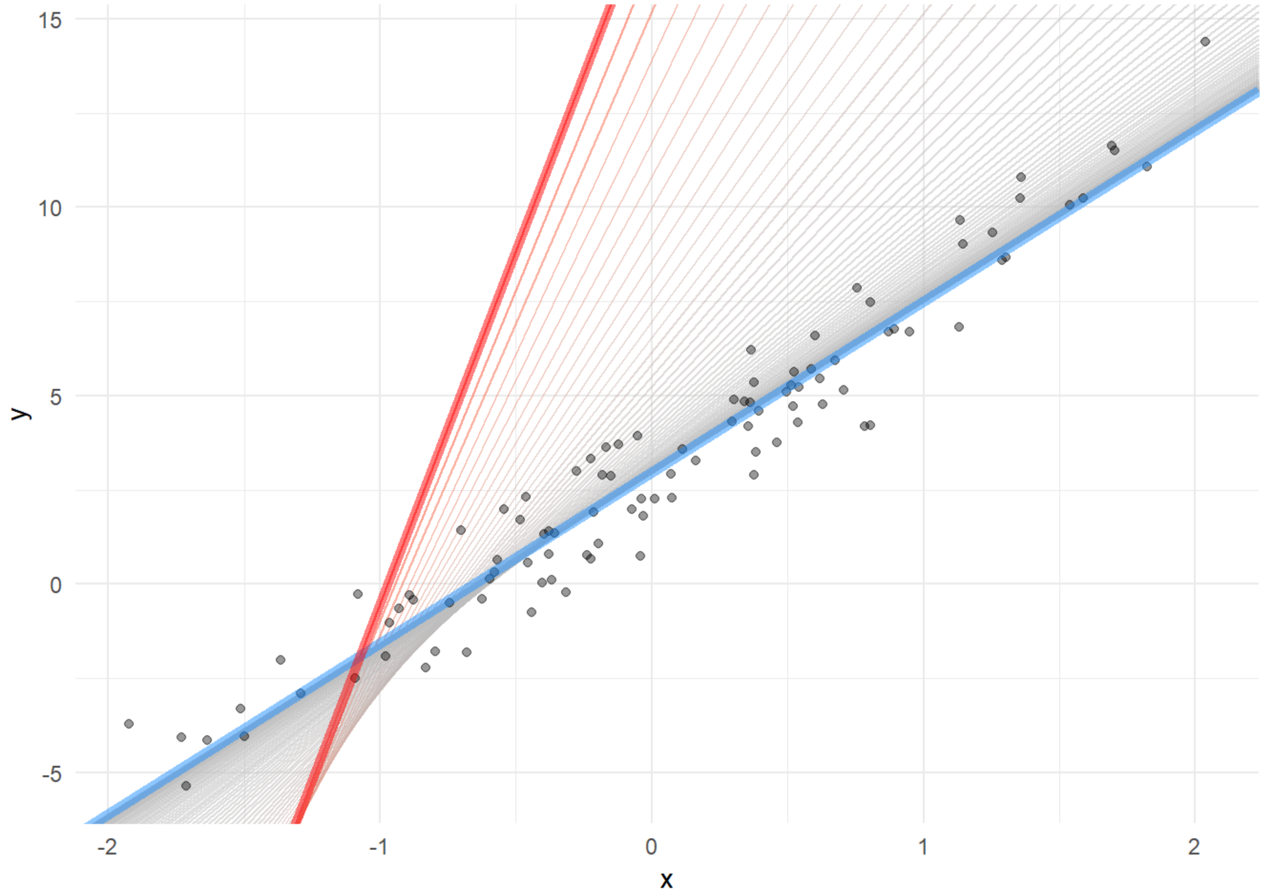
```
##    model_iter        sse        b0        b1
## 1           1 40679.1139 18.221173 18.757314
## 2           2 33357.6724 16.626219 17.616424
## 3           3 27374.3581 15.196260 16.568808
## 4           4 22482.0113 13.914347 15.606673
## 5           5 18479.5121 12.765262 14.722894
## 6           6 15203.1470 11.735346 13.910951
## 7           7 12519.6023 10.812336 13.164883
## 8           8 10320.2780  9.985223 12.479231
## 9           9  8516.6707  9.244126 11.849001
## 10         10  7036.6244  8.580176 11.269621
## 11         11  5821.2858  7.985415 10.736905
## 12         12  4822.6294  7.452697 10.247018
## 13         13  4001.4475  6.975612  9.796449
## 14         14  3325.7150  6.548408  9.381980
## 15         15  2769.2604  6.165922  9.000662
## 16         16  2310.6847  5.823523  8.649792
## 17         17  1932.4812  5.517055  8.326895
## 18         18  1620.3198  5.242790  8.029697
## 19         19  1362.4631  4.997384  7.756116
## 20         20  1149.2922  4.777838  7.504241
## 21         21   972.9188  4.581460  7.272319
## 22         22   826.8700  4.405838  7.058742
## 23         23   705.8306  4.248808  6.862033
## 24         24   605.4332  4.108429  6.680838
## 25         25   522.0867  3.982961  6.513912
## 26         26   452.8359  3.870844  6.360113
## 27         27   395.2475  3.770681  6.218392
## 28         28   347.3163  3.681217  6.087785
## 29         29   307.3884  3.601329  5.967407
## 30         30   274.0988  3.530010  5.856444
## 31         31   246.3201  3.466358  5.754150
## 32         32   223.1202  3.409564  5.659835
## 33         33   203.7279  3.358904  5.572870
## 34         34   187.5047  3.313730  5.492673
## 35         35   173.9214  3.273459  5.418710
## 36         36   162.5390  3.237572  5.350489
## 37         37   152.9932  3.205603  5.287559
## 38         38   144.9813  3.177133  5.229504
## 39         39   138.2514  3.151791  5.175941
## 40         40   132.5940  3.129241  5.126519
## 41         41   127.8347  3.109184  5.080912
## 42         42   123.8278  3.091353  5.038824
## 43         43   120.4520  3.075507  4.999979
## 44         44   117.6059  3.061434  4.964124
## 45         45   115.2046  3.048941  4.931027
## 46         46   113.1774  3.037856  4.900473
## 47         47   111.4647  3.028027  4.872264
## 48         48   110.0170  3.019318  4.846219
## 49         49   108.7924  3.011604  4.822170
## 50         50   107.7559  3.004778  4.799961
## 51         51   106.8782  2.998742  4.779452
## 52         52   106.1344  2.993408  4.760511
## 53         53   105.5039  2.988699  4.743016
## 54         54   104.9691  2.984545  4.726856
## 55         55   104.5152  2.980884  4.711929
```

```
## 56          56   104.1298  2.977661  4.698138
## 57          57   103.8025  2.974827  4.685398
## 58          58   103.5243  2.972338  4.673628
## 59          59   103.2878  2.970155  4.662752
## 60          60   103.0866  2.968242  4.652703
## 61          61   102.9154  2.966569  4.643417
## 62          62   102.7698  2.965108  4.634835
## 63          63   102.6457  2.963835  4.626904
## 64          64   102.5400  2.962727  4.619575
## 65          65   102.4500  2.961765  4.612800
## 66          66   102.3732  2.960932  4.606539
## 67          67   102.3078  2.960213  4.600751
## 68          68   102.2520  2.959593  4.595400
## 69          69   102.2044  2.959061  4.590455
## 70          70   102.1638  2.958606  4.585882
## 71          71   102.1291  2.958219  4.581655
## 72          72   102.0995  2.957891  4.577747
## 73          73   102.0743  2.957614  4.574134
## 74          74   102.0527  2.957383  4.570793
## 75          75   102.0342  2.957190  4.567703
## 76          76   102.0185  2.957032  4.564847
## 77          77   102.0050  2.956904  4.562205
## 78          78   101.9935  2.956801  4.559763
## 79          79   101.9837  2.956720  4.557504
## 80          80   101.9753  2.956658  4.555414
## 81          81   101.9681  2.956612  4.553482
## 82          82   101.9620  2.956581  4.551695
## 83          83   101.9567  2.956561  4.550043
## 84          84   101.9522  2.956551  4.548514
## 85          85   101.9483  2.956549  4.547100
## 86          86   101.9451  2.956554  4.545792
## 87          87   101.9422  2.956565  4.544582
## 88          88   101.9398  2.956581  4.543463
## 89          89   101.9378  2.956601  4.542428
## 90          90   101.9360  2.956624  4.541470
## 91          91   101.9345  2.956649  4.540584
## 92          92   101.9332  2.956675  4.539765
## 93          93   101.9321  2.956703  4.539007
## 94          94   101.9311  2.956732  4.538305
## 95          95   101.9303  2.956762  4.537656
## 96          96   101.9296  2.956791  4.537056
## 97          97   101.9290  2.956821  4.536500
## 98          98   101.9285  2.956850  4.535986
## 99          99   101.9281  2.956879  4.535511
## 100        100   101.9277  2.956907  4.535070
```

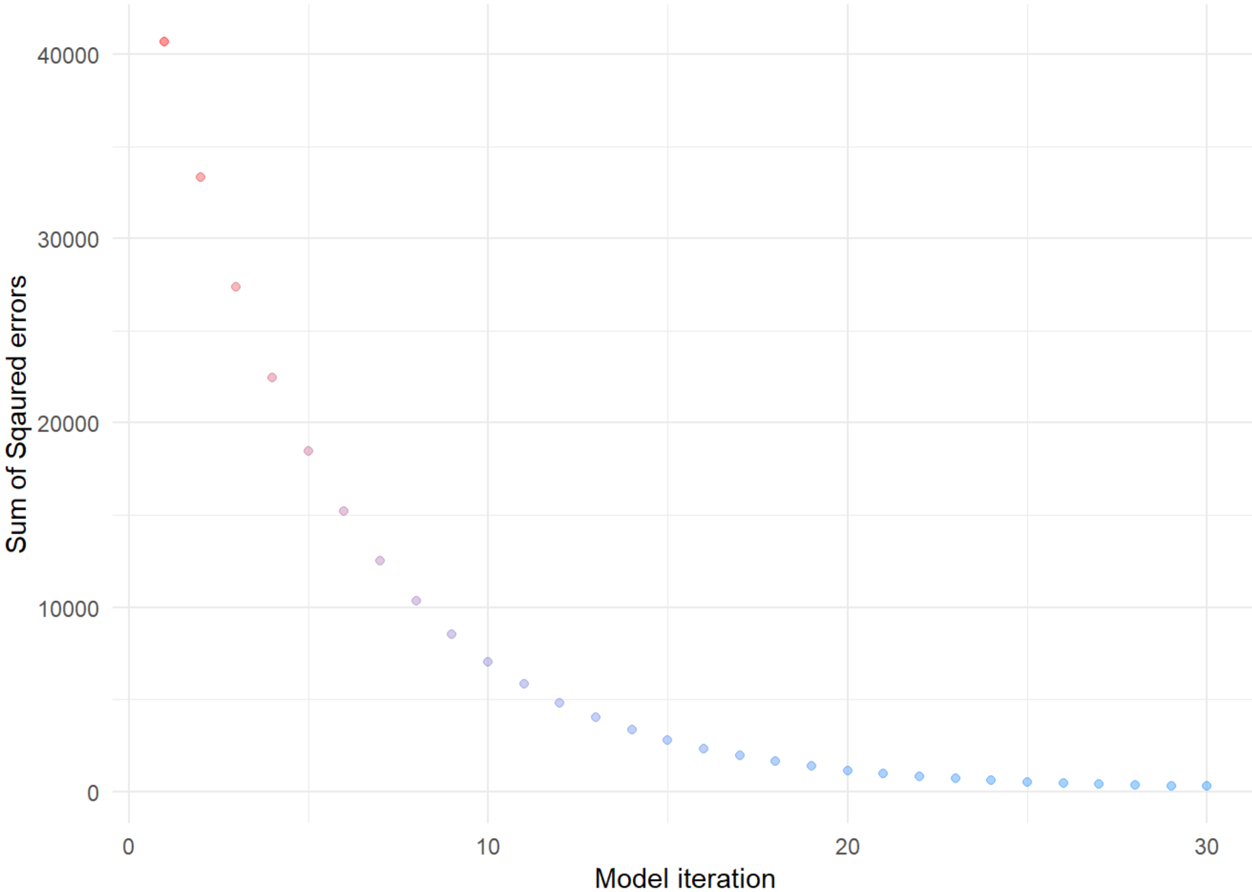
```
###plotting all the lines and the final regression line

p1 <- data %>%
  ggplot(aes(x=x, y=y)) +
  geom_abline(aes(intercept = b0,
                  slope = b1,
                  colour = -sse,
                  frame = model_iter),
              data = model_i,
              alpha = .50
  ) +
  geom_point(alpha = 0.4) +
  geom_abline(aes(intercept = b0,
                  slope = b1),
              data = model_i[100, ],
              alpha = 0.5,
              size = 2,
              colour = "dodger blue") +
  geom_abline(aes(intercept = b0,
                  slope = b1),
              data = model_i[1, ],
              colour = "red",
              alpha = 0.5,
              size = 2) +
  scale_color_continuous(low = "red", high = "grey") +
  guides(colour = FALSE) +
  theme_minimal()
```

```
## Warning: Ignoring unknown aesthetics: frame
```

```
p1
```




```
##plotting the decreasing sse errors

p2 <- model_i[1:30,] %>%
  ggplot(aes(model_iter, sse, colour = -sse)) +
  geom_point(alpha = 0.4) +
  theme_minimal() +
  labs(x = "Model iteration",
       y = "Sum of Sqaured errors") +
  scale_color_continuous(low = "red", high = "dodger blue") +
  guides(colour = FALSE)
p2
```


40000
30000
20000
10000
0
Sum of Sqaured errors
0
10
20
30
Model iteration