module2_part2

Andrew Estes

3/29/2022

```
library(tidyverse)
```

1

Write code to simulate a set of 100 random x values between 0 and 1 and a set of y values that generally follow the line y = 1 + 2x, but with a random error term added in.

This would be a standard case where linear regression is appropriate. Plot your points.

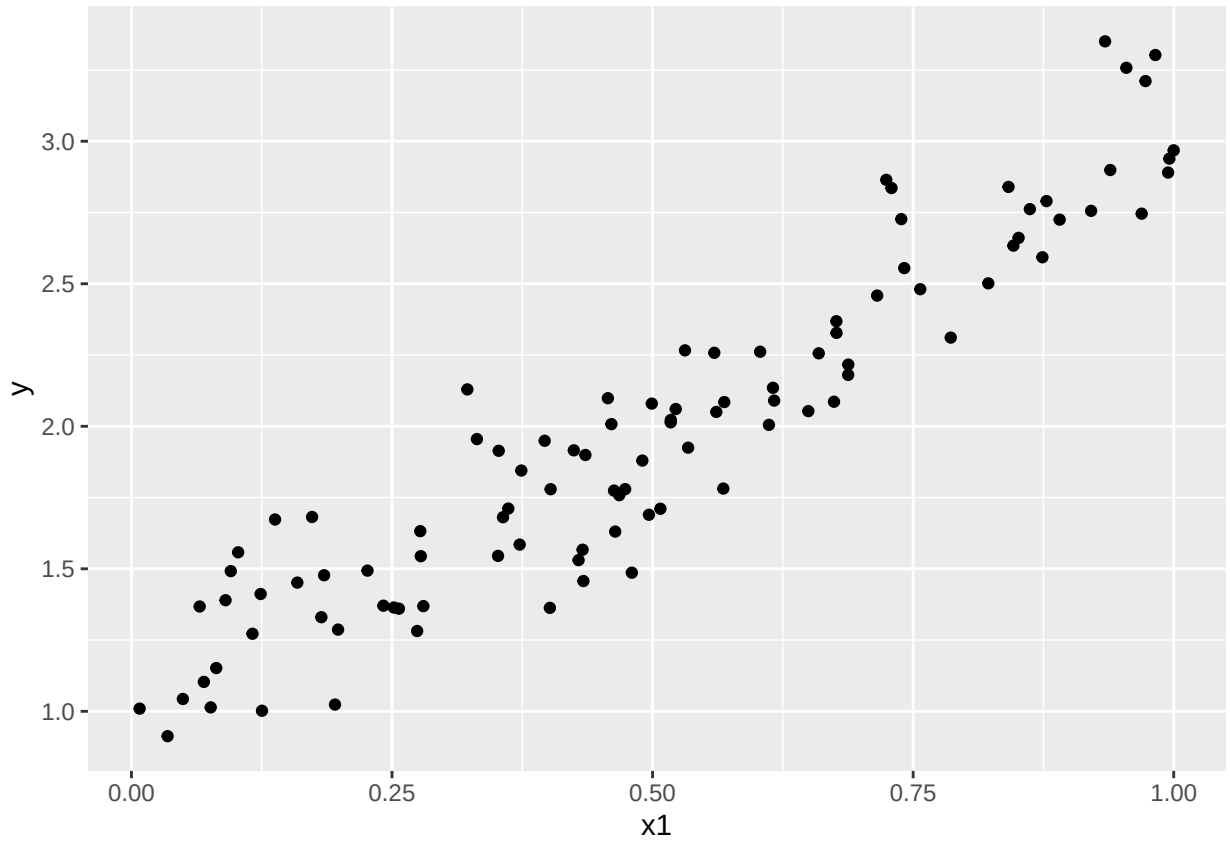
```
set.seed(9876)

#initializing the direction vector
w <- runif(2)

#creating the dataframe
df <- data.frame(
  x1 = runif(100, min=0, max=1),
  y = numeric(100)
)

df <- df %>%
  mutate(y = 1 + (2*x1) + rnorm(nrow(df), mean=0, sd=0.2))


ggplot(df) +
  geom_point(aes(x=x1, y=y))
```


3.0
2.5
2.0
1.5
1.0
y
0.00
0.25
0.50
0.75
1.00
x1

2

Define a loss function to calculate the usual loss function for linear regression, the sum of squared residuals. The function should do the following:

A) Take as input a vector b of length 2, as well as your vector of x values and your vector of y values. The element b1 should represent the line's intercept, and b2 should represent its slope.

B) Inside the function, compute a vector of y hat values

C) Use those computed values to calculate and return the sum of squared residuals.

```
#the error has to deal with some peculiarity of rmd and knittr
#it works just find in rmd but when knitting, it throws an error

fit.loss <- function(b, x, y){
  y.hat <- b[1] + (b[2]*x)
  sse <- sum((y - y.hat)^2)

  return(sse)
}

fit.loss(df, x, y)
```

```
## Error in fit.loss(df, x, y): object 'x' not found
```

3

Write code to use the optim function to find the vector b that minimizes the sum of squared residuals.

A) Use the Nelder-Mead method
B) Use Simulated Annealing method

```
x <- df$x1
y <- df$y

b.init <- c(.1, .1)

#3a - nelder mead
b.opt.neldermead <- optim(b.init, fit.loss, x=x, y=y, method="Nelder-Mead")
glimpse(b.opt.neldermead)
```

```
## List of 5
##  $ par        : num [1:2] 0.979 2.006
##  $ value      : num 4.32
##  $ counts     : Named int [1:2] 93 NA
##   ..- attr(*, "names")= chr [1:2] "function" "gradient"
##  $ convergence: int 0
##  $ message    : NULL
```

```
#3b - nelder mead
b.opt.simann <- optim(b.init, fit.loss, x=x, y=y, method="SANN")
glimpse(b.opt.simann)
```

```
## List of 5
##  $ par        : num [1:2] 0.982 2.002
##  $ value      : num 4.32
##  $ counts     : Named int [1:2] 10000 NA
##   ..- attr(*, "names")= chr [1:2] "function" "gradient"
##  $ convergence: int 0
##  $ message    : NULL
```

C) You can also try using BFGS method with gradient functions.

```
#3c - BFGS
#this did not work.
#procedurally, the first step is to take a partial derivative of initial variables
#called in the function
#then, output those partial derivatives into an encompassing "gr.total" vector
#then, utilize the optim function -
    #input #s that are not equivalent to the gr.total vector

bfgs.loss <- function(b, x, y){
  y.hat <- b[1] + b[2]*x
  sse <- sum((y - y.hat)^2)

  gr.b1 <- D(sse, 'b[1]')
  gr.b2 <- D(sse, 'b[2]')
  gr.x <- D(sse, 'x')
  gr.y <- D(sse, 'y')

  gr.total <- c(gr.b1, gr.b2, gr.x, gr.y)

  return(gr.total)
}

#optim(
#  par=c(1, 1, 1, 1),
  #fn = bfgs.loss,
  #gr = gr.total,
  #method="L-BFGS-B"
#)
```

4

The lm command finds regression coefficients. Use it and make a table showing the coefficient estimates from part 3 and here. Comment on how closely optim came to matching lm.

```
linear <- lm(y ~ x1, data=df)
linear$coefficients
```

```
## (Intercept)          x1
##   0.9790054   2.0057343
```

Optim came very close to the coefficients provided by the linear regression command. In this particular instance, Nelder-Mead performed slightly better than Simulated Annealing.

5

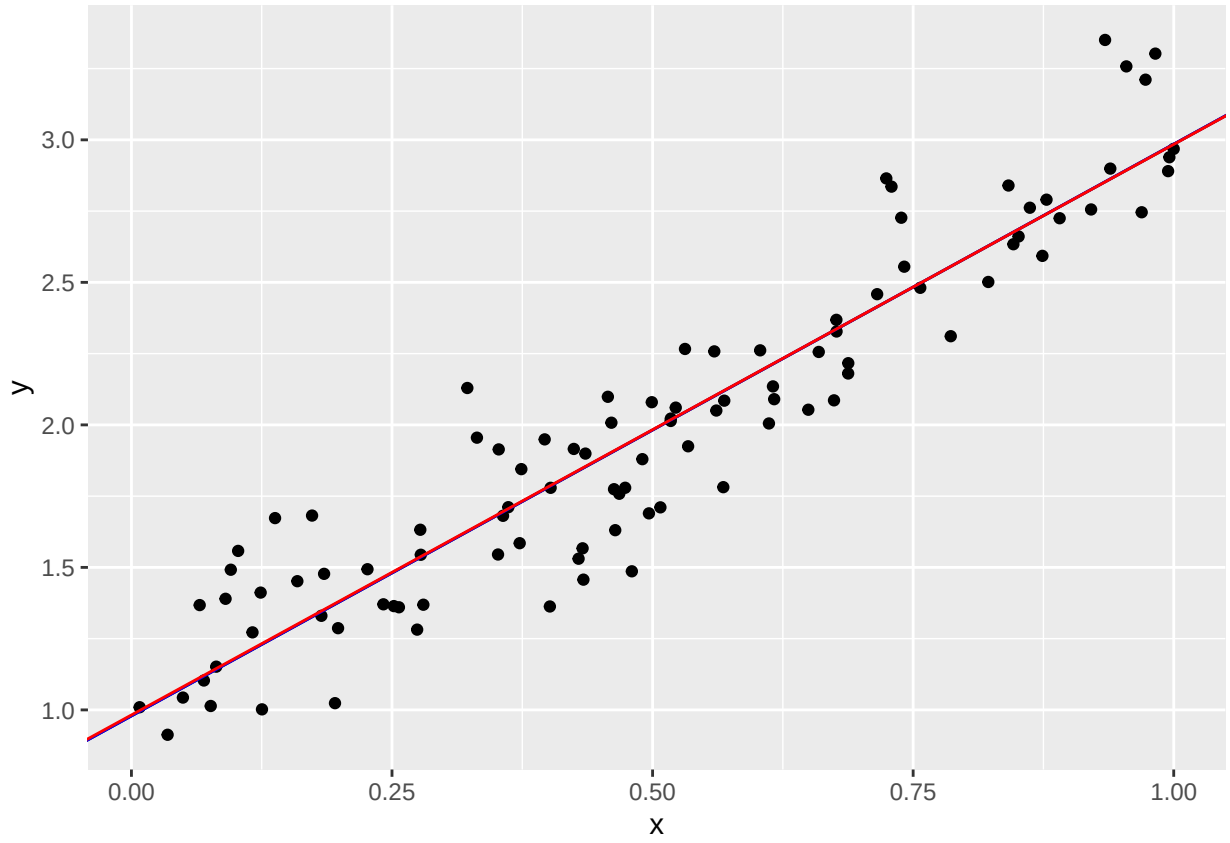
Create a graph showing the three lines obtained by the three methods, overlayed on top of the data points. The plot/abline commands or ggplot/geom_abline commands will be helpful. Comment on what you see in the graph.

```
ggplot(df) +
  geom_point(aes(x=x, y=y)) +

  geom_abline(intercept = linear$coefficients[1],
              slope = linear$coefficients[2]) +

  geom_abline(intercept = b.opt.neldermead$par[1],
              slope = b.opt.neldermead$par[2], col="blue") +

  geom_abline(intercept = b.opt.simann$par[1],
              slope = b.opt.simann$par[2], col="red")
```



All three lines are basically the same. This is to be expected given the tiny difference in the coefficients each algorithm found.

6

Create your own loss function for fitting a line.

```
#not great code etiquette, sorry for the repeated functions!
#also should look into weighting the RMSE and MAE at multiple levels
#rather than the 3 predefined weights given

equal.loss <- function(b, x, y){
  y.hat <- b[1] + (b[2]*x)

  #calculating RMSE and MAE by hand and weighting them
  equal.error <- (.5 * (sqrt((1/nrow(df)*(sum((y - y.hat)^2))))) +
                (.5 * (1/nrow(df)*(sum(abs(y - y.hat)))))

  #rmse.focused.error <- (.8 * (sqrt((1/nrow(df)*(sum((y - y.hat)^2))))) +
  #                       (.2 * (1/nrow(df)*(sum(abs(y - y.hat)))))

  #mae.focused.error <- (.2 * (sqrt((1/nrow(df)*(sum((y - y.hat)^2))))) +
  #                      (.8 * ((1/nrow(df)*(sum(abs(y - y.hat))))))

  return(equal.error)
}


rmse.loss <- function(b, x, y){
  y.hat <- b[1] + (b[2]*x)

  #calculating RMSE and MAE by hand and weighting them
  #equal.error <- (.5 * (sqrt((1/nrow(df)*(sum((y - y.hat)^2))))) +
  #               (.5 * (1/nrow(df)*(sum(abs(y - y.hat)))))

  rmse.focused.error <- (.8 * (sqrt((1/nrow(df)*(sum((y - y.hat)^2))))) +
                         (.2 * (1/nrow(df)*(sum(abs(y - y.hat)))))

  #mae.focused.error <- (.2 * (sqrt((1/nrow(df)*(sum((y - y.hat)^2))))) +
  #                      (.8 * ((1/nrow(df)*(sum(abs(y - y.hat))))))

  return(rmse.focused.error)
}

mae.loss <- function(b, x, y){
  y.hat <- b[1] + (b[2]*x)

  #calculating RMSE and MAE by hand and weighting them
  #equal.error <- (.5 * (sqrt((1/nrow(df)*(sum((y - y.hat)^2))))) +
  #               (.5 * (1/nrow(df)*(sum(abs(y - y.hat)))))

  #rmse.focused.error <- (.8 * (sqrt((1/nrow(df)*(sum((y - y.hat)^2))))) +
  #                       (.2 * (1/nrow(df)*(sum(abs(y - y.hat)))))

  mae.focused.error <- (.2 * (sqrt((1/nrow(df)*(sum((y - y.hat)^2))))) +
                        (.8 * ((1/nrow(df)*(sum(abs(y - y.hat))))))
```

```
  return(mae.focused.error)
}



new.init <- c(.1, .1)

equal.opt.neldermead <- optim(b.init, equal.loss, x=x, y=y, method="Nelder-Mead")
equal.opt.simann <- optim(b.init, equal.loss, x=x, y=y, method="SANN")

rmse.opt.neldermead <- optim(b.init, rmse.loss, x=x, y=y, method="Nelder-Mead")
rmse.opt.simann <- optim(b.init, rmse.loss, x=x, y=y, method="SANN")

mae.opt.neldermead <- optim(b.init, mae.loss, x=x, y=y, method="Nelder-Mead")
mae.opt.simann <- optim(b.init, mae.loss, x=x, y=y, method="SANN")

b.opt.neldermead <- optim(b.init, fit.loss, x=x, y=y, method="Nelder-Mead")
b.opt.simann <- optim(b.init, fit.loss, x=x, y=y, method="SANN")

df2 <- data.frame(
  c(linear$coefficients, equal.opt.neldermead[1], rmse.opt.neldermead[1],
    mae.opt.neldermead[1], b.opt.neldermead[1], equal.opt.simann[1],
    rmse.opt.simann[1], mae.opt.simann[1], b.opt.simann[1])
)

names(df2) <- c("lm x", "lm y", "neldermead equal weight",
                "nelder mead rmse weighted", "nelder mead mae weighted",
                "nelder mead pre-defined function", "simann equal weight",
                "simann rmse weighted", "simann mae weighted",
                "simann pre-defined function")

#further work would include finding the absolute value difference between the
#x, y coordinates of the vaarious methods compared to the lm method
#also, the df needs to be cleaned up a bit
glimpse(df2)
```

```
## Rows: 2
## Columns: 10
## $ `lm x`                              <dbl> 0.9790054, 0.9790054
## $ `lm y`                              <dbl> 2.005734, 2.005734
## $ `neldermead equal weight`           <dbl> 0.9873484, 1.9733969
## $ `nelder mead rmse weighted`         <dbl> 0.9838138, 1.9875854
## $ `nelder mead mae weighted`          <dbl> 0.991986, 1.960827
## $ `nelder mead pre-defined function` <dbl> 0.9787434, 2.0063090
## $ `simann equal weight`               <dbl> 0.999086, 1.980341
## $ `simann rmse weighted`              <dbl> 0.9600089, 1.9655846
## $ `simann mae weighted`               <dbl> -0.2761922, 4.1931307
## $ `simann pre-defined function`       <dbl> 0.9796707, 2.0014767
```

The biggest standouts were the function I created utilizing the Simulated Annealing Method with a MAE emphasis. It's slope and intercept were far from the linear regression model values. This is likely due to the emphasis on absolute error for a very linear pattern. I suspect if this was a non-linear dataframe, the simulated annealing with MAE emphasis would have better results.