Assignment 8

Tushar Ponkshe, tvp2110

December 6, 2018

---

(1)

```
n <- 100
p <- 10
s <- 3
set.seed(0)
x <- matrix(rnorm(n*p), n, p)
b <- c(-0.7, 0.7, 1, rep(0, p-s))
y <- x %*% b + rt(n, df=2)

cors = rep(NA, 10)
for (i in 1:10) {
  cors[i] = cor(x[,i],y)
}
order(abs(cors), decreasing = TRUE)
```

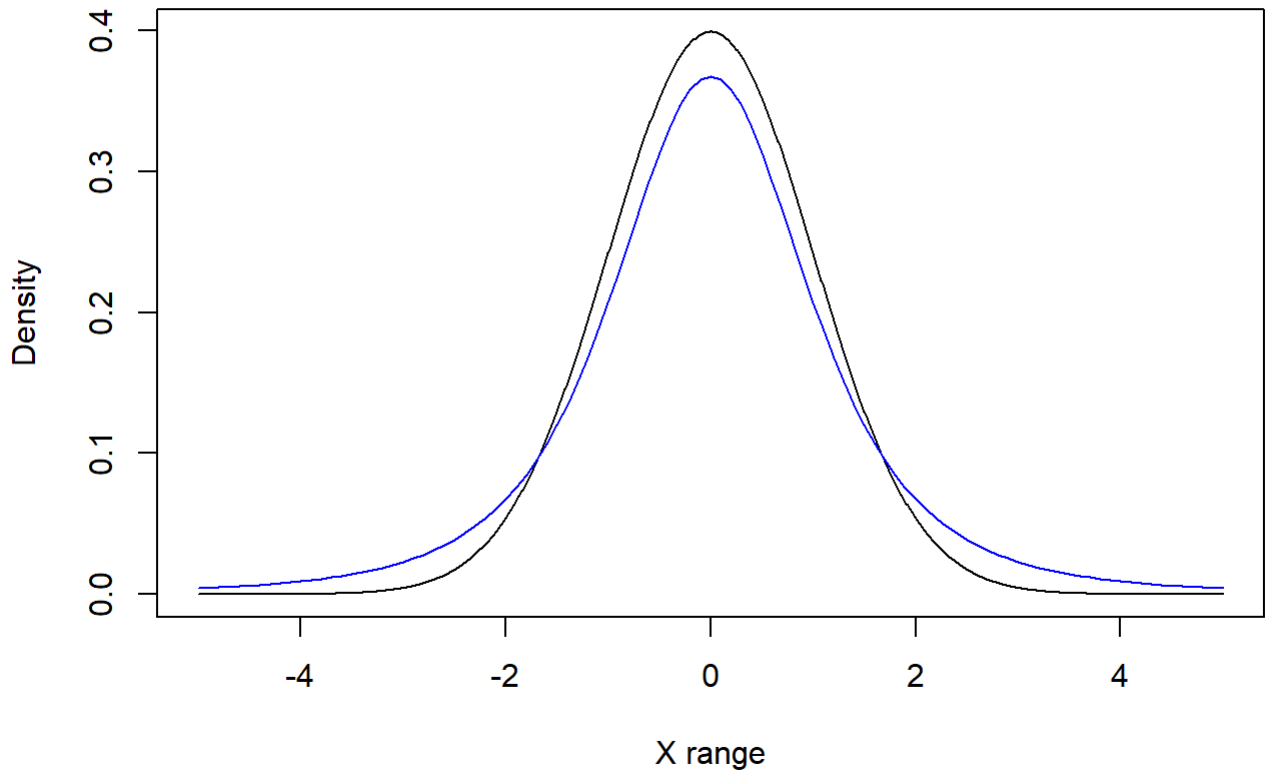
```
##  [1]  1  4  3  6  2  7  8 10  5  9
```

No we would not be able to pick out each of the 3 relevant variables based on correlations alone.

---

(2)

```
xvals <- seq(-5, 5, by = 1/100)
plot(xvals, dnorm(xvals), type = "l", xlab = "X range", ylab = "Density")
curve(dt(x, df = 3), add = TRUE, col = "blue")
```



We can see from the plot that t-distribution has thicker tails (blue)

(3)

```
psi <- function(r, c = 1) {
  return(ifelse(r^2 > c^2, 2*c*abs(r) - c^2, r^2))
}

huber.loss = function(beta) {
  sum(psi(y - x %*% beta))
}
```

---

(4)

```
library(numDeriv)

grad.descent <- function(f, x0, max.iter = 200, step.size = 0.05, stopping.deriv = 0.01, ...) {

  n    <- length(x0)
  xmat <- matrix(0, nrow = n, ncol = max.iter)
  xmat[,1] <- x0

  for (k in 2:max.iter) {
    # Calculate the gradient
    grad.cur <- grad(f, xmat[ ,k-1], ...)

    # Should we stop?
    if (all(abs(grad.cur) < stopping.deriv)) {
      k <- k-1; break
    }

    # Move in the opposite direction of the grad
    xmat[ ,k] <- xmat[ ,k-1] - step.size * grad.cur
  }

  xmat <- xmat[ ,1:k] # Trim
  return(list(x = xmat[,k], xmat = xmat, k = k))
}

gd = grad.descent(huber.loss, x0 = rep(0, p), max.iter = 200, step.size = 0.001, stopping.deriv
 = 0.1)
gd$k
```

```
## [1] 127
```

```
gd$x
```

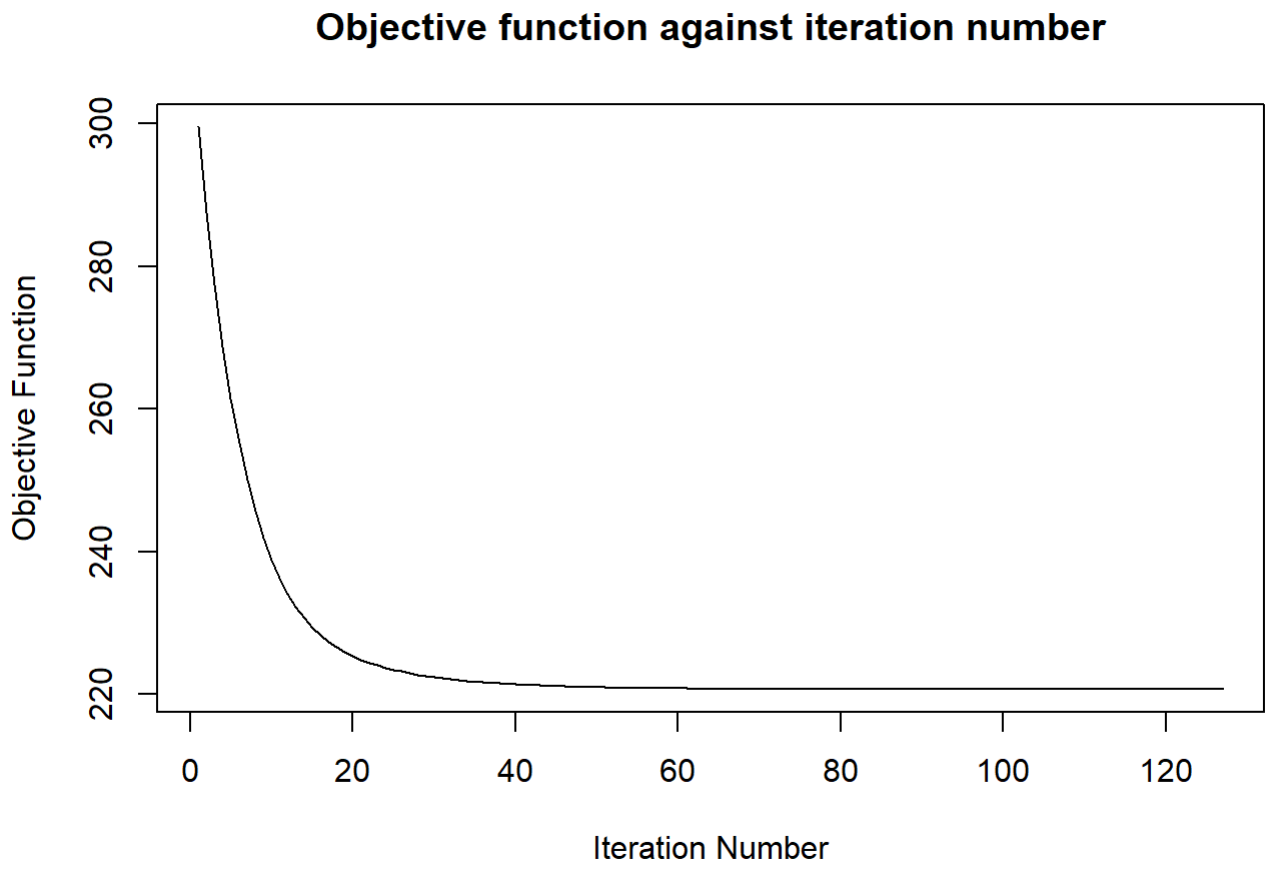
```
##  [1] -0.87346579  0.61828938  0.87989797 -0.04910821  0.07277491
##  [6]  0.10229815 -0.12513246 -0.14559243 -0.11903666 -0.02250130
```

It took 127 iterations to converge. Final coefficient estimates are shown in the output

---

(5)

```
iterations = gd$k
xval = c(1:iterations)
x.mat = gd$xmat
obj <- apply(x.mat[, 1:iterations], 2, huber.loss)
plot(xval, obj, type = "l", xlab = "Iteration Number", ylab = "Objective Function", main = "Obje
ctive function against iteration number")
```



Till 40 or so iterations, the objective function decreases sharply. However in later iterations the progress slows down and the obj function doesn't decrease as quickly - alsmot seems to remain constant after 40 or so iterations.

---

(6)

```
gd.new = grad.descent(huber.loss, x0 = rep(0, p), max.iter = 200, step.size = 0.1, stopping.deri
v = 0.1)

iterations.new = gd.new$k

gd.new$x
```

```
##  [1]  1.0740298 -0.7971898  2.8860325 -1.8822687  2.1897562  0.8721260
##  [7] -1.0055026 -1.5049278  0.9241456  4.7508245
```

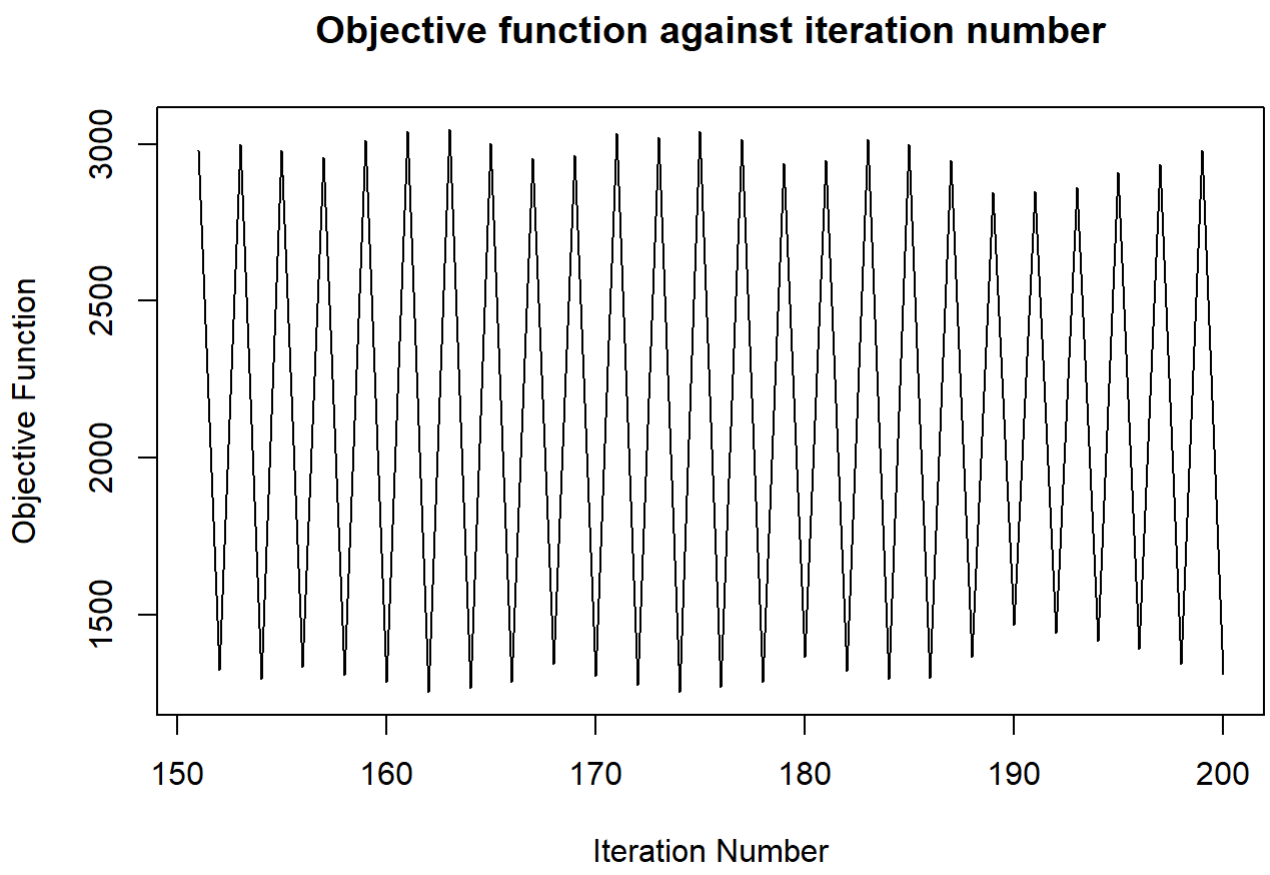
```
iterations.new
```

```
## [1] 200
```

```
xval.new = c((iterations.new-49):iterations.new)

obj <- apply(gd.new$xmat[, 1:iterations.new], 2, huber.loss)

plot(xval.new, obj[(iterations.new-49):iterations.new], type = "l", xlab = "Iteration Number", y
lab = "Objective Function", main = "Objective function against iteration number")
```



It is clear from the plot that the gradient descent algorithm has failed to converge with the step size of 0.1 in the last 50 iterations.

No, the criterion is not decreasing with each step and hasn't converged at the end. It seems to oscilate between values and doesn't settle on a single criterion value.

The coeffcieint estimates also oscillate between negative and positive values. This is confirmed by the xmat values in gd.

---

(7)

```
gd$x
```

```
##  [1] -0.87346579  0.61828938  0.87989797 -0.04910821  0.07277491
##  [6]  0.10229815 -0.12513246 -0.14559243 -0.11903666 -0.02250130
```

```
b
```

```
##  [1] -0.7  0.7  1.0  0.0  0.0  0.0  0.0  0.0  0.0  0.0
```

```
sparse.grad.descent <- function(f, x0, max.iter = 200, step.size = 0.05, stopping.deriv = 0.01,
...) {

  n    <- length(x0)
  xmat <- matrix(0, nrow = n, ncol = max.iter)
  xmat[,1] <- x0

  for (k in 2:max.iter) {
    # Calculate the gradient
    grad.cur <- grad(f, xmat[ ,k-1], ...)

    # Should we stop?
    if (all(abs(grad.cur) < stopping.deriv)) {
      k <- k-1; break
    }

    # Move in the opposite direction of the grad
    thr <- xmat[ ,k-1] - step.size * grad.cur
    thr[abs(thr) < 0.05] <- 0
    xmat[ ,k] <- thr
  }

  xmat <- xmat[ ,1:k] # Trim
  return(list(x = xmat[,k], xmat = xmat, k = k))
}

gd.sparse = sparse.grad.descent(huber.loss, x0 = rep(0, p), max.iter = 200, step.size = 0.001, s
topping.deriv = 0.1)

gd.sparse$k
```

```
## [1] 200
```

```
gd.sparse$x
```

```
##  [1] -0.8944804  0.6332991  0.8823860  0.0000000  0.0000000  0.0000000
##  [7]  0.0000000  0.0000000  0.0000000  0.0000000
```

---

(8)

```
fit = lm(y~-1+x)
fit$coefficients
```

```
##            x1            x2            x3            x4            x5
## -0.9477210986  0.4864220270  0.5875664655 -0.7416200316  0.0008874065
##            x6            x7            x8            x9           x10
##  0.3149846567 -0.3994729398 -0.2712937636 -0.1445449407  0.0788007924
```

```
gd$x
```

```
##  [1] -0.87346579  0.61828938  0.87989797 -0.04910821  0.07277491
##  [6]  0.10229815 -0.12513246 -0.14559243 -0.11903666 -0.02250130
```

```
gd.sparse$x
```

```
##  [1] -0.8944804  0.6332991  0.8823860  0.0000000  0.0000000  0.0000000
##  [7]  0.0000000  0.0000000  0.0000000  0.0000000
```

The coefficients obtained by fitting a linear model match closest to those obtained by the gradient descent in (4). The results are not similar to those obtained by sparse gradient descent in (7) since a linear model doesn't output the sparse result.

```
mse.lm = mean((b-fit$coefficients)^2)
mse.lm
```

```
## [1] 0.1186581
```

```
mse.gd = mean((b-gd$x)^2)
mse.gd
```

```
## [1] 0.01208955
```

```
mse.sparse.gd = mean((b-gd.sparse$x)^2)
mse.sparse.gd
```

```
## [1] 0.005610471
```

Thw sparse gd gives the best results in terms of MSE (lowest MSE)

---

(9)

```
set.seed(10)
y = x%*% b + rt(n, df=2)

gd <- grad.descent(huber.loss, x0 = rep(0,p), max.iter=200, step.size=0.001, stopping.deriv=0.1)
gd$x
```

```
##  [1] -0.46329748  0.92390614  0.92287242 -0.06526259  0.24633002
##  [6] -0.04406371  0.01858892 -0.18921630  0.19479185 -0.18395820
```

```
gd.sparse = sparse.grad.descent(huber.loss, x0 = rep(0, p), max.iter = 200, step.size = 0.001, s
topping.deriv = 0.1)
gd.sparse$x
```

```
##  [1] 0.0000000 0.7850744 0.9398727 0.0000000 0.0000000 0.0000000 0.0000000
##  [8] 0.0000000 0.0000000 0.0000000
```

```
mse.gd = mean((b-gd$x)^2)
mse.gd
```

```
## [1] 0.02869228
```

```
mse.sparse.gd = mean((b-gd.sparse$x)^2)
mse.sparse.gd
```

```
## [1] 0.0500853
```

The estimates found by regular gd in this case are close to the actual values. For sparse gd, we expected values below the threshold to be converted to 0 which is the case here. However we also observe that the value for the 1st estimate is 0.

In this case we see that the mse obtained by running the regular gd is lower than mse of sparse.gd.

This suggests high variablity in the estimates of sparse.gd.

---

(10)

```
all.mse.gd = rep(NA, 10)
all.mse.sparse = rep(NA, 10)

mse.calc = function(est) {
  return(mean((b-est)^2))
}

for (i in 1:10) {
  y = x%*% b + rt(n, df=2)

  gd <- grad.descent(huber.loss, x0 = rep(0,p), max.iter=200, step.size=0.001, stopping.deriv=0.
1)

  gd.sparse = sparse.grad.descent(huber.loss, x0 = rep(0, p), max.iter = 200, step.size = 0.001,
 stopping.deriv = 0.1)

  all.mse.gd[i] = mse.calc(gd$x)
  all.mse.sparse[i] = mse.calc(gd.sparse$x)
}

mean(all.mse.gd)
```

```
## [1] 0.02495459
```

```
min(all.mse.gd)
```

```
## [1] 0.01430856
```

```
mean(all.mse.sparse)
```

```
## [1] 0.02650818
```

```
min(all.mse.sparse)
```

```
## [1] 0.0006265157
```

The average mse of the regular gd is lower than that of the sparse gd. However, in 10 iterations, the minimum of mse for sparse gd is much smaller than the minimum of mse for regular gd.

This confirms the fact that sparse gd has higher variability.