Problem Set 8

Alexander Brandt
SID: 24092167

December 7 2015

Friendly Collaborators: Milos Atz, Alex Ojala and Guillame Baquiast.

1

1.1

We generate several data sets given a linear model, with a random variable signifying noise. We will input the x_i and y_i into both of the models we wish to test. For each model, we will compute the prediction error and the coverage as follows:

Prediction error is the absolute difference between the data and the predicted value, and the coverage can be generated in a non-parametric fashion by selecting, with replacement, n values (n should not be too small) and use order statistics to generate the prediction interval of a given percentage (this is the boostrap).

To compare these methods to see which is "better" we can generate a mean and variance of both preidiction error and coverage. We will choose the method which minimizes the variance.

1.2

We choose some linear function f that we wish to recapitulate with our analysis. The x-values are generated uniformly on a given range, and the y-values are determined to be $y_i = f(x_i) + \epsilon_i$. ϵ_i has some large probability of being a "minor" error, or a small probability of begin a "major" error, giving rise to an outlier value. We simulate a random variable on the interval 0 to 1, and if it is below some small value (.05, .01, etc.), we greatly increase the value of the magnitude of the error (so if normally ϵ_i is N(0,1), now it will be something like N(0,10).

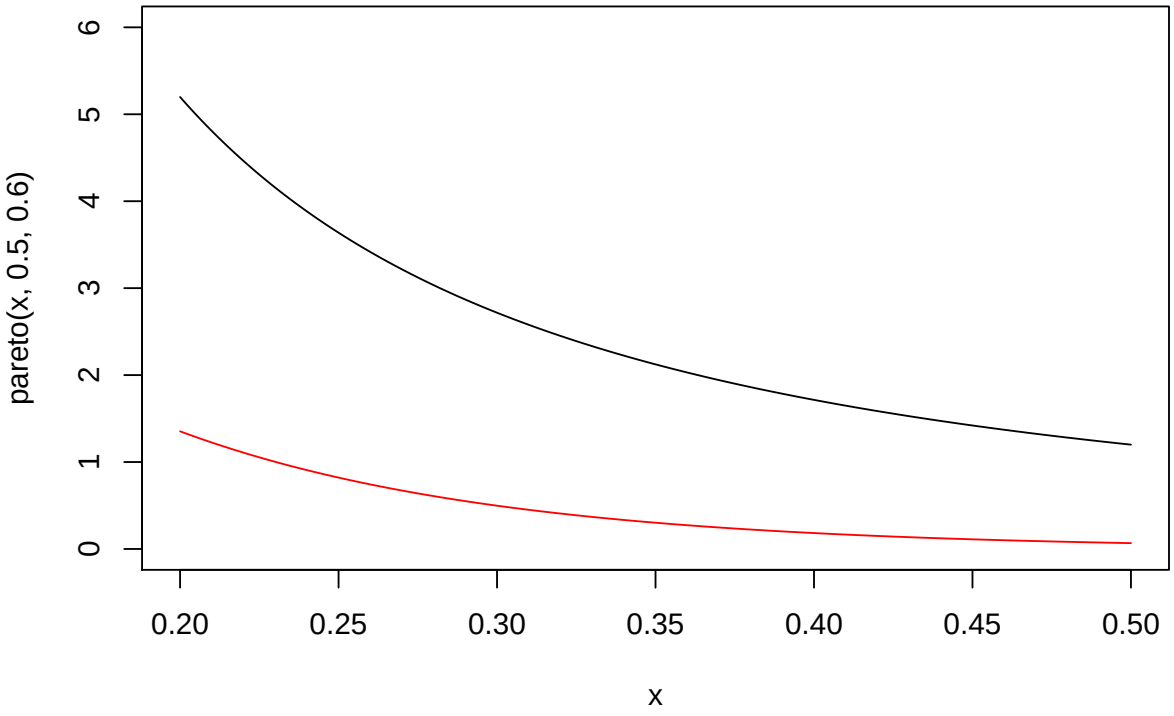
2

2.1

The pareto decays more slowly than the exponential. Note: pareto here is graphed in black.

```
pareto <- function(x, alpha, beta) {
  return(beta * (alpha^beta) / (x^(beta+1)))
}

expd <- function(x, lambda, xshift) {
  return( lambda * exp( -lambda * (x - xshift) ))
}

alpha <- 2
```

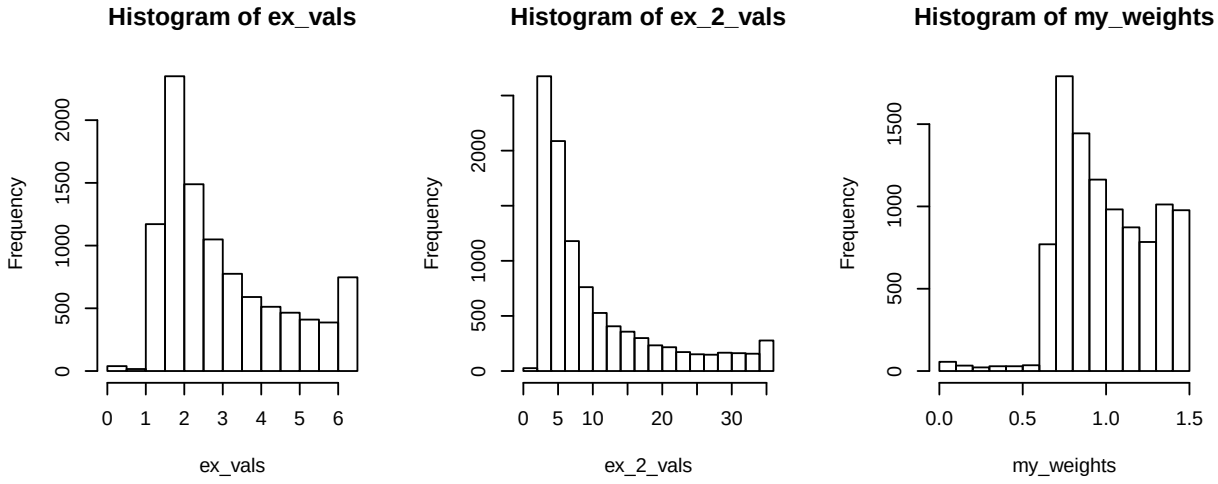
```r
beta <- 3

x <- seq(from=.2, to=.5, by=.001)
plot(x, pareto(x, .5, .6), type='l',ylim=c(0,6))
lines(x, expd(x, 10, 0), col=26)
```



2.2

```r
# Rewriting our functions just a bit to take into account the problem
# statement:

pareto <- function(x, alpha, beta) {
  return(beta * (alpha^beta) / (x^(beta+1)))
}

inverse_cdf_pareto <- function(p, alpha, beta) {
  return( alpha * ((1 - p) ^ (-1 / beta)) )
}

expd <- function(x, lambda, xshift) {
  return( (x >= 2) * lambda * exp( -lambda * (x - xshift) ))
}

# This is JUST the inverse cdf for the specific function asked
```

```r
inverse_cdf_myexp <- function(p) {
  return(2 - log(1-p))
}

alpha <- 2
beta <- 3

m <- 10000
us <- runif(n=m)

# Sample our values
vals <- inverse_cdf_pareto(us, alpha=2, beta=3)

# Define our functions f and g
f <- expd(vals, lambda = 1, xshift=2)
g <- pareto(vals, alpha=alpha, beta=beta)

# Contruct our weights, and expectations of X and X^2
my_weights <- f / g
ex_vals <- vals * my_weights
ex_2_vals <- vals^2 * my_weights
EX <- mean(ex_vals)
EX2 <- mean(ex_2_vals)

attach(mtcars)
par(mfrow=c(1,3))

# Plot the histograms
hist(ex_vals)
hist(ex_2_vals)
hist(my_weights)
```




```r
# Display the values
EX
```

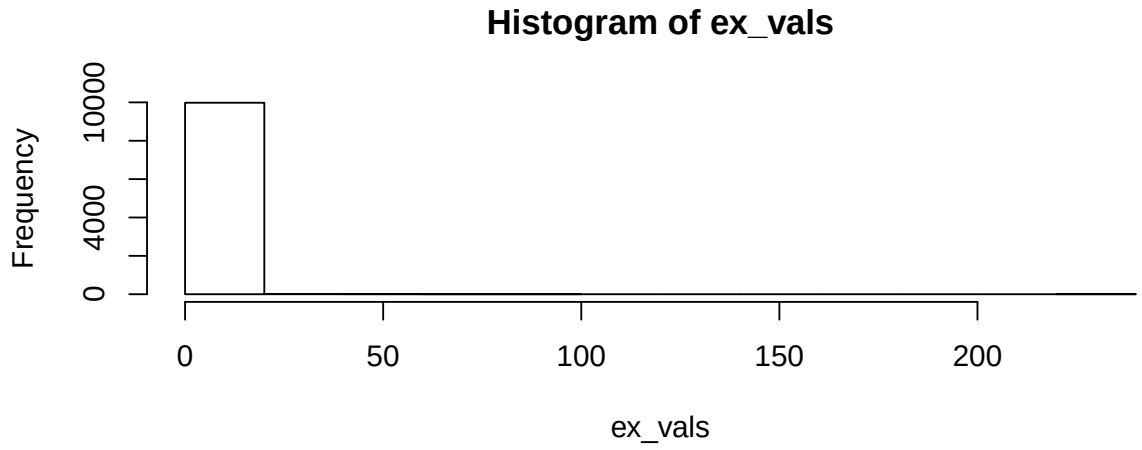
```
## [1] 3.014705
```

```
EX2

## [1] 10.07258
```

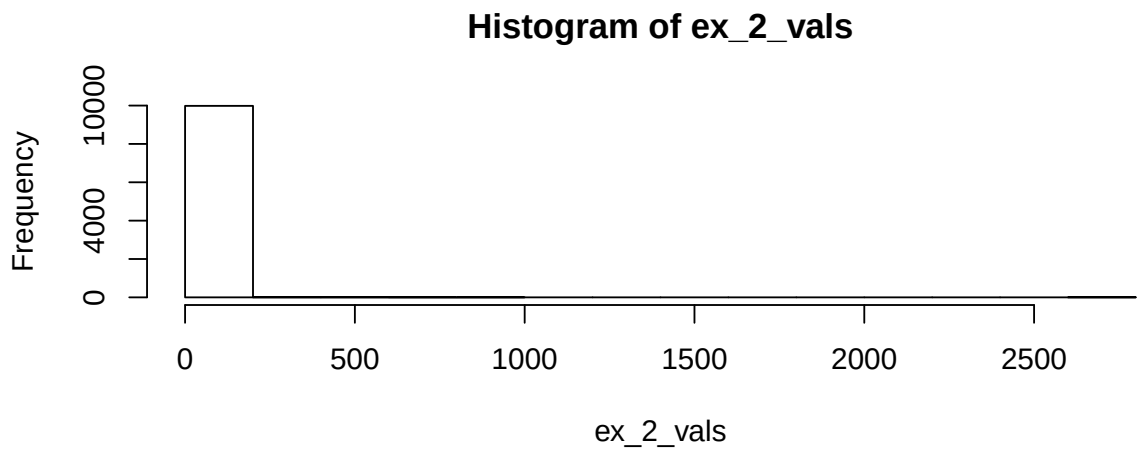
Because we are attempting to perform an importance sampling, we can see our higher weights are sampled more frequently than would be expected from an agnostic sampling of the distribution (in the first scenario).

2.3

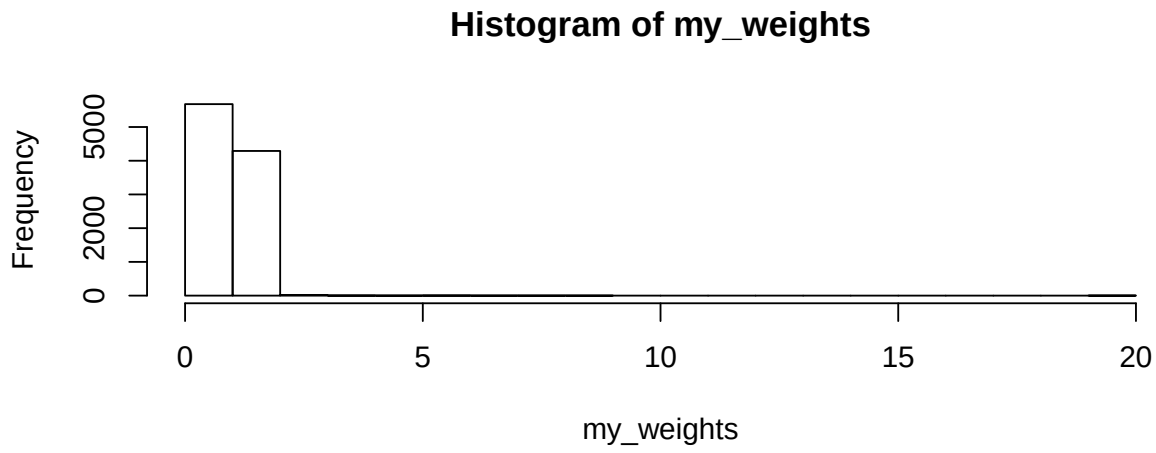
```
# c)
# Almost exactly the same as above, now we just switch our
# definitions of g and f:

m <- 10000
us <- runif(n=m)

vals <- inverse_cdf_myexp(us)

g <- expd(vals, lambda = 1, xshift=2)
f <- pareto(vals,alpha=alpha,beta=beta)

my_weights <- f / g
ex_vals <- vals * my_weights
ex_2_vals <- vals^2 * my_weights
EX <- mean(ex_vals)
EX2 <- mean(ex_2_vals)

hist(ex_vals)
```




```
hist(ex_2_vals)
```




```
hist(my_weights)
```




```
EX
```

```
## [1] 2.942006
```

```
EX2
```

```
## [1] 10.26565
```

The weights here are not as extreme/deviant from the distribution.

3

3.1

Starting for the log likelihood definition given by Harold's section notes:

$$l(\theta, Y) = \frac{n}{2}\log\sigma^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{c}(Y_{obs,i} - \mu)^2 - c$$

Take $Y = Z$, $\theta = \beta$, and $\sigma = 1$, from the problem statement. Then:

$$l(\theta, Y) = -\frac{1}{2}\sum_{i=1}^{c}(z_i - x_i^T\beta)^2 - c$$

So now we can complete:

$$\begin{aligned}
Q(\beta|\beta^{(t)}) &= E[l(\beta, Y)|y_{obs}, \beta^{(t)}] \\
&= E[\sum_{i=1}^{c}(z_i - x_i^T\beta)^2|Y = y_obs, \beta = \beta^{(T)}] + c \\
&= \frac{1}{2}E[(z - x\beta)^T(z - x\beta)] \\
&= -\frac{1}{2}\beta^T x^T x\beta + E[\beta^T x^T z] + c' \\
&= \frac{1}{2}\beta^T x^T x\beta + \beta^T x^T E[z] + c'
\end{aligned}$$

Our observations y_i will have two values, 0 or 1. Conditioning on either, our z_i values will be distributed according to the truncated normal distribution for the mean $= x_i^T\beta^{(t)}$ and the standard deviation of 1 for ranges from -infinity to 0, and 0 to infinity, respectively. Then, differential Q, with respect to beta:

$$\begin{aligned}
\frac{\partial Q}{\partial\beta} &= -\beta^T X^T X + (E[x^T z])^T = 0 \\
(\beta^T x^T x)^T &= E[x^T z]
\end{aligned}$$

So:

$$\beta^{(t+1)} = (x^T x)^{-1} x^T E[z]$$

Where z is advanced from our problem statement for $y_i = 0$:

$$E[z]^{(t+1)} = x_i^T\beta^{(t)} - \frac{\phi(x_i^T\beta^{(t)})}{\Phi(-x_i^T\beta^{(t)})}$$

and for when $y_i = 1$:

$$E[z]^{(t+1)} = x_i^T\beta^{(t)} + \frac{\phi(x_i^T\beta^{(t)})}{1 - \Phi(-x_i^T\beta^{(t)})}$$

These form the basis for the implementation of the E/M algorithm that follows.
Citation: Harold's notes, Wikipedia, and lecture notes from STAT 250 at UCLA (Prof. Baines).

3.2

I propose initializing all the $\beta_i = 0$. These, signifies a starting probability of 50% for each occurence.

3.3

```r
# c) the EM algorithm implementation described above, keeping track of iterations
# for the later part(s) of the question
Probit_EM <- function(X, y, beta, epsilon) {
    iter <- 0
    mean <- X %*% beta
    converge <- FALSE

    while (!converge) {
        Ez <- mean - (1-y)*dnorm(mean)/pnorm(-mean) + y*dnorm(mean)/(1-pnorm(-mean))
        former_beta <- beta
        beta <- solve(t(X) %*% X) %*% t(X) %*% Ez
        mean <- X %*% beta

        iter <- iter + 1
        # An arbitrary convergence criterea.
        if (max(abs(beta - former_beta)) < epsilon ) {
             converge <- TRUE
        }
    }
    return(c(beta, iter))

}

# Choose beta as allows us to have the needed SE ratio, and then]
# build our inputs based on the probit

beta <- c(3,4,0,0)
X <- cbind(1, rnorm(n = 100), rnorm(n = 100), rnorm(n = 100))
mean <- X %*% beta
Z <- rnorm(mean,sd=1)
y <- (Z > 0)
iter <- 0

pem <- Probit_EM(X, y, beta, .00001)

# Show our resulting betas

print("Iterations, and betas, respectively:")

## [1] "Iterations, and betas, respectively:"

print(pem[1:4])

## [1] -0.24295412  0.02624449  0.14998443  0.09184392

print(pem[5])

## [1] 15
```

3.4

Here we write in the log-likelihood directly and maximize. The EM converges faster. The details for the standard error are shown below.

```r
# d) how to check answer by maximizing the log-likelihood
# Defining our log-likelihood for the probit

llp <- function(beta, x, y) {
  mean <- x %*% beta
  loglike <- sum(y * log(pnorm(mean)) + (1-y) * log(pnorm(-mean)))
  return(loglike)
}

# Perform the optimization (fnscale = -1 gives the maximum), using BFGS

llp_optim = optim(beta, llp, x=X, y=y, control=list(fnscale=-1, maxit=1000, reltol=.00001), method="BFGS
llp_optim$par
```

```
## [1] -0.24304582  0.02624396  0.14988449  0.09178409
```

```r
llp_optim$counts
```

```
## function gradient
##       38        9
```

```r
# A brief illustration showing how we found the beta1 / se(beta1) --
# Also confirming both EM and LL solutions are correct with the GLM, and showing
# that we have the correct se ratio.

glm_probit <- glm(y ~ X[,2] + X[,3] + X[,4], family = binomial(link = "probit"))
glm_probit$coefficients
```

```
## (Intercept)      X[, 2]      X[, 3]      X[, 4]
## -0.24295458  0.02623951  0.14998252  0.09184285
```

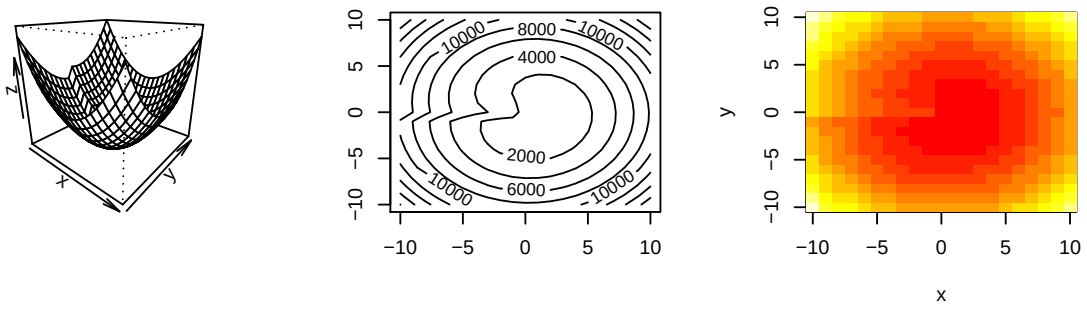
```r
summary(glm_probit)
```

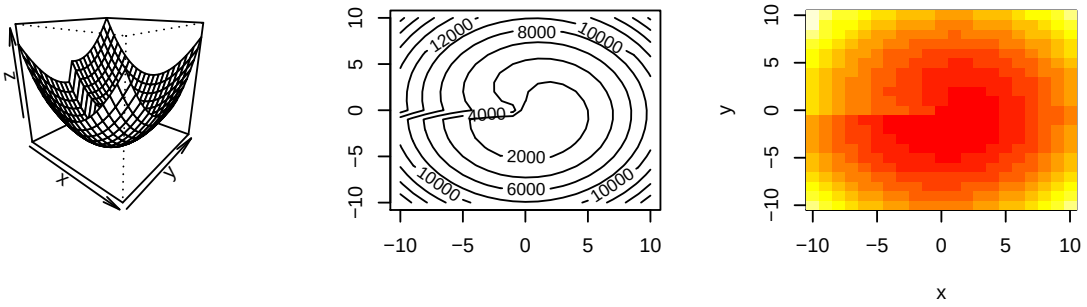
```
##
## Call:
## glm(formula = y ~ X[, 2] + X[, 3] + X[, 4], family = binomial(link = "probit"))
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.3009  -1.0057  -0.8737   1.2723   1.5879
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.24295    0.13334  -1.822   0.0685 .
## X[, 2]       0.02624    0.12290   0.213   0.8309
## X[, 3]       0.14998    0.12573   1.193   0.2329
## X[, 4]       0.09184    0.13011   0.706   0.4803
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 133.75  on 99  degrees of freedom
## Residual deviance: 131.69  on 96  degrees of freedom
## AIC: 139.69
##
## Number of Fisher Scoring iterations: 4
```

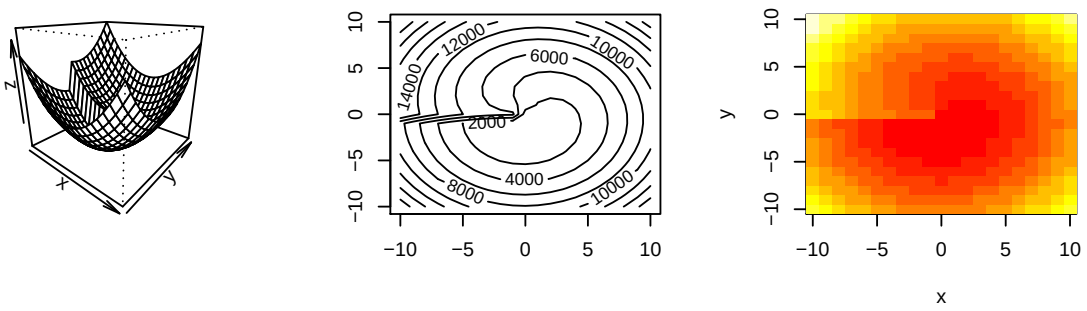
4

We print a few graphs for the last coordinate value fixed at -1, -2, and -3 (the negative values seemed to give rise to the helical shape I believe was intended). NLM and OPTIM give results in common, but vary both in terms of total number of minimia given as well as minima chosen. The global minimum seems to be at (1, 0, 0).

```
attach(mtcars)

## The following objects are masked from mtcars (pos = 3):
##
##     am, carb, cyl, disp, drat, gear, hp, mpg, qsec, vs, wt

par(mfrow=c(3,3))

# Code copy + pasted from Chris' problme statement

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)
}

# Show three graphs for three different z conditions

print("For a constant z of -1, -2, and -3...")

## [1] "For a constant z of -1, -2, and -3..."

for (q in 1:3) {
    x=seq(-10,10,1)
    y=seq(-10,10,1)
    x_len= length(x)
    y_len = length(y)
    z=array(0,dim=c(x_len,y_len))

    for (i in 1:x_len) for (j in 1:y_len) z[i,j] = f(c(x[i],y[j],-q))

    persp(x, y, z, phi = 20, theta = 40)
    contour(x, y, z)
    image(x, y, z)
}
```








```
# Comparing optimization methods for all possible values of (x, y, z) for all integers between -10 and

x_range = seq(-10,10,2)
y_range = seq(-10,10,2)
z_range = seq(-10,10,2)

# Initialize empty results for later
optim_results = c()
nlm_results = c()

for (xi in x_range) {
  for (yi in y_range) {
    for (zi in z_range) {
      out.optim <- optim(c(xi,yi,zi), fn = f)
```

```
      out.nlm <- nlm(p=c(xi,yi,zi), f = f)
      # We need to round agressively to get unique results
      optim_results <- rbind(optim_results, round(c(out.optim$par, out.optim$value),2))
      nlm_results <- rbind(nlm_results, round(c(out.nlm$estimate, out.nlm$minimum),digits=2))
    }
  }
}

# Just give the unique results

print(unique(optim_results))
```

```
##        [,1]  [,2]  [,3]    [,4]
##  [1,]  1.00  0.00  0.00    0.00
##  [2,]  1.00 -0.01 -0.01    0.00
##  [3,]  1.00  0.01  0.01    0.00
##  [4,]  1.00  0.00  0.01    0.00
##  [5,] -1.00 -0.03 -4.90   24.27
##  [6,]  1.00 -0.01 -0.02    0.00
##  [7,]  1.00  0.00 -0.01    0.00
##  [8,]  0.96  0.42  0.64    0.64
##  [9,] -6.33  0.00  2.00 3748.44
## [10,]  1.00  0.01  0.02    0.00
## [11,]  0.94 -0.44 -0.71    0.72
## [12,] -0.50  0.00 -7.19  557.96
## [13,]  1.00 -0.02 -0.03    0.00
## [14,] -1.00  0.00 -4.95   24.74
## [15,]  1.01 -0.05 -0.08    0.01
## [16,] -1.51  0.00 -5.38   69.05
## [17,] -4.80  0.00  0.93 3098.65
## [18,] -0.48  0.88  3.28   10.76
## [19,]  1.00  0.05  0.08    0.01
## [20,]  1.00  0.02  0.03    0.00
## [21,] -1.00  0.00  4.95   24.72
## [22,]  1.01  0.01  0.01    0.01
## [23,]  0.00  0.00  6.98  539.05
## [24,]  0.96  0.28  0.46    0.21
## [25,] -0.64  0.00 -5.43   60.81
## [26,]  0.97  0.26  0.41    0.17
## [27,] -2.38  0.00  3.47  437.03
## [28,]  1.00 -0.02 -0.04    0.00
## [29,] -1.23  0.00 -5.18   35.51
## [30,]  1.00 -0.06 -0.09    0.01
## [31,]  1.00 -0.07 -0.11    0.01
## [32,]  1.00  0.03  0.05    0.00
## [33,] -1.18  0.00 -5.11   30.65
## [34,]  0.99  0.04  0.06    0.01
## [35,]  1.00  0.02  0.04    0.00
## [36,]  1.00  0.06  0.09    0.01
## [37,]  1.00  0.04  0.06    0.00
## [38,]  1.00  0.06  0.10    0.01
## [39,]  1.00  0.03  0.06    0.00
## [40,] -1.00  0.00 -4.95   24.73
## [41,]  1.00 -0.03 -0.05    0.00
```

```
## [42,]  1.00 -0.04 -0.07    0.01
## [43,]  1.00 -0.06 -0.10    0.01
## [44,]  0.98 -0.19 -0.30    0.09
## [45,] -0.60  0.00  6.06  165.01
## [46,]  1.00  0.03  0.04    0.00
## [47,]  0.00  0.00  7.00  549.76

print(unique(nlm_results))

##        [,1]  [,2]   [,3]    [,4]
##  [1,]  1.00  0.00   0.00    0.00
##  [2,] -4.00  0.00   6.00 1036.00
##  [3,] -0.75  0.00   5.05   32.09
##  [4,] -1.37  0.00  -6.25  210.10
##  [5,] -2.00  0.00   6.00  236.00
##  [6,] -2.00  0.00   8.00 1064.00
##  [7,] -2.00  0.00  10.00 2700.00
##  [8,] -1.37  0.00   6.25  210.10
##  [9,] -1.08  0.00  -5.03   25.98
## [10,]  0.00  0.00 -10.00 5824.02
## [11,]  0.00 -0.01  -8.00 3187.65
## [12,]  0.00  0.00  -6.00 1359.99
## [13,]  0.00  0.00  -4.00  340.35
## [14,]  0.00  0.00  -2.00  128.68
## [15,]  0.00  0.00   0.00  100.00
## [16,]  0.00  0.01   2.00  127.34
## [17,]  0.00  0.01   4.00  339.54
## [18,]  0.00  0.00   6.00 1359.95
## [19,]  0.00  0.01   8.00 3187.61
## [20,]  0.00  0.01  10.00 5823.28
## [21,] -0.39  0.00   5.47   89.26
## [22,] -1.00  0.00  -4.95   24.75
## [23,] -0.63  0.00   5.42   60.61
## [24,] -0.55  0.00   5.49   74.56
## [25,] -0.55  0.00  -5.49   74.56
```