Biomath 204 hw2

Benjamin Chu

2/17/2017

Problem 1

Define $f(x) = x^2 - a, f'(x) = 2x$. According to newton's method, we have

$$x_{n+1} = x_n - \frac{f(x)}{f'(x)} = x_n - \frac{x_n^2 - a}{2x_n} = \frac{1}{2}(x_n - \frac{a}{x_n}).$$

```
newton <- function(f, fp, x) {
    tol = 1e-14
    xo = x
    xn = Inf
    iterations = 1
    for (i in 1:100) {
        xn = xo - f(xo) / fp(xo)
        if (abs(xn - xo) < tol) {
            break
        }
        # print(xn)
        # print(abs(xn - 12))
        xo = xn
        iterations = iterations + 1
    }

    if (iterations > 100) {
      error("Did not converge in 100 steps")
    }
    else {
      # print(iterations)
      return(xn)
    }
}

a = 144
test <- function(x) {return (x^2 - a)}
test_derivative <- function(x) {return (2*x)}
initial_guess = 2.34
newton(test, test_derivative, initial_guess)
```

```
## [1] 12
```

To show empiracally that this does converge to the desired quantity, I tested for the case $a = 144$, whose square root is 12. With initial guess 2.34, it took 8 steps to converge within tolerance of 10e-14. I never got the plots to work, but I can illustrate the same idea using numbers. For each iterations we have

$$x_1 = 31.93923, x_2 = 18.2239, x_3 = 13.0628, x_4 = 12.04324, x_5 = 12.00008, x_6 = x_7 = x_8 = 12$$

From this hint, we know that quadratic convergence implies $e_{n+1} \leq Ke_n^2 \iff K \geq \frac{e_{n+1}}{e_n^2}$ for some K and all

n. From the 7 terms above, the errors are

$$e_1 = 19.93923, e_2 = 6.223896, e_3 = 1.062805, e_4 = 0.0432355, e_5 = 7.760824e-05, e_6 = 2.509584e-10, e_7 = 0$$

and from this the first couple K's are:

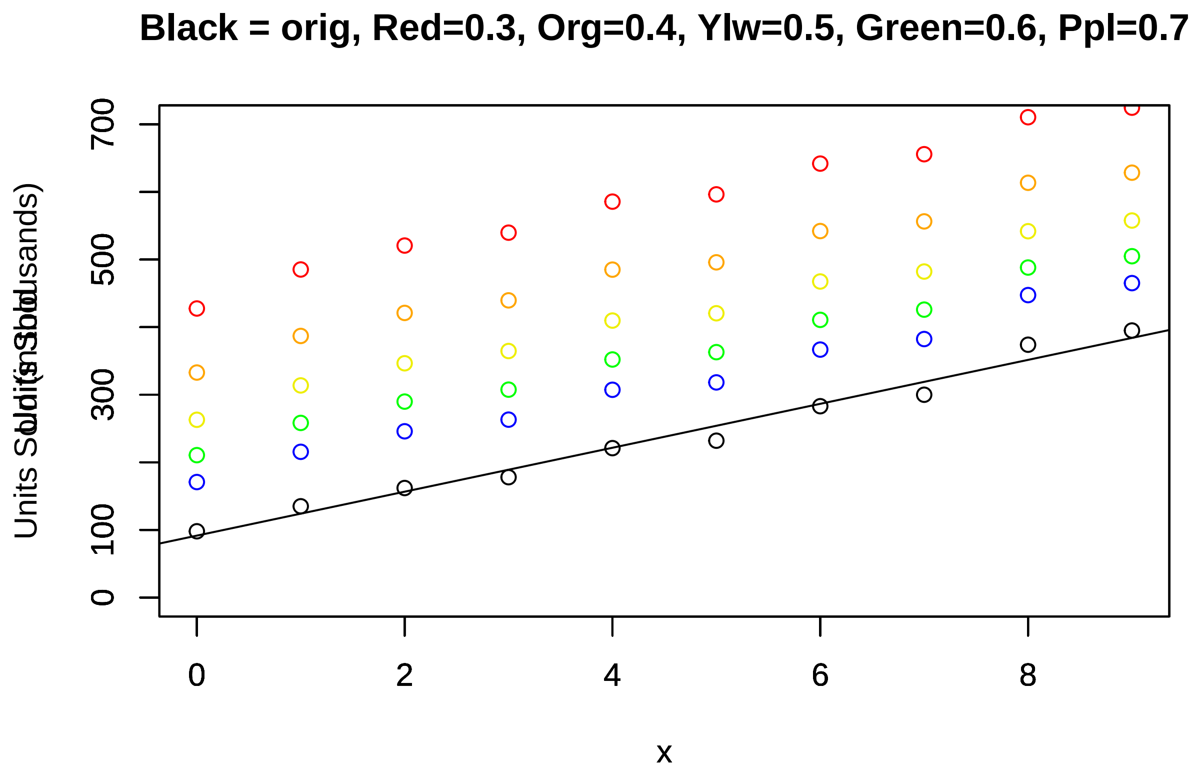
$$K_1 = 0.01565473, K_2 = 0.02743651, K_3 = 0.0382766, K_4 = 0.04151709, K_5 = 0.04166641$$

We observe that the K's eventually stabalize at around 0.042. In other words, we can pick $K = 0.05$ to satisfy $K \geq \frac{e_{n+1}}{e_n^2}$ for all n. i.e. we have quadratic convergence.

Problem 2

```
library(MASS)
x = seq(0, 9, 1)
y = c(98, 135, 162, 178, 221, 232, 283, 300, 374, 395)
regl <- lm(y ~ x)
plot(x, y, ylim=c(0,700), ylab="Units Sold")
abline(regl)

g <- function(y) {
  n = length(y)
  output = 1
  for (num in y) {
    output = output * num^(1/n)
  }
  return(output)
}

z <- function(y, lambda, g) {
  output <- list()
  for (i in y) {
    z_i = (i^lambda - 1) / lambda / g^(lambda-1)
    # z_i = (i^lambda - 1) / lambda
    output <- append(output, z_i)
  }
  return(output)
}

num = g(y)
three = z(y, 0.3, num)
four = z(y, 0.4, num)
five = z(y, 0.5, num)
six = z(y, 0.6, num)
seven = z(y, 0.7, num)

par(new=TRUE)
plot(x, three, col="red", ylim=c(0,700), ylab="Units Sold (in thousands)", main="Black = orig, Red=0.3,
par(new=TRUE)
plot(x, four, col="orange", ylim=c(0,700), ylab="Units Sold")
par(new=TRUE)
plot(x, five, col="yellow2", ylim=c(0,700), ylab="Units Sold")
par(new=TRUE)
plot(x, six, col="green", ylim=c(0,700), ylab="Units Sold")
```

```r
par(new=TRUE)
plot(x, seven, col="blue", ylim=c(0,700), ylab="Units Sold")
```



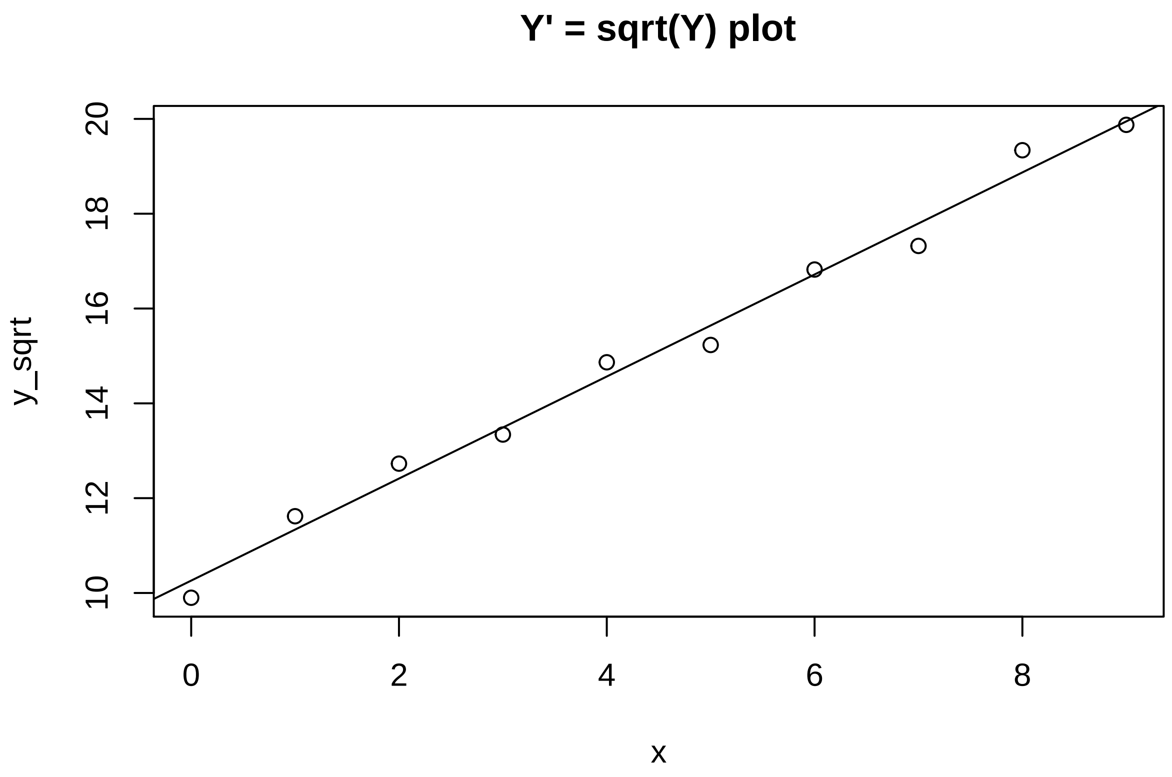
```r
# anova(lm(unlist(three) ~ x))
# anova(lm(unlist(four) ~ x))
# anova(lm(unlist(five) ~ x))
# anova(lm(unlist(six) ~ x))
# anova(lm(unlist(seven) ~ x))
```

It appears that the function, both before (black) and after box-cox transfer(red ~ blue) is quite linear. Here yellow line is the box cox transform with $\lambda = 0.5$. The SSE for each case can be computed by calling anova(lm(y ~ x)) as long as we converted the list element into a vector by unlisting them first. The SSE for the original = 1799, and the transformed ones have

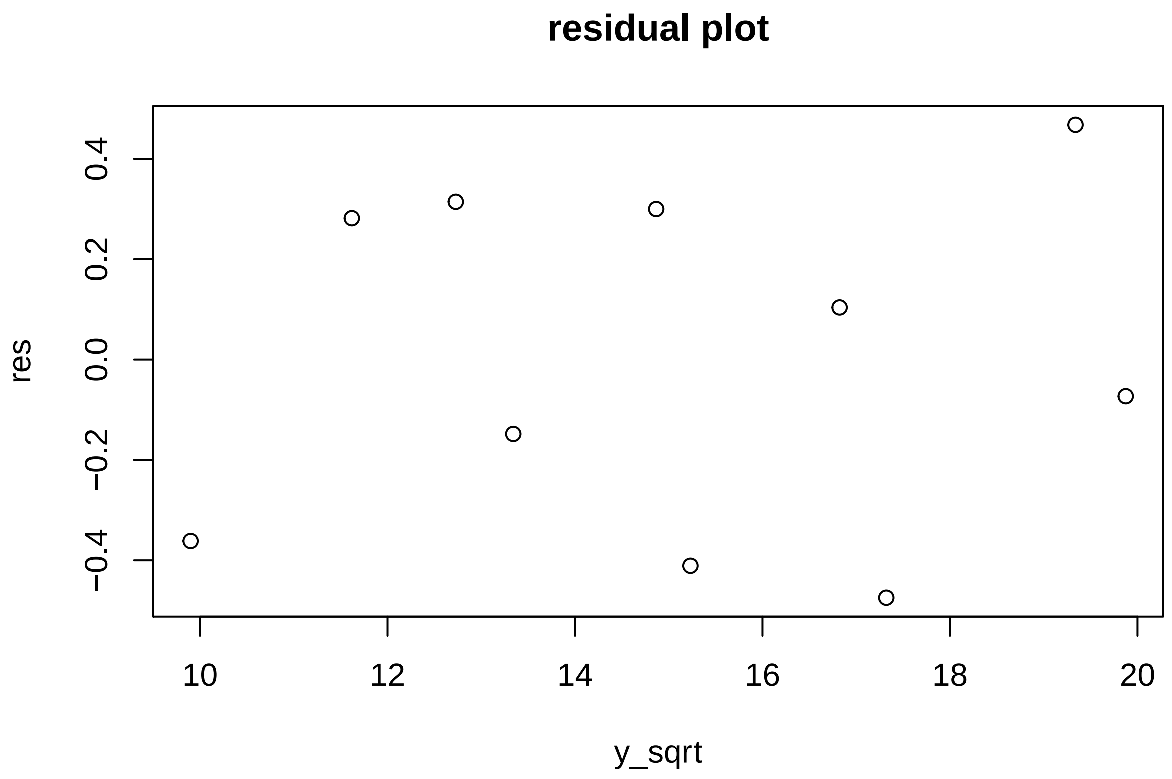
$$\text{SSE: } \lambda_{0.3} = 1100, \lambda_{0.4} = 968, \lambda_{0.5} = 916, \lambda_{0.6} = 942, \lambda_{0.7} = 1044$$

In particular this suggests that $\lambda = 0.5$ is the transform that minimizes the SSE.

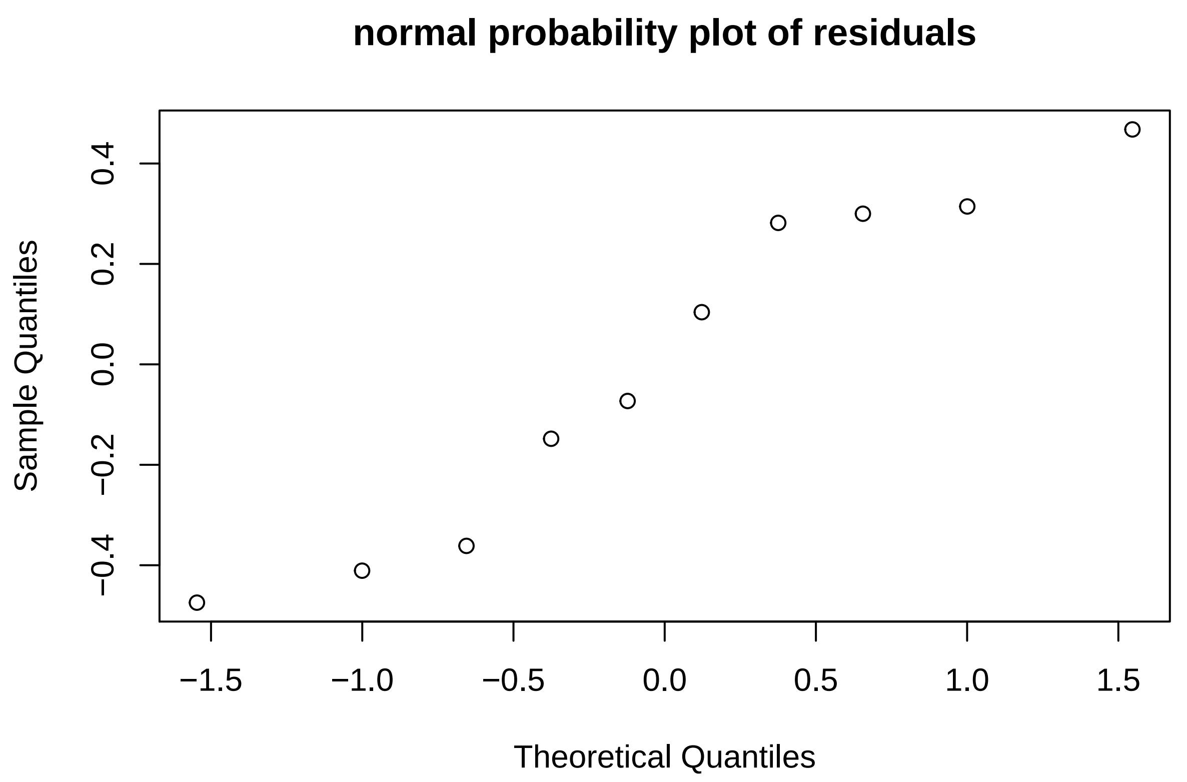
```r
x = seq(0, 9, 1)
y = c(98, 135, 162, 178, 221, 232, 283, 300, 374, 395)
y_sqrt = c(98^.5, 135^.5, 162^.5, 178^.5, 221^.5, 232^.5, 283^.5, 300^.5, 374^.5, 395^.5)

regl <- lm(y_sqrt ~ x)
plot(x, y_sqrt, main="Y' = sqrt(Y) plot")
abline(regl)
```



```
res <- residuals(lm(y_sqrt ~ x))
plot(y_sqrt, res, main="residual plot")
```



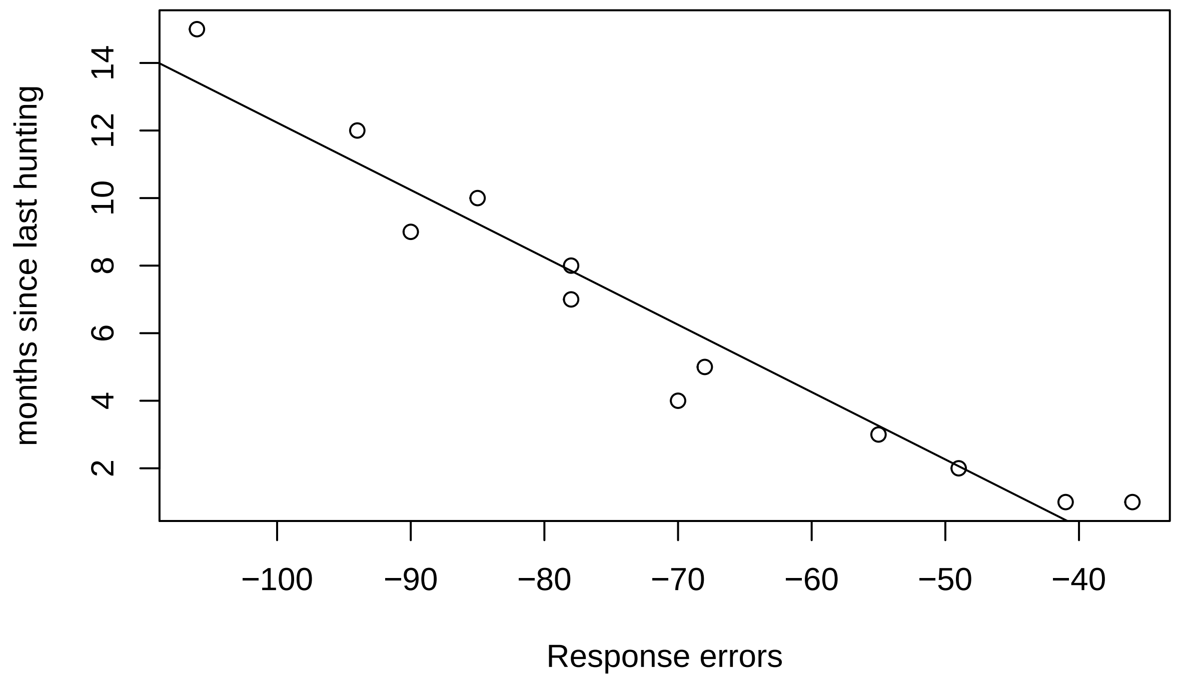
```
qqnorm(res, main="normal probability plot of residuals")
```



Here the regression line appears to be a reasonably good fit to the transformed data, since it's linear. It appears that just plotting the residuals against the fitted values ($\sqrt{Y}$) does have have any indication that it's normally distributed. However graphing it in the normal probability plot, we see a rather straight line, indicating that it is almost normal.

Problem 3

```
mydata = read.table("ch03pr18.dat.txt")
x = mydata[1]
y = mydata[2]
regl <- lm(unlist(y) ~ unlist(x))
plot(unlist(x), unlist(y), xlab="Response errors", ylab="months since last hunting")
abline(regl)
```



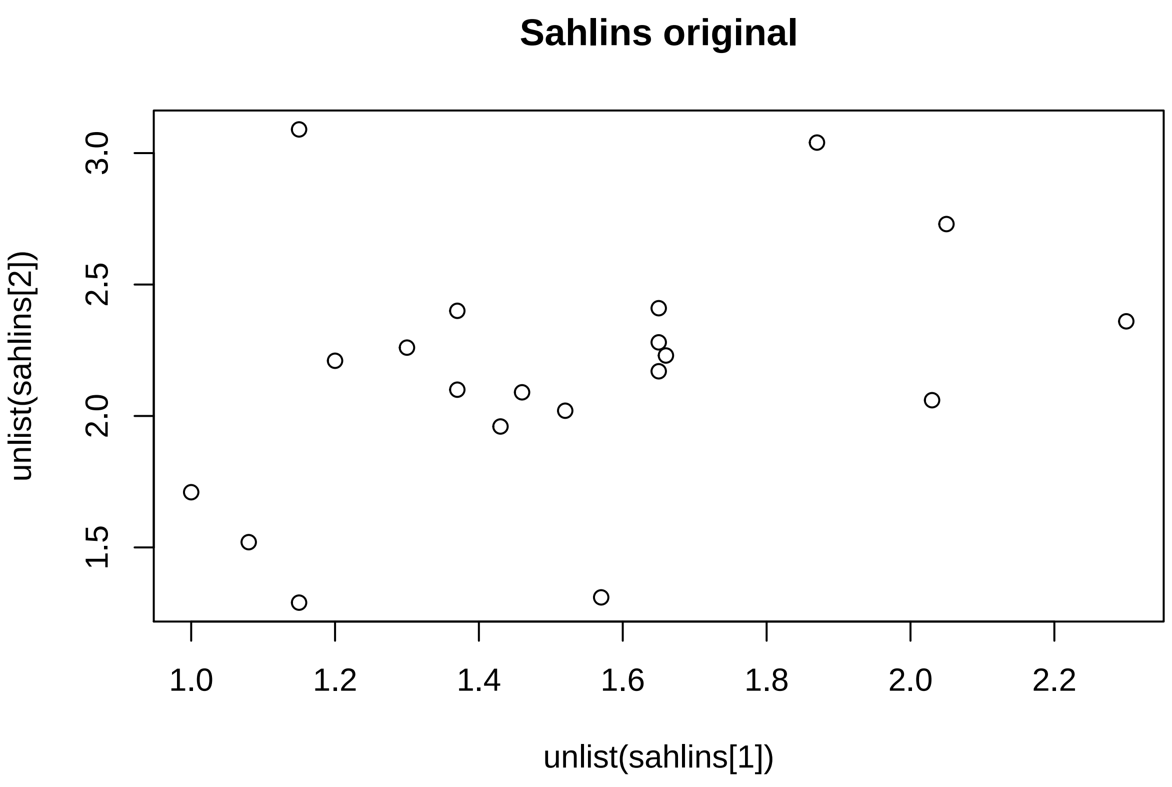
```
#residuals(lm(unlist(mydata[2]) ~ unlist(mydata[1])))
```

Here the correlation between response errors and the months since last hinting is very linear, with correlation of -0.997. If we were to rescale this plot, I propose to transform the Y-axis. This is because the plot has slightly more data points towards the right side of the x-axis (i.e. negatively skewed). If we had instead scaled the x-axis, we would need to push data points on the right side to the left, rendering them even closer to each other. Thus the graph would appear as if many data points are on the right side with little on the left, which is presumably undesirable.

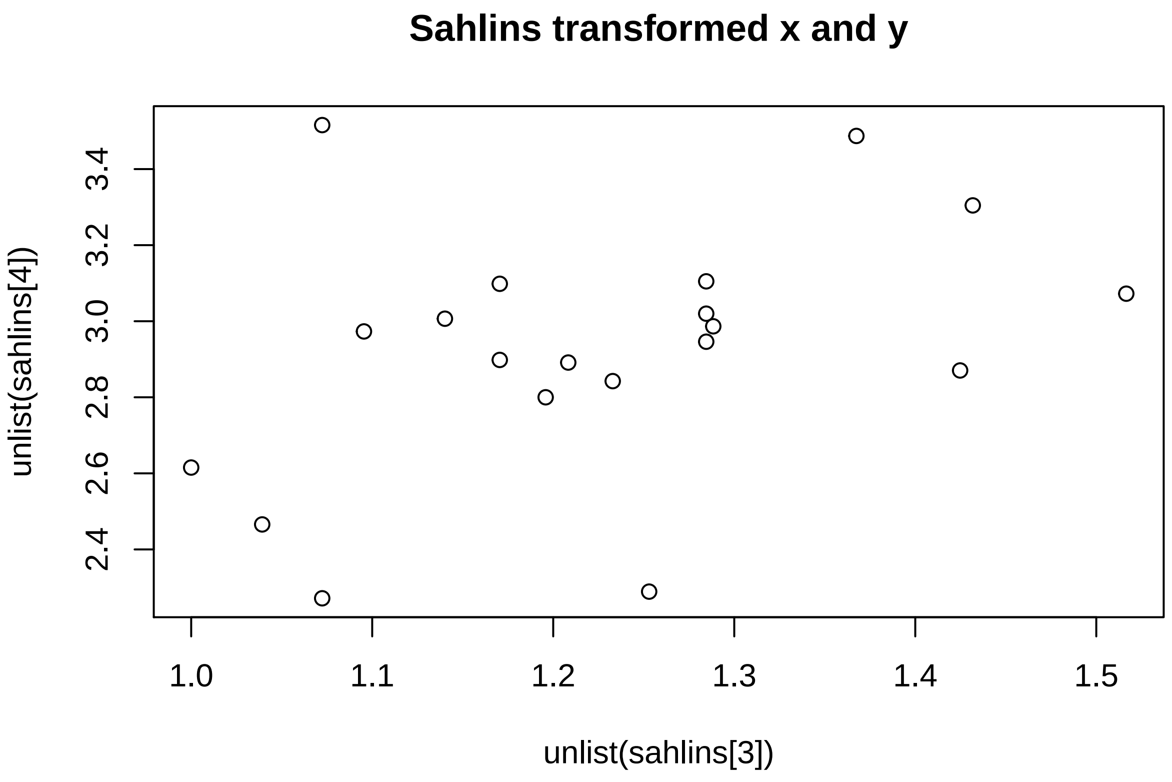
Problem 4

This problem (taken from version 1) asks to make four datasets more linear. To do so, I transformed one or both variables.

```
sahlins = read.table("sahlins.txt")
sahlins$trans_X <- (sahlins$consumers)^(1/2)
sahlins$trans_Y <- (sahlins$acres * 4)^(1/2)
plot(unlist(sahlins[1]), unlist(sahlins[2]), main="Sahlins original")
```

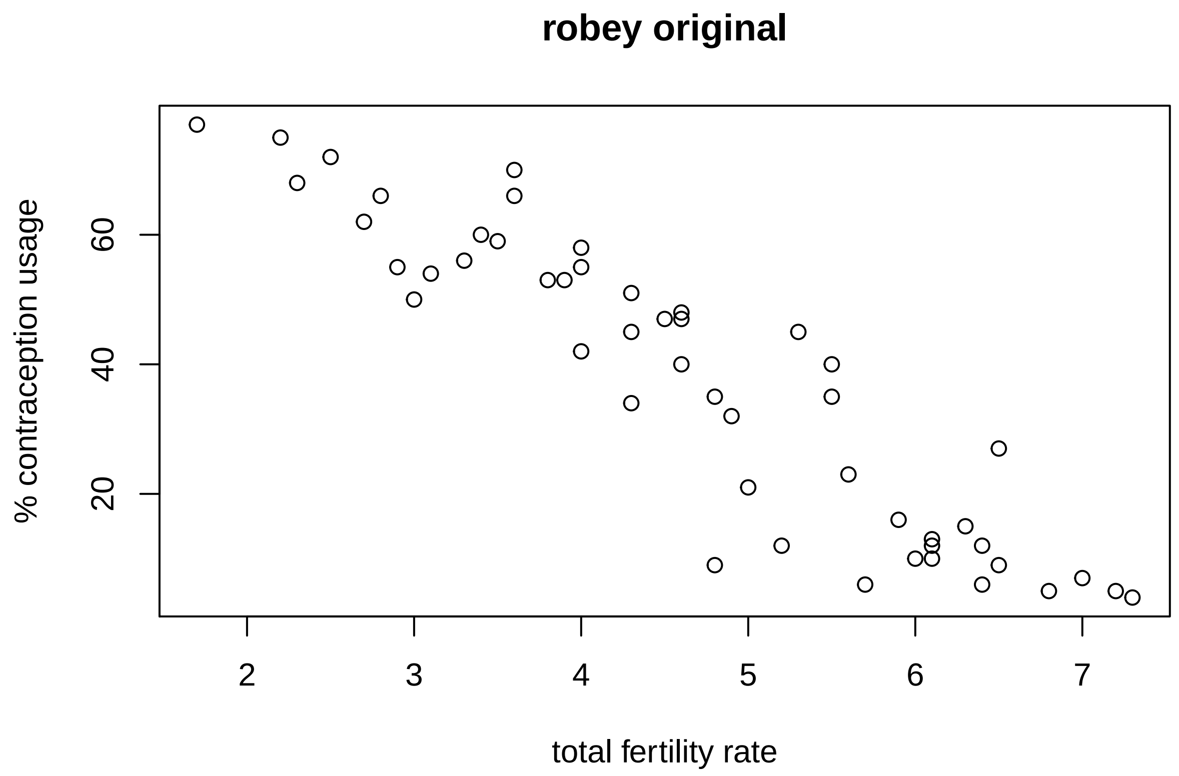


```r
plot(unlist(sahlins[3]), unlist(sahlins[4]), main="Sahlins transformed x and y")
```

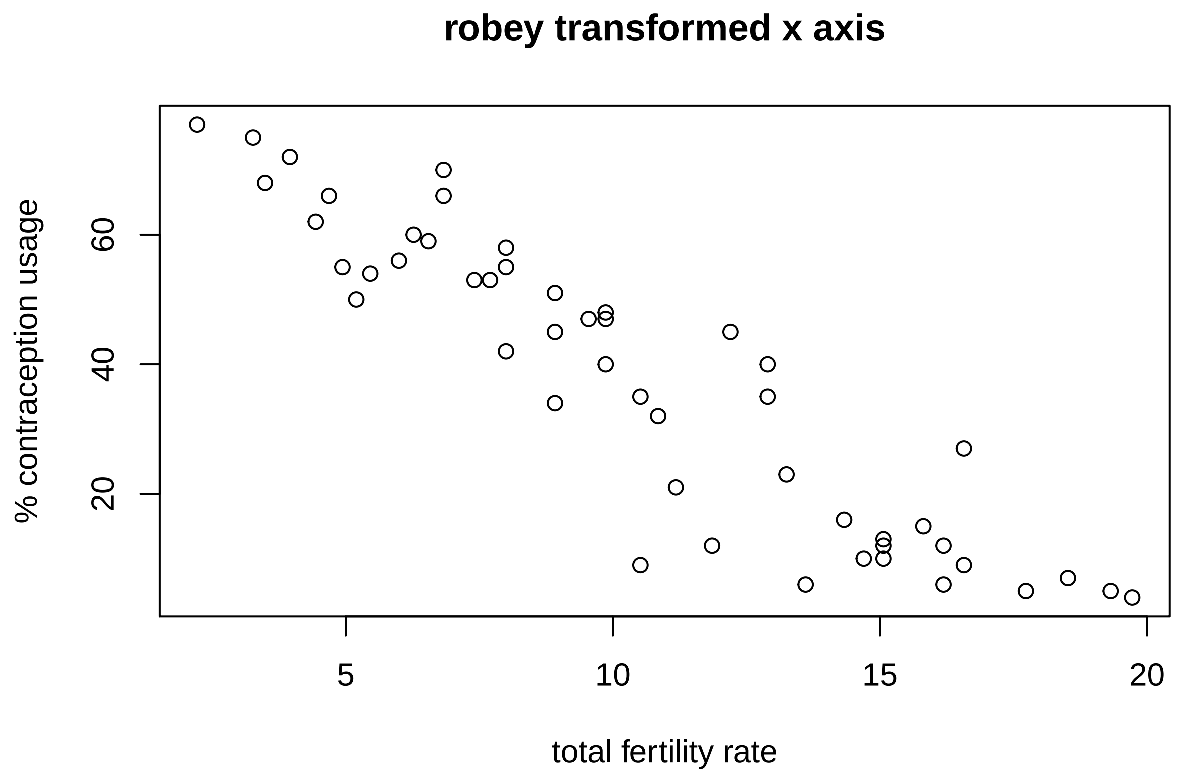


Here I transformed both x and y according to $X = \sqrt{X}, Y + Y^{1/3}$. The rational behind doing this is that the original data has very dispersed X values, so I want to compress them towards the left. The Y values are just too messed up for small Y values. I couldn't figure out how to make this plot look linear.

```
robey <- read.table("Robey.txt")
robey$x_trans <- (robey$tfr)^1.5
plot(robey$tfr, robey$contraceptors, main="robey original", xlab="total fertility rate", ylab="% contrac
```

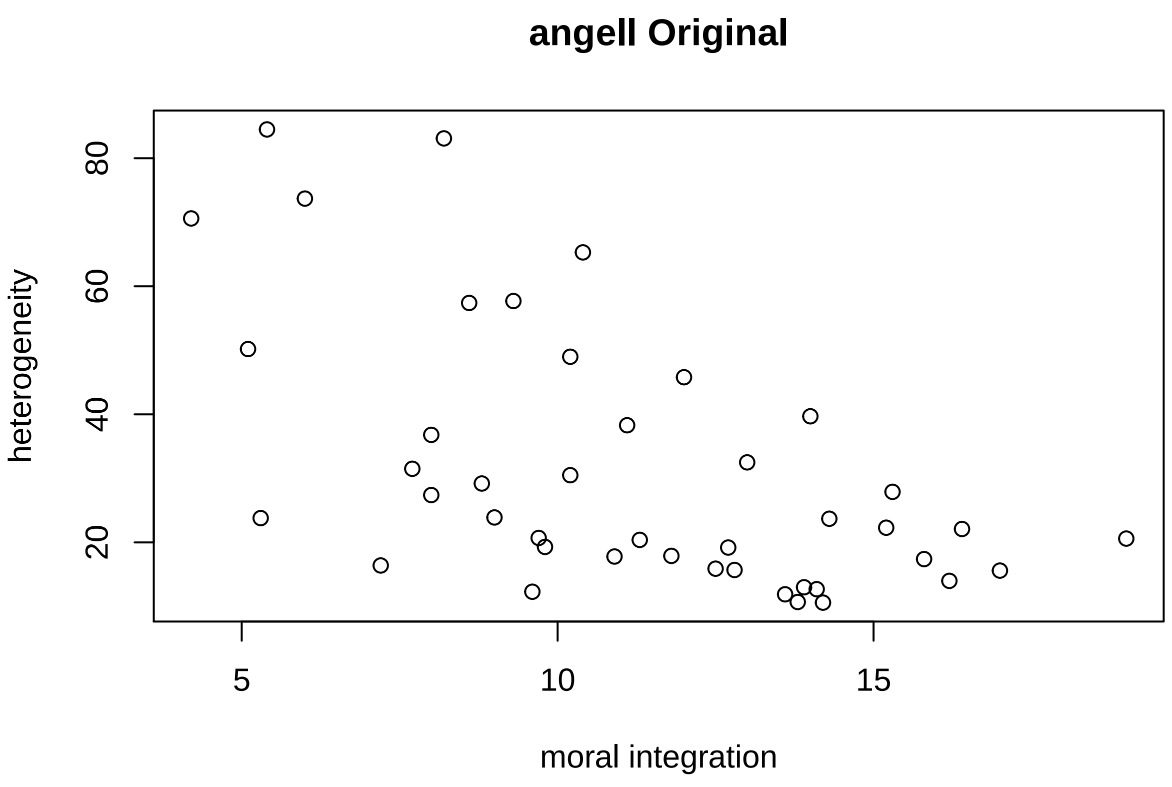


```
plot(robey$x_trans, robey$contraceptors, main="robey transformed x axis", xlab="total fertility rate", y
```

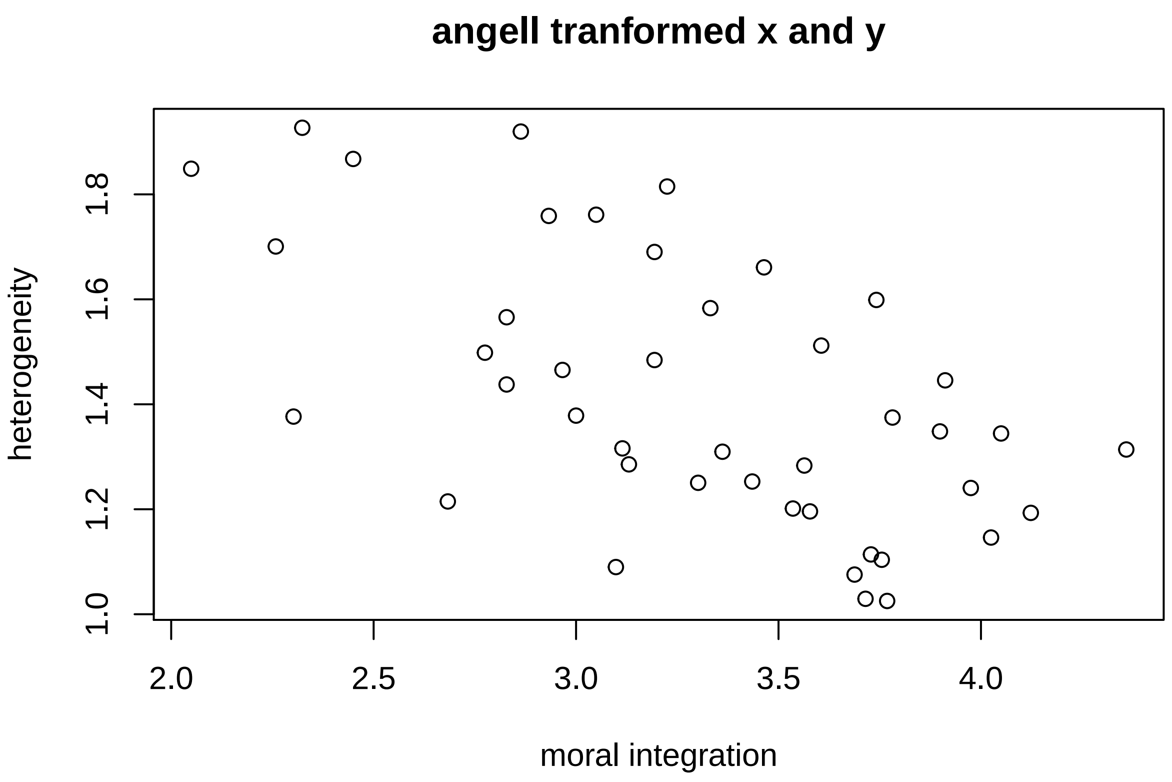


For the Robey.txt data, the original data had long x-tail on the left, suggesting that I need to scale x-values up (total fertility rate). So I raised x-values to their 1.5 power, and the result looks pretty decent.

```
angell = read.table("angell.txt")
angell$tran_moral = (angell$moralIntegration)^(1/2)
angell$tran_het = log10(angell$heterogeneity)

plot(angell$moralIntegration, angell$heterogeneity, main="angell Original", xlab="moral integration", yl
```

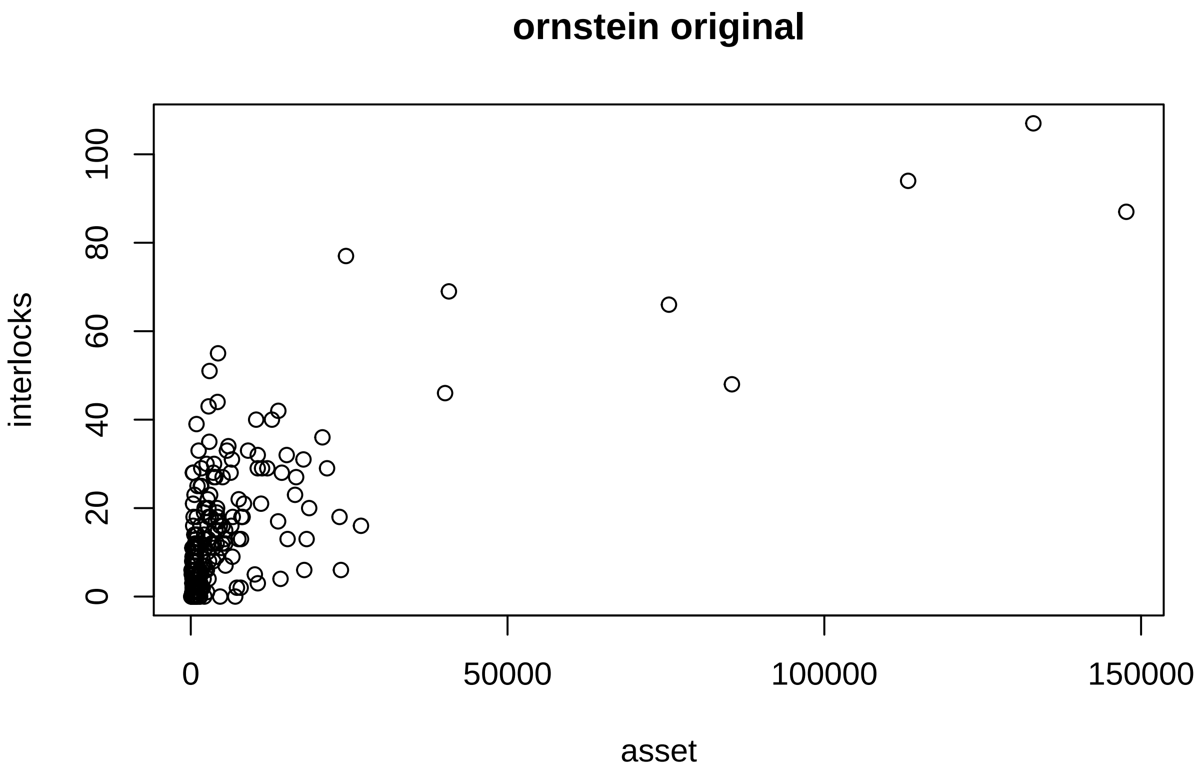


```
plot(angell$tran_moral, angell$tran_het,  main="angell tranformed x and y", xlab="moral integration", yl
```

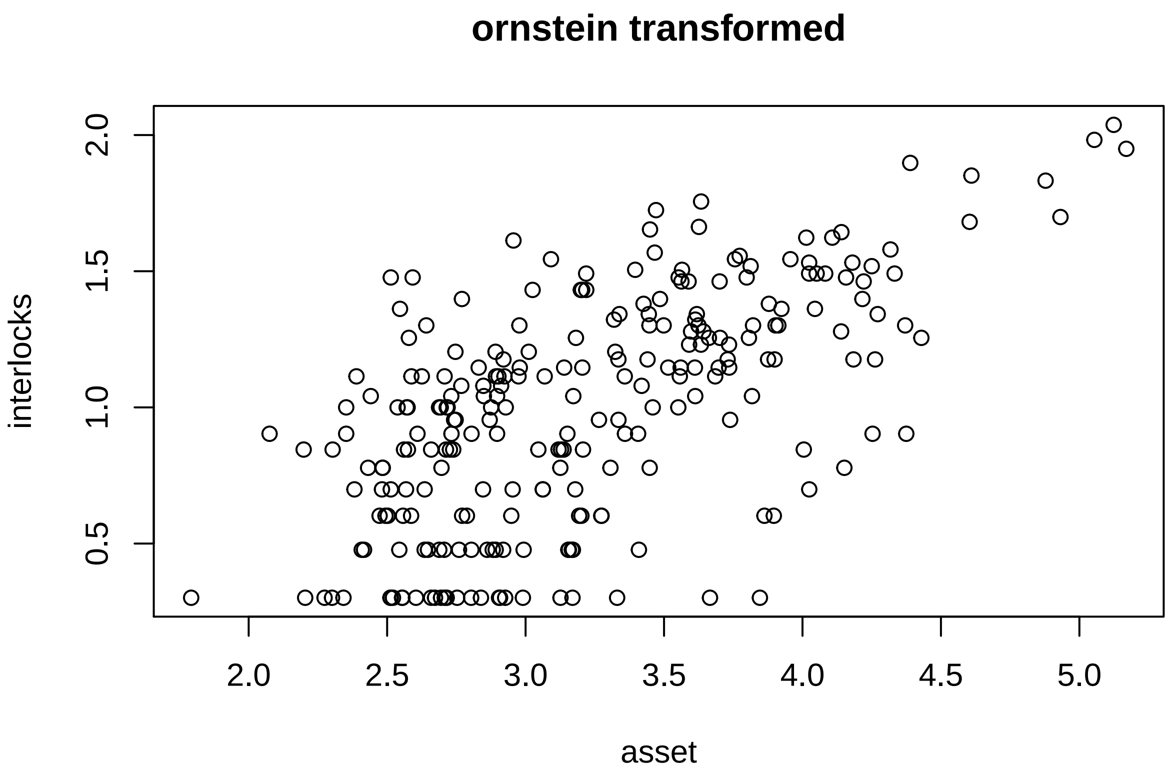


For this one it looks like we have very big variance at small x, but otherwise okay fit. So initially I tried to do box-cox. However the result did not appear to look better (or different) at all. Thus I decided to compress y values by taking the log. Then the graph appeared skewed towards the right, so I square rooted the x values, obtaining the resulting graph.

```
orn = read.table("ornstein.txt")
orn$tran_asset <- log10(orn$assets)
orn$tran_inter <- log10(orn$interlocks+2)

plot(orn$assets, orn$interlocks, main="ornstein original", xlab="asset", ylab="interlocks")
```



```
plot(orn$tran_asset, orn$tran_inter, main="ornstein transformed", xlab="asset", ylab="interlocks")
```



Here the original data was most crumbled near the origin, so I log-scaled both the x and y axis. Since the interlock column can potentially take values of 0, I added 2 to each value to prevent a negative infinity.