STAT243 Problem Set 8

Name: Chih-Hui Wang SID: 26955255

November 18, 2015

1.

(a) To compare the performance of two methods, we have to generate the simulated dataset. For the detailed of creating te datset, see (b). We seperate the data into two parts. We take the first part of the data to fit models by two methods while another part is for evaulating performance. After fitting models, we can report the absoulte prediction error by calculating the absolute difference between the true value and predicted value by two methods. For the coverage of the prediction interval, we utilize the nonparametric bootstrap. We will repeat the following procedure for many times. First, we sample n observations from the data which is used to fit model with replacement.(Assume the total number in the data for fitting model is n) Second, we get the predicted value for X in the data for evaluation. After doing the process for several times, we will have a bunch of predicted value for each X in the data for evaluation. We can construct the prediction interval from those predicted value. If we want to construct the 95% prediction interval, we can pick the 2.5% and 97.5% quantiles as the lower bound and upper bound of the interval. With the prediciton interval, we can check whether the true value is inside the interval and report the coverage of prediction interval. The pseudo code is below:

```
#Data
Generate data D with outlier from your technique.
#Setup
Seperate D into data for fitting model D1 and data for evaluting performance D2.
#Fitting model
Run two methods on D1 and compute the predicted value for D2.
#Prediciton error
Calculate the absolute difference between the above predicted value and
true value in D2 and report absolute prediction error
#Coverage of prediciton interval; m is the times of bootstraping
for i from 1 to m:
  #There are n observations in D1
  sample n observations from D1 with replacement Dboot
  Run two two methods on Dboot
  Compute the predicted value for D2 and save both predicted value in pred1 and pred2

#Get the prediction interval
Set the alpha value
Construct the prediction interval by picking the alpha/2 and 1-(alpha/2) quantile
Calculate how many obersvations in D2 falling into the corresponding interval and
report the coverage of the prediction interval
```

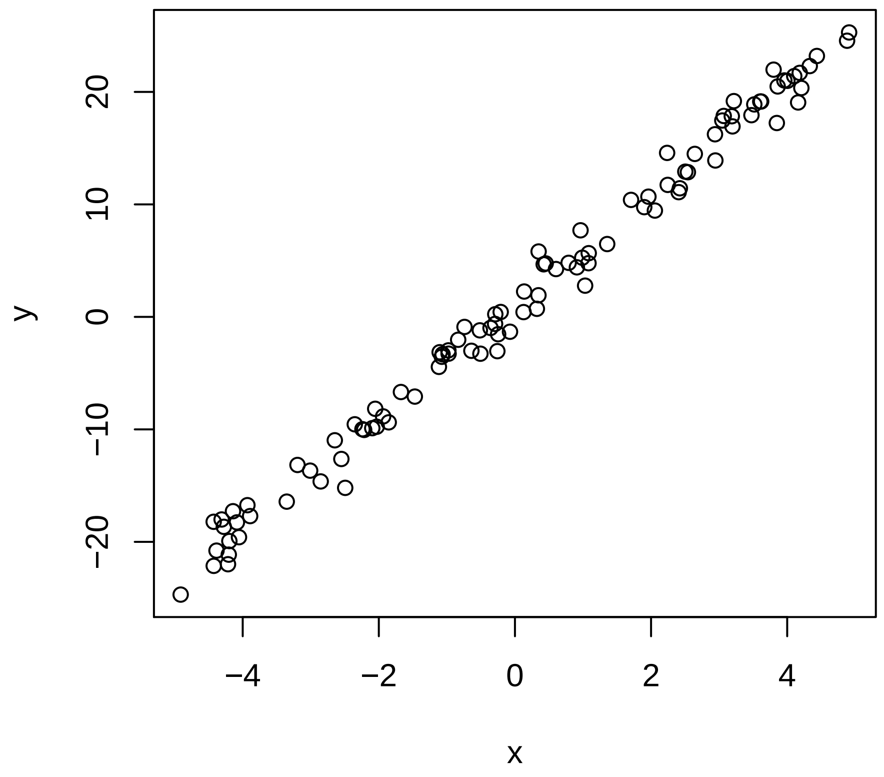
(b) Here we assume that we only have one covariate x_i. According to the regression equation, we can construct (x_i, y_i) by $y_i = \beta_0 + \beta_1 x_i + \epsilon_i i = 1, 2, \cdots, n$ where $\epsilon_i \sim N(0, \sigma^2)$. To generate the outlier, my strategy is to change the distribution of error terms. If we make their distribution heavier tail than normal distribution such as t-distribution, then we can generate outliers. Those outliers will have a huge influence on the coefficient estmation for the standard linear regression.

The following is the code for generating simulated dataset. I use t-distribution as the distribution for the error term distribution of outliers. There are several arguments which can adjust. n is the total number of data, outlier_prop is the proportion of outlier, a is the slope, b is the intercept, xrange is the range for generating x, and sigma is the variance of normal distribution.

```r
sim_data <- function(n, outlier_prop=0.01, a=5, b=1, xrange=c(-5, 5), sigma=2){
  #Construct pair of x,y
  x <- runif(n, xrange[1], xrange[2])
  y <- seq_along(x)

  #Number of outlier
  nout <- n*outlier_prop

  idx <- sample(n, n - nout)
  y[idx] <- a*x[idx] + b + rnorm(n - nout, 0, sqrt(sigma))

  #Outlier
  y[-idx] <- a*x[-idx] + b + rt(nout, 1)

  data <- cbind(x, y)
  as.data.frame(data)
}
```

Below is the example for generating data and graph.

```r
data <- sim_data(100, outlier_prop=0.05)
with(data, plot(x, y))
```



2.

(a) The density of Pareto distribution is $\dfrac{\beta\alpha^\beta}{x^{\beta+1}}$. The density of exponential distribution is $\dfrac{1}{\beta}e^{-\frac{x}{\beta}}$. The tail of Pareto distribution will decay more slowly than exponential distribution because Pareto distribution decaies in power way while exponential distribution decaies in exponential way.

(b) f is the exponential density and g is the Pareto density. To use the importance sampling for estimating $E_f(X)$, we have to generate random number x_i from g, Pareto distribution and evalute $f(x_i)$ and $g(x_i)$. Then, we can get the estimate $\hat{\mu} = \dfrac{1}{m}\sum_{i=1}^m x_i \dfrac{f(x_i)}{g(x_i)}$ where m is the total number of x_i. We have to find a way to generate random number from Pareto distribution. I use the inverse CDF method to write a function **rpareto** to draw number from Pareto distribution.

$$\begin{aligned} F_g(x) &= P(G \le x) \\ &= \int_\alpha^x \frac{\beta\alpha^\beta}{t^{\beta+1}}dt \\ &= -\frac{\alpha^\beta}{t^\beta}|_\alpha^x \\ &= 1-(\frac{\alpha}{x})^\beta \end{aligned}$$

We can generate random number u_i from uniform(0,1) distribution and let $x_i = \frac{\alpha}{(1-u_i)^{1/\beta}}$. The following codes are dpareto and rpareto:

```
#Multiple by x > 2 to make sure the function evalute when x > 2
dpareto <- function(x, alpha, beta){
  (beta*alpha^beta)/(x^(beta + 1))*(x > 2)
}

rpareto <- function(n, alpha, beta){
  u <- runif(n)
  alpha/((1 - u)^(1/beta))
}
```

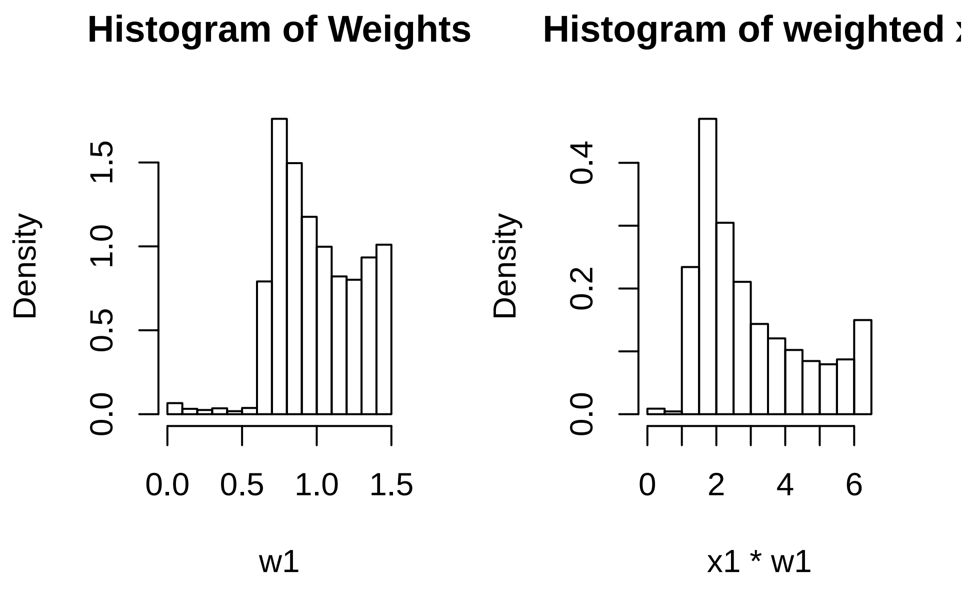
```
#Setting
alpha <- 2; beta <- 3; rate <- 1; m <- 10000
```

The steps for the following calculation are similar. First, we will generate random number x_i from g, Pareto distribution. After evaluating the density of x_i under Pareto and exponential distribution, we can compute the weight $\frac{f(x_i)}{g(x_i)}$ and plug into the formula $\hat{\mu} = \frac{1}{m}\sum_{i=1}^{m} x_i \frac{f(x_i)}{g(x_i)}$. In this case, we should get a value close to $E(X) + 2 = 3$ which is the shifted mean of exponential distribution with rate equal to 1.

```
#Shifted exponential
set.seed(0)

#E(X)
x1 <- rpareto(m, alpha, beta)
f1 <- dexp(x1 - 2, rate)
g1 <- dpareto(x1, alpha, beta)

#Weight
w1 <- f1/g1
mu_hat <- mean(x1*w1)
mu_hat

## [1] 3.010117
```

The following is the histogram of weight $\frac{f(x_i)}{g(x_i)}$ and weighted $x_i' = x_i\frac{f(x_i)}{g(x_i)}$.

```
par(mfrow=c(1, 2))
hist(w1, prob=TRUE, main="Histogram of Weights")
hist(x1*w1, prob=TRUE, main="Histogram of weighted x")
```

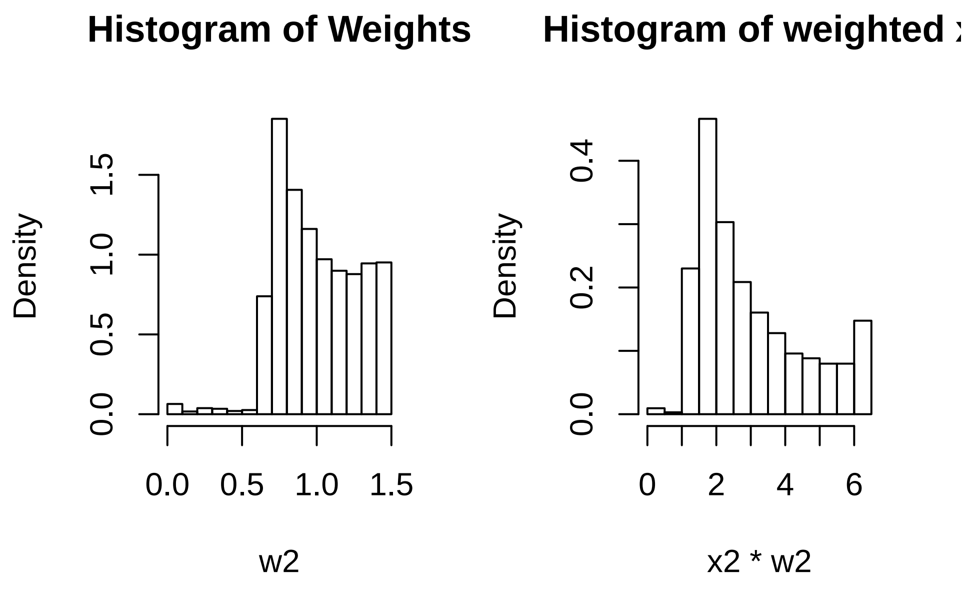


The procedure is the same as previous. We only have to change the formula by $E(\hat{X}^2) = \frac{1}{m}\sum_{i=1}^{m} x_i^2 \frac{f(x_i)}{g(x_i)}$. The value should be closed to $Var(X) + [E(X)]^2$, which is 10.

```
#E(X^2)
x2 <- rpareto(m, alpha, beta)
f2 <- dexp(x2 - 2, rate)
g2 <- dpareto(x2, alpha, beta)

#Weight
w2 <- f2/g2
xsquare_hat <- mean((x2^2)*w2)
xsquare_hat

## [1] 10.03446
```

The following is the histogram of weight $\frac{f(x_i)}{g(x_i)}$ and weighted $x_i' = x_i^2 \frac{f(x_i)}{g(x_i)}$.

```
par(mfrow=c(1, 2))
hist(w2, prob=TRUE, main="Histogram of Weights")
hist(x2*w2, prob=TRUE, main="Histogram of weighted x")
```

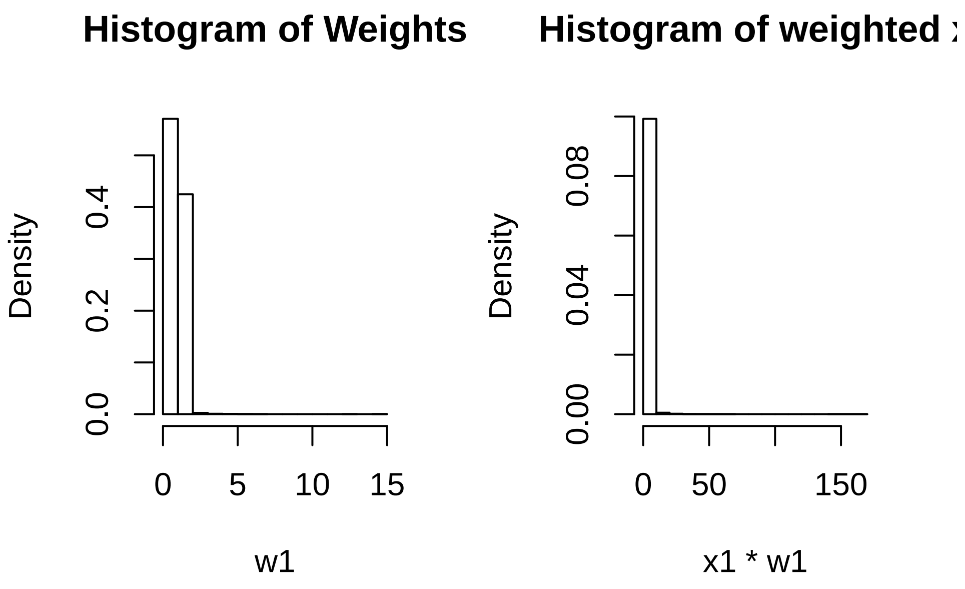


(c) Now we only have to exchange the f into Pareto distribution and g into exponential distribution. Note that when we generate the random number from exponential distribution, we have to add 2 because we should shifted by 2. The estimates should be closed to $E(X) = \frac{\beta\alpha}{\beta - 1} = 3$.

```
#Pareto
set.seed(0)

#E(X)
x1 <- rexp(m, rate) + 2
f1 <- dpareto(x1, alpha, beta)
g1 <- dexp(x1 - 2, rate)

#Weight
w1 <- f1/g1
mu_hat <- mean(x1*w1)
mu_hat

## [1] 2.962398
```

In this case, the histogram indicate that there are outliers in the data, which is the reason make the estimates will be not stable. It is becasue the exponential distribution have a lighter tail than Pareto distribution, which will let the weight $\frac{f(x_i)}{g(x_i)}$ become large. It will enlarge the variance when we do the estimation.

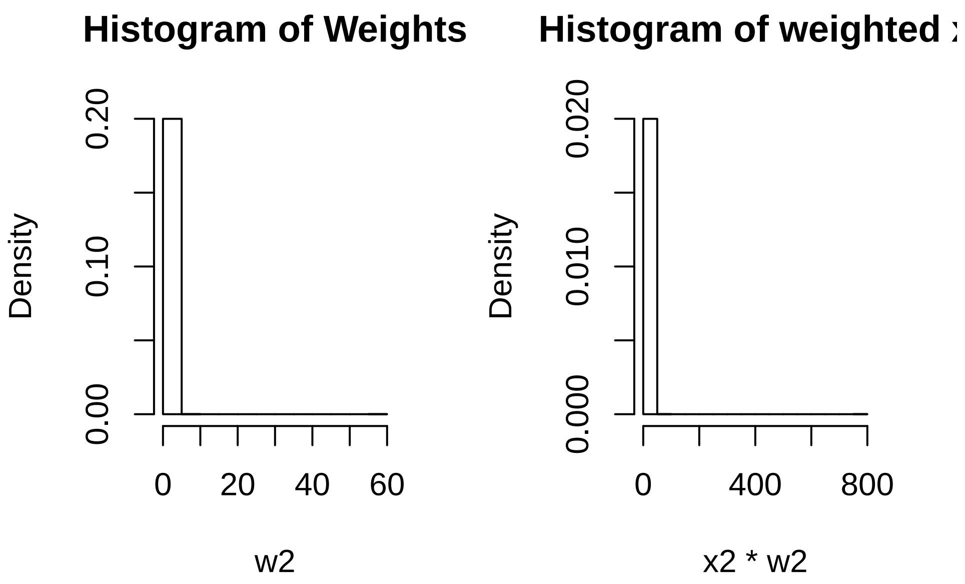
```
par(mfrow=c(1, 2))
hist(w1, prob=TRUE, main="Histogram of Weights")
hist(x1*w1, prob=TRUE, main="Histogram of weighted x")
```



The estimates should be closed to $E(X^2) = Var(X) + [E(X)]^2 = \frac{\beta\alpha^2}{(\beta-1)^2(\beta-2)} + (\frac{\beta\alpha}{\beta-1})^2 = 12.$

```
#E(X^2)
x2 <- rexp(m, rate) + 2
f2 <- dpareto(x2, alpha, beta)
g2 <- dexp(x2 - 2, rate)

#Weight
w2 <- f2/g2
xsquare_hat <- mean((x2^2)*w2)
xsquare_hat

## [1] 10.41318
```

```
par(mfrow=c(1, 2))
hist(w2, prob=TRUE, main="Histogram of Weights")
hist(x2*w2, prob=TRUE, main="Histogram of weighted x")
```



3.

(a)

$$l(\beta) = \frac{-n}{2}\log 2\pi - \frac{1}{2}\sum_{i=1}^{n}(z_i - X_i^T\beta)^2$$

E-step:

$$\begin{aligned} Q(\beta) = E[l(\beta)] &= \frac{-n}{2}\log 2\pi - \frac{1}{2}\big(\sum_{i=1}^{n} E(z_i^2|y_i,\beta) - (X_i^T\beta)E(z_i|y_i,\beta) + X_i^T\beta\beta^T X_i\big) \\ &= \frac{-n}{2}\log 2\pi - \frac{1}{2}\sum_{i=1}^{n} Var(z_i|y_i,\beta) - \frac{1}{2}\big(\sum_{i=1}^{n}[E(z_i|y_i,\beta)]^2 - (X_i^T\beta)E(z_i|y_i,\beta) + X_i^T\beta\beta^T X_i\big) \\ &= \frac{-n}{2}\log 2\pi - \frac{1}{2}\sum_{i=1}^{n} Var(z_i|y_i,\beta) - \frac{1}{2}\sum_{i=1}^{n}\big(E(z_i|y_i,\beta) - X_i^T\beta\big)^2 \end{aligned}$$

M-step:

$$\begin{aligned} \frac{\partial Q(\beta)}{\partial\beta} &= \sum_{i=1}^{n}\big(E(z_i|y_i,\beta) - X_i^T\beta\big)X_i \\ 0 &= \sum_{i=1}^{n} X_i^T E(z_i|y_i,\beta) - \sum_{i=1}^{n} X_i^T X_i\beta \\ \text{Update} \to \beta_{t+1} &= \frac{\sum_{i=1}^{n} X_i^T E(z_i|y_i,\beta_t)}{\sum_{i=1}^{n} X_i^T X_i} \end{aligned}$$

where $E(z_i|y_i,\beta) = \begin{cases} X_i^T\beta + \dfrac{\phi(-X_i^T\beta)}{1-\Phi(-X_i^T\beta)}, & y_i = 1 \\ X_i^T\beta - \dfrac{\phi(-X_i^T\beta)}{\Phi(-X_i^T\beta)}, & y_i = 0 \end{cases}$

It is actually the solution of linear regression for $E(z_i|y_i,\beta)$ is the dependent variable while x_i are the independent variables.

(b) Set the initial value to (0, 0, 0, 0).

(c) To generate the data with $\hat{\beta}_1/se(\hat{\beta}_1) \approx 2$, I use glm with binomial family and link probit to try different beta parameter and come out with a beta with the value $\hat{\beta}_1/se(\hat{\beta}_1)$ close to 2.

```
set.seed(0)

#beta to generate data
mybeta <- c(1, -20, 3, 0)

#Setup
n <- 100
p <- length(mybeta) - 1
X <- matrix(rnorm(n*p), ncol=p)
z <- rnorm(n, X*mybeta)
y <- (z > 1)*1

#Use glm to fit probit model
fit <- glm(y ~ X, family=binomial(link="probit"))
beta_1_hat <- summary(fit)$coefficients[2, 1]
se_1_hat <- summary(fit)$coefficients[2, 2]

#Should be close to 2
beta_1_hat/se_1_hat

## [1] 2.040266
```

First, I write a function to compute the term $E(z_i|y_i,\beta)$. In **my_em**, I use **lm** function for updating beta. My criteria for convergence is that if difference between new beta and old beta is small than the tolerance **tol**, then the functon will stop the iteration. It will output the coefficient and the number of iteration.

```
#Expectation of zi when yi equal to 1 and 0
fun <- function(X, y, beta){
  mu <- cbind(1, X) %*% beta
  ifelse(y == 1,
         mu + dnorm(-mu)/(1 - pnorm(-mu)),
         mu - dnorm(-mu)/pnorm(-mu))
}

#EM
my_em <- function(X, y, beta, tol=1e-8){
  #Initial setup
  beta_old <- beta
  expect_z <- fun(X, y, beta_old)
  fit <- lm(expect_z ~ X)
  beta_new <- coef(fit)
  iter <- 1
  #Update step
  while(sqrt(sum((beta_old - beta_new)^2)) > tol){
    beta_old <- beta_new
    expect_z <- fun(X, y, beta_old)
    fit <- lm(expect_z ~ X)
    beta_new <- coef(fit)
    iter <- iter + 1
  }
  list(beta_new, iter)
}

#Initial value of beta
beta_init <- c(0, 0, 0, 0)
out <- my_em(X, y, beta_init)
out

## [[1]]
## (Intercept)          X1          X2          X3
##  -0.3880520   0.3151817  -0.1142404   0.2350182
##
## [[2]]
## [1] 29
```

```
#Should be close
rbind(coef(fit), out[[1]])

##      (Intercept)        X1         X2        X3
## [1,]  -0.3880516 0.3151810 -0.1142394 0.2350182
## [2,]  -0.3880520 0.3151817 -0.1142404 0.2350182
```

(d) We can maximize the expected log-likelihood $Q(\beta) = E[l(\beta)] = \sum_{i=1}^{n} \left(E(z_i|y_i, \beta) - X_i^T \beta\right)^2$. We can find the estimates are quite different from the EM algorithm and glm.

```
#Log-likelihood
loglike <- function(beta){
  sum((fun(X, y, beta) - cbind(1, X) %*% beta)^2)

}
optim(beta_init, loglike, method="BFGS")

## $par
## [1] -0.27350966  0.22279365 -0.08126941  0.16675843
##
## $value
## [1] 57.0409
##
## $counts
## function gradient
##       28        8
##
## $convergence
## [1] 0
##
## $message
## NULL
```

4.

This is the function we want to minimize in the helical.R.

```r
#functions in helical.R
theta <- function(x1, x2){
  atan2(x2, x1)/(2*pi)
}

f <- function(x) {
  f1 <- 10*(x[3] - 10*theta(x[1], x[2]))
  f2 <- 10*(sqrt(x[1]^2 + x[2]^2) - 1)
  f3 <- x[3]
  return(f1^2 + f2^2 + f3^2)
}
```

I will generate data by holding one variable constant(for 3 points -5, 0, 5) and varying other variables from -10 to 10. Therefore, it has 9 different scenarios. For the purpose of modularity, I write a function to generate data and process them into the form we can use for creating plot.

```r
#functions that generate data
data_generate <- function(fix_value, fix_position, limit){
  n <- length(limit)^2

  #Empty matrix for filling value
  m <- matrix(0, nrow=n, ncol=4)
  colnames(m) <- c("x1", "x2", "x3", "value")

  #Fill in the x1, x2, x3
  m[, fix_position] <- rep(fix_value, n)
  m[, -c(fix_position, 4)] <- as.matrix(expand.grid(limit, limit))

  #Compute the value
  m[, 4] <- apply(m[, -4], 1, f)

  as.data.frame(m)
}
```

As mentioned previously, I set the range from -10 to 10 and for the constant value, it will be -5, 0, 5 respectively.

```r
#setting
limit <- -10:10

#Constant x1 for 0, 5, -5
dt <- data_generate(0, 1, limit)
dt2 <- data_generate(5, 1, limit)
dt3 <- data_generate(-5, 1, limit)
```

```
#Constant x2 for 0, 5, -5
dt4 <- data_generate(0, 2, limit)
dt5 <- data_generate(5, 2, limit)
dt6 <- data_generate(-5, 2, limit)

#Constant x3 for 0, 5, -5
dt7 <- data_generate(0, 3, limit)
dt8 <- data_generate(5, 3, limit)
dt9 <- data_generate(-5, 3, limit)
```

Here I will use the **geom_tile** in ggplot to present how the function behaves under 9 distinct scenarios. The darker the region is, the higher the function vale is. I use **%+%** to update data render to the plot as well as change the title.

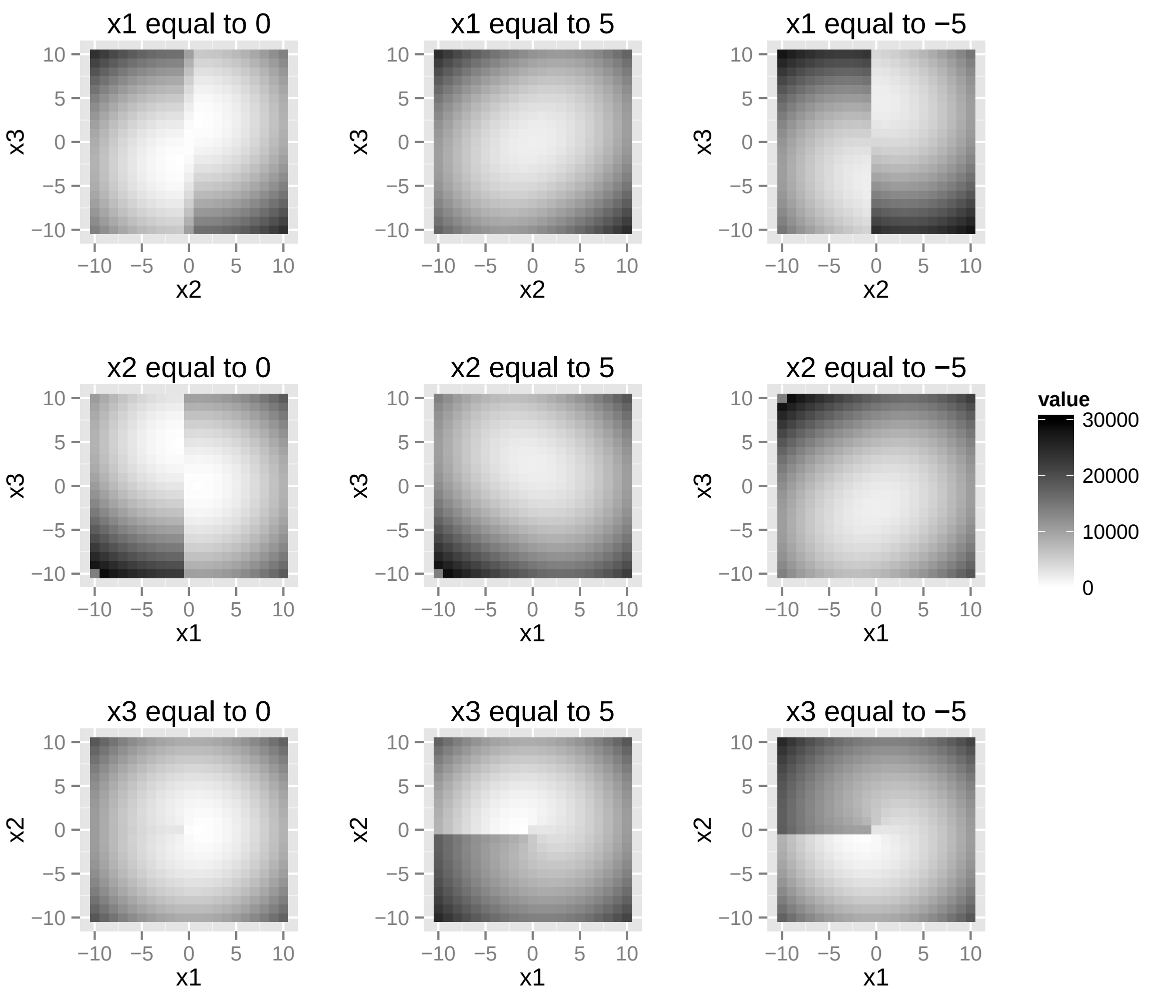
```
#packages
library(ggplot2)

## Loading required package:  methods

library(gridExtra)

## Warning:  package 'gridExtra' was built under R version 3.2.2

library(grid)

#Graph for constant x1
g1 <- ggplot(dt, aes(x=x2, y=x3, fill=value)) +
  geom_tile() +
  scale_fill_continuous(low="white", high="black", limit=c(0, 30000)) +
  labs(title="x1 equal to 0") +
  theme(legend.margin=unit(1, "cm"))
g2 <- g1 %+% dt2 + labs(title="x1 equal to 5")
g3 <- g1 %+% dt3 + labs(title="x1 equal to -5")

#Graph for constant x2
g4 <- g1 %+% dt4 + aes(x=x1) + labs(title="x2 equal to 0")
g5 <- g1 %+% dt5 + aes(x=x1) + labs(title="x2 equal to 5")
g6 <- g1 %+% dt6 + aes(x=x1) + labs(title="x2 equal to -5")

#Graph for constant x3
g7 <- g1 %+% dt7 + aes(x=x1, y=x2) + labs(title="x3 equal to 0")
g8 <- g1 %+% dt8 + aes(x=x1, y=x2) + labs(title="x3 equal to 5")
g9 <- g1 %+% dt9 + aes(x=x1, y=x2) + labs(title="x3 equal to -5")
```

For simplicity, I want to make all the plot share one legend. I find the function **grid_arrange_shared_legend** from ggplot2 github. I adjust it a little bit to let the legend shown on the right side of the plot.

```
#Reference: https://github.com/hadley/ggplot2/wiki/Share-a-legend-between-two-ggplot2-graphs
grid_arrange_shared_legend <- function(...) {
  plots <- list(...)
  g <- ggplotGrob(plots[[1]])$grobs
  legend <- g[[which(sapply(g, function(x) x$name) == "guide-box")]]
  lwidth <- sum(legend$widths)
  grid.arrange(
    do.call(arrangeGrob, lapply(plots, function(x)
      x + theme(legend.position="none"))),
    legend,
    ncol = 2,
    widths = unit.c(unit(1, "npc") - lwidth, lwidth))
}
```

Overall, the value of the function minimize when two non-constant varable are around zero. It seems to be marginally symmetric when $x_1 = 0, x_2 = 0, x_1 = -5$.

```
#Final plot
grid_arrange_shared_legend(g1, g2, g3, g4, g5, g6, g7, g8, g9, ncol=3)
```



I try four different initial values here for (0, 0, 0), (5, 0, 0), (0, 5, 0), (0, 0, 5) and use the **optimx** function(in the optimx library) to perform 5 minimizing methods. There are several local minimum points and most of them are closed to (1, 0, 0). The results show that some of the methods such as nlm perform worsely in this optimization. For the Nelder-Mead method, 4 initial values converge to the almost same result.

```
library(optimx)

## Warning:  package 'optimx' was built under R version 3.2.2
```

```r
usemethod <- c("Nelder-Mead", "BFGS", "nlm", "nlminb", "L-BFGS-B")
try1 <- optimx(c(0, 0, 0), f, method=usemethod)

## Warning in max(logpar):  no non-missing arguments to max; returning -Inf
## Warning in min(logpar):  no non-missing arguments to min; returning Inf

try2 <- optimx(c(5, 0, 0), f, method=usemethod)
try3 <- optimx(c(0, 5, 0), f, method=usemethod)
try4 <- optimx(c(0, 0, 5), f, method=usemethod)

all <- unique(rbind(try1[, 1:4], try2[1:4], try3[1:4], try4[1:4]))
all[order(all$value), ]

##                         p1            p2            p3        value
## L-BFGS-B1     1.000000e+00  0.000000e+00  0.000000e+00 0.000000e+00
## BFGS          1.000000e+00  0.000000e+00  0.000000e+00 3.155444e-28
## BFGS1         1.000000e+00  0.000000e+00  0.000000e+00 4.930381e-28
## nlminb1       1.000000e+00 -2.372401e-12 -2.201929e-12 4.557975e-22
## nlminb2       1.000000e+00  7.987537e-11  1.369421e-10 4.368188e-20
## BFGS3         1.000000e+00  5.447509e-10  8.961821e-10 3.393023e-18
## BFGS2         1.000000e+00  2.384227e-09  3.913394e-09 6.483568e-17
## nlminb3       1.000000e+00 -1.254058e-09 -2.996888e-09 1.110549e-16
## L-BFGS-B2     1.000000e+00 -7.380766e-08 -1.175603e-07 1.796494e-14
## L-BFGS-B3     1.000000e+00 -5.692813e-07 -9.622277e-07 1.416657e-12
## nlm1          1.000000e+00 -1.074525e-06 -1.689871e-06 2.896830e-12
## nlm2          9.999995e-01 -8.223121e-05 -1.300794e-04 1.700897e-08
## Nelder-Mead1  9.997726e-01 -3.948626e-06 -7.301922e-05 5.623899e-06
## Nelder-Mead2  9.997559e-01  6.047497e-04  7.770766e-04 9.998543e-06
## Nelder-Mead   9.999783e-01  2.730698e-03  4.284640e-03 1.876851e-05
## Nelder-Mead3  9.995364e-01 -3.223910e-03 -4.468572e-03 8.518097e-05
## nlm           0.000000e+00  0.000000e+00  0.000000e+00 1.000000e+02
## nlm3         -4.234688e-07  8.994279e-03  5.000000e+00 7.481718e+02
```