Econ 21410 - Problem Set I

Review of R and Regression Analysis[*]

April 7, 2014

This homework should be done in LaTeX The homework will be graded on correctness, but will also heavily weight clarity and professionalism. Being able to produce clean, clear, well documented write-ups and code is an important skill which will be rewarded. Its better to not do some of the harder parts than to turn in an incomprehensible document. Your R script as well as a log-file should be submitted. Alternatively, use knitr to print your code inline in your latex document.

Make sure to write code which is clear and flexible. Read the whole problem before you begin coding. Some parameters will change and the code should be written in a way to make this easy to implement. We will re-use code in this course. Flexibility and documentation now will save you headaches later in the quarter.

SUBMISSION: The homework must be emailed to Oliver and myself by 2p.m. Monday, April the 7th. The email must include a pdf with the filename lastname_pset1.pdf and R code called lastname_pset1_code.R where "lastname" should be replaced with your last name. The subject of your email should be [ECON 21410: pset1 submission]

1 A Quick Review of R

1. Create a vector $y = \begin{bmatrix} 100 \\ 200 \\ 300 \\ 400 \\ 500 \end{bmatrix}$
2. Set a random seed equal to 1234
3. Create a matrix x which is 5×5 and contains random draws from a normal with mean 100 and variance 10. Display this output in your code (preferably inline with knitr). None of these should require more than a single line of R. These exercises must be calculated in R, not done by hand.
4. Display $x'x$
5. Display $(x'x)^{-1}$

[*]Please email johneric@uchicago.edu and obrowne@uchicago.edu if you have questions.

6. Calculate the sum of the entries in y
7. Calculate the row sums of the entries of x
8. Return the maximum value in y
9. Return which entry is the maximum value of y
10. Return which entry is the maximum value of x
11. Return $3 * x$
12. Return z a new vector which contains the value of y sorted from low to high
13. Replace the third row of x with 0s and display it

The Solution to 1:

```r
y <- matrix(c(200, 100, 500, 400, 300), 5, 1)
y
```

```
##      [,1]
## [1,]  200
## [2,]  100
## [3,]  500
## [4,]  400
## [5,]  300
```

```r
set.seed(1234)
x <- matrix(rnorm(25, mean = 100, sd = sqrt(10)), 5,
    5)
x
```

```
##        [,1]   [,2]   [,3]   [,4]   [,5]
## [1,]  96.18 101.60  98.49  99.65 100.42
## [2,] 100.88  98.18  96.84  98.38  98.45
## [3,] 103.43  98.27  97.55  97.12  98.61
## [4,]  92.58  98.22 100.20  97.35 101.45
## [5,] 101.36  97.19 103.03 107.64  97.81
```

```r
xx <- t(x) %*% x
xx
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]
## [1,] 48970 48784 49052 49478 49095
## [2,] 48784 48711 48956 49351 49029
## [3,] 49052 48956 49251 49662 49287
## [4,] 49478 49351 49662 50106 49674
## [5,] 49095 49029 49287 49674 49359
```

```
xx_inv <- solve(xx)
xx_inv
```

```
##            [,1]     [,2]      [,3]      [,4]
## [1,]  0.012521 -0.01289  0.004975 -0.009165
## [2,] -0.012891  0.50915  0.631856 -0.364406
## [3,]  0.004975  0.63186  1.039633 -0.599355
## [4,] -0.009165 -0.36441 -0.599355  0.358087
## [5,]  0.004607 -0.75712 -1.067505  0.609187
##            [,5]
## [1,]  0.004607
## [2,] -0.757123
## [3,] -1.067505
## [4,]  0.609187
## [5,]  1.200359
```

```
sum(y)
```

```
## [1] 1500
```

```
apply(x, 1, sum)
```

```
## [1] 496.3 492.7 495.0 489.8 507.0
```

```
max(y)
```

```
## [1] 500
```

```
which(x == max(x), x, arr.ind = TRUE)
```

```
## Warning:  the condition has length > 1 and only the first element will be used
```

```
##      row col
## [1,]   5   4
```

```
3 * x
```

```
##       [,1]  [,2]  [,3]  [,4]  [,5]
## [1,] 288.5 304.8 295.5 299.0 301.3
## [2,] 302.6 294.5 290.5 295.2 295.3
## [3,] 310.3 294.8 292.6 291.4 295.8
## [4,] 277.7 294.6 300.6 292.1 304.4
## [5,] 304.1 291.6 309.1 322.9 293.4
```

```
z <- sort(y)
z

## [1] 100 200 300 400 500

x[3, ] <- 0
x

##         [,1]   [,2]   [,3]   [,4]   [,5]
## [1,]   96.18 101.60  98.49  99.65 100.42
## [2,] 100.88  98.18  96.84  98.38  98.45
## [3,]   0.00   0.00   0.00   0.00   0.00
## [4,]  92.58  98.22 100.20  97.35 101.45
## [5,] 101.36  97.19 103.03 107.64  97.81
```

2 A Quick Review of LaTeX

1. Display the matrix and vector $x$ and $y$ above in LaTeX(no need to include the decimals)

$$y = \begin{bmatrix} 200 \\ 100 \\ 500 \\ 400 \\ 300 \end{bmatrix}$$

$$x = \begin{bmatrix} 96 & 101 & 98 & 99 & 100 \\ 100 & 98 & 96 & 98 & 98 \\ 0 & 0 & 0 & 0 & 0 \\ 92 & 98 & 100 & 97 & 101 \\ 101 & 97 & 103 & 107 & 97 \end{bmatrix}$$

2. Print the symbols α, θ_j, $\lambda_{t,t+1}$, $\gamma^{s,t}$ inline with text.

3. Write on its own centered line:
$$\sum_{t=1}^{T} \frac{a_t}{b_t} \xrightarrow{p} \infty$$

4. Write $a \neq b$ and $c \geq d$

The Solution to 2:

```
\section{A Quick Review of \LaTeX }
\begin{enumerate}
    \item Display the matrix and vector $x$ and $y$ above in \LaTeX
    (no need to include the decimals)
    \begin{eqnarray*}
        y & = & \left[\begin{array}{c}
        200\\
        100\\
        500\\
        400\\
        300
        \end{array}\right]\\
        x & = & \left[\begin{array}{ccccc}
        96  & 101 & 98  & 99  & 100\\
        100 & 98  & 96  & 98  & 98\\
        0   & 0   & 0   & 0   & 0\\
        92  & 98  & 100 & 97  & 101\\
        101 & 97  & 103 & 107 & 97
        \end{array}\right]
    \end{eqnarray*}

    \item Print the symbols $\alpha$, $\theta_j$, $\lambda_{t,t+1}$,
    $\gamma^{s,t}$ inline with text.

    \item Write on its own centered line:
    $$\sum_{t = 1}^T \frac{a_t}{b_t} \overset{p}{\rightarrow} \infty  $$

    \item Write $a \neq b$ and $c \geq d$

\end{enumerate}
```

3 Getting started with Github

1. You should have already made a github.com account and shared your user name with Oliver and myself, if not, do so as soon as you read this.

2. Go to "CompEcon" at github.com/CompEcon. You should be able to see 3 repositories, including a repository with your name in it. Go into this repository and click on README.md. Modify the README.md file to include your name and email address.

4 Regression

Recall what you have learned about ordinary least squares and what we reviewed in class. In this problem you will estimate OLS coefficients and standard errors a number of ways. You will experiment with how sample size, optimizer options, and other parameters change the estimates. For this assignment, we will provide you with code to generate *almost* all of the data you will need to answer this problem.

Here is the code needed to generate the data:

```r
# ===================== TITLE: computational
# economics: assignment 1

# AUTHOR: John Eric Humphries

# abstract: problem set on regression for econ
# 21410

# Date: 2014-03-14 ====================

# ======================== Section 0: setup
# ======================== setwd('')
rm(list = ls())  # Clear the workspace
set.seed(21410)  # Set random seed


# ===================== Section 1: Generating Data
# ======================
n <- 200  # observations
x1 <- rbinom(n, 10, 0.5)
x2 <- rnorm(n, 20, 10)
X <- cbind(1, x1, x2)
eps <- rnorm(n, 0, sqrt(4))
beta <- matrix(cbind(2, 3, 1.4), 3, 1)
Y <- X %*% beta + eps
```

We will consider the model:

$$Y = X\beta + \epsilon$$

Algebraic Approach

1. What is the algebraic formula for the OLS estimate of $\hat{\beta}$ (in matrix notation)?

$$\hat{\beta} = (X'X)^{-1}X'y$$

2. What are the dimensions of Y_i, X_i, β, and ϵ

$$\underbrace{Y_i}_{1\times 1} = \underbrace{X_i}_{1\times k}\underbrace{\beta}_{k\times 1} + \underbrace{\epsilon_i}_{1\times 1}\ \forall i \in 1\ldots n$$

and

$$\underbrace{Y}_{n\times 1} = \underbrace{X}_{n\times k}\underbrace{\beta}_{k\times 1} + \underbrace{\epsilon}_{n\times 1}$$

In this example $k = 3$ and $n = 200$

3. What are the formulas for the standard errors for the three elements in β?
Let $\hat{\epsilon}_i = Y_i - X_i\hat{\beta}$ and then $\widehat{\sigma^2} = \frac{1}{n}\sum_{i=1}^{n}\hat{\epsilon_i}^n$

$$\widehat{Var}(\hat{\beta}|X) = \widehat{\sigma^2}(X'X)^{-1}$$

Least Squares Approach

1. Write out the minimization problem which OLS solves.

$$\min_{\beta}(Y - X\beta)'(y - X\beta)$$

2. Take the first order condition and solve for $\hat{\beta}$.[1] Show that you get the algebraic formula from above.

$$\begin{aligned} 0 &= -2X'(Y - X\hat{\beta}) \\ \Rightarrow \hat{\beta} &= (X'X)^{-1}X'Y \end{aligned}$$

Maximum Likelihood Approach

1. Assuming $\epsilon_i \sim N(0, \sigma^2)$, write the (log) likelihood contribution for a single individual

$$\begin{aligned} \mathcal{L}(\beta, \sigma^2 | Y_i, X_I) &= f(Y_i|\beta) \\ &= \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(Y_i - X_i\beta)^2} \\ \log \mathcal{L}(\beta, \sigma^2 | Y_i, X_I) &= -\frac{1}{2}\log(\sigma^2) - \frac{1}{2}\log(2\pi) - \frac{1}{2\sigma^2}(Y_i - X_i\beta)^2 \end{aligned}$$

2. Write (log) likelihood for the whole sample.

$$\begin{aligned} \log \mathcal{L}(\beta, \sigma^2 | Y_i, X_I) &\propto -\frac{n}{2}\log(\sigma^2) - \sum_{i=1}^{n} \frac{1}{2\sigma^2}(Y_i - X_i\beta)^2 \\ &= -\frac{n}{2}\log(\sigma^2) - \frac{1}{2\sigma^2}(Y - X\beta)'(Y - X\beta) \end{aligned}$$

3. Write the optimization problem which characterizes the maximum likelihood. Highlight why $\hat{\beta}$ estimated this way will be equivalent to the least-squares estimate of $\hat{\beta}$.

$$\max_{\beta, \sigma^2} \log \mathcal{L}(\beta, \sigma^2 | Y_i, X_I)$$

$$\begin{aligned} [\text{FOC } \beta] : \quad 0 =& -2X'(Y - X\hat{\beta}) \\ \Rightarrow \quad \hat{\beta} =& (X'X)^{-1}X'Y \end{aligned}$$

[1] If taking derivatives of matrices is causing problems, there will be only a small deduction for doing this for the single-variate case..

4. How many variables are being optimized We are optimizing over 4 variables $\beta_0, \beta_1, \beta_2, \sigma^2$

5. Are there bounds on any of the variables being optimized? Why? $\sigma^2 > 0$ since we cannot have a negative variance (in practice this will lead to a division by zero)

6. What is the formula for the standard errors of $\hat{\beta}$? (you do not need to explicitly solve for the second derivative, just write the formula down).

$$\begin{aligned}
[\text{FOC}\sigma^2]: \quad 0 &= \frac{-n}{2\sigma^2} - \frac{-1}{2(\sigma^2)^2}(Y - X\beta)'(Y - X\beta) \\
\Rightarrow \quad \widehat{\sigma^2_{MLE}} &= \frac{1}{n}(Y - X\beta)'(Y - X\beta) \\
\Rightarrow \quad \text{VAR}(\hat{\beta}|X, Y) &= \widehat{\sigma^2_{MLE}}(X'X)^{-1}
\end{aligned}$$

Note that this is exactly equivalent to OLS

Implementing OLS

Using the derivations from the previous section we will now construct a function to estimate $\hat{\beta}$ various ways. The end goal will be to construct a function which takes three inputs X, Y, and "method = c(0,1,2,3)", where method will specify which method the code should use to estimate $\hat{\beta}$. If necessary, you may create 3 new functions rather than 1 general purpose function with little penalty.

- method=0 should call the 'lm()' function, which is R's default method for performing OLS.
- method=1 should estimate β using the algebraic approach.
- method=2 should estimate β using the least squares approach.
- method=3 should estimate β using MLE.

Methods 2 and 3 require a use of numeric optimization to solve the optimization problem. R can call world-class optimization packages, but for now we will use "optim()", the standard multivariate optimization command.

Output: Produce a 4 x 3 matrix where each column contains the estimates of β from one of the four methods.

```r
# OLS Functions ============================

SumOfSquares <- function(b, x = X, y = Y) {
    # Sum of squares function for numerical optimizer
    sum((y - x %*% b)^2)
}

Llik <- function(pars, x = X, y = Y) {
    # Log Likelihood function for normal OLS with
    # numerical optimizer
    b <- pars[1:3]
    t2 <- pars[4]
    if (t2 <= 1e-06)
        value <- 1e+21 else value <- -1 * (-n/2 * log(2 * pi) - n/2 *
        log(t2) - 1/(2 * t2) * (t(y - x %*% b) %*%
        (y - x %*% b)))
    value
}

OLS.gradient <- function(beta, X, Y) {
    # Returns the analytic gradient of OLS
    return(-2 * t(X) %*% (Y - X %*% beta))
}

OLS <- function(x = X, y = Y, method = 1, optim.method = "Nelder-Mead",
    gradient = NULL, hessian = FALSE) {
    # Calculates OLS using one of four methods: Method
    # == 0 uses the standard lm function Method == 1
    # Calculates OLS algebraically Method == 2 Uses an
    # optimizer to minimize the sum of squares Method
    # == 1 Uses an optimizer to maximize a likelihood
    if (method == 0) {
        result <- lm(y ~ x - 1)
        beta_hat0 <- result$coefficients
        return(beta_hat0)

    } else if (method == 1) {
        beta_hat1 <- solve(t(x) %*% x) %*% (t(x) %*%
            y)
        return(t(beta_hat1))

    } else if (method == 2) {
        beta_hat2 <- optim(c(0, 0, 0), SumOfSquares,
            method = optim.method, x = x, y = y, hessian = hessian)

        if (hessian == TRUE) {
            return(list(beta_hat2$par, beta_hat2$hessian))
        } else {
            return(beta_hat2$par)
        }

    } else if (method == 3) {
        beta_hat3 <- optim(c(1, 1, 1, 1000), Llik,
            method = optim.method, x = x, y = y, gr = gradient)$par
        return(beta_hat3[1:3])
    } else {
        return("Error! Method not found")
    }
}

# ============================
# OLS Estimates ============================

method <- c(0, 1, 2, 3)

# Generate OLS estimates for each method
sapply(method, function(m, x, y) OLS(x, y, m), x = X,
    y = Y, simplify = "array")


##      [,1]  [,2]  [,3]  [,4]
## x   1.158 1.158 1.145 1.159
## xx1 3.129 3.129 3.129 3.129
## xx2 1.411 1.411 1.412 1.411


beta_hat <- sapply(method, function(m, x, y) OLS(x,
    y, m), x = X, y = Y, simplify = "array")
# beta_hat



# ============================
```

Optimizers

The optim() command has an option which takes a number of optimizers. Re-estimate the model using the "BFGS", "CG", and "SANN" methods. The BFGS method uses gradient decent. As part of this method it numerically approximates the gradient and Hessian. You can improve the numerical accuracy of the optimization by providing the analytic gradient.

In three different plots, plot the three elements of $\hat{\beta}$ for sample sizes $\{8, 20, 50, 100, 1000, 10000\}$ (estimates on Y axis, sample size on X axis, one parameter per plot.) Plot a line for (a) the algebraic approach (b) the least squares approach (c) the least squares approach using "BFGS" (d) least squares using "CG" (e) least squares using "SANN" (f) least squares using "BFGS" and providing the analytic gradient.[2] Briefly discuss the results (no more than 5 sentences).

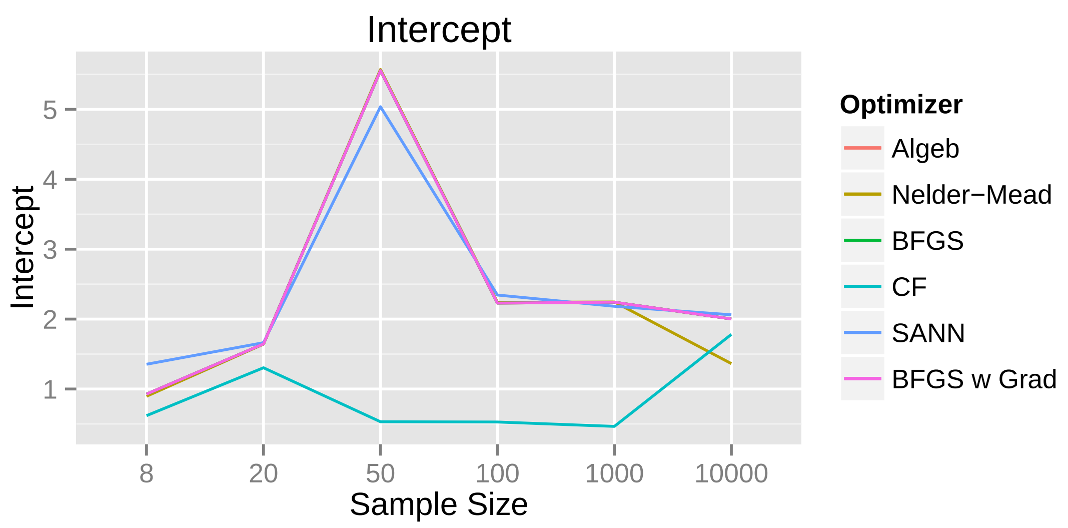
```
# Generate Data ============================

GenerateData <- function(n) {
    x1 <- rbinom(n, 10, 0.5)
    x2 <- rnorm(n, 20, 10)
    X <- cbind(1, x1, x2)
    theta2 <- 5
    eps <- rnorm(n, 0, sqrt(4))
    beta <- matrix(cbind(2, 3, 1.4), 3, 1)
    Y <- X %*% beta + eps
    return(list(X, Y))
}
```

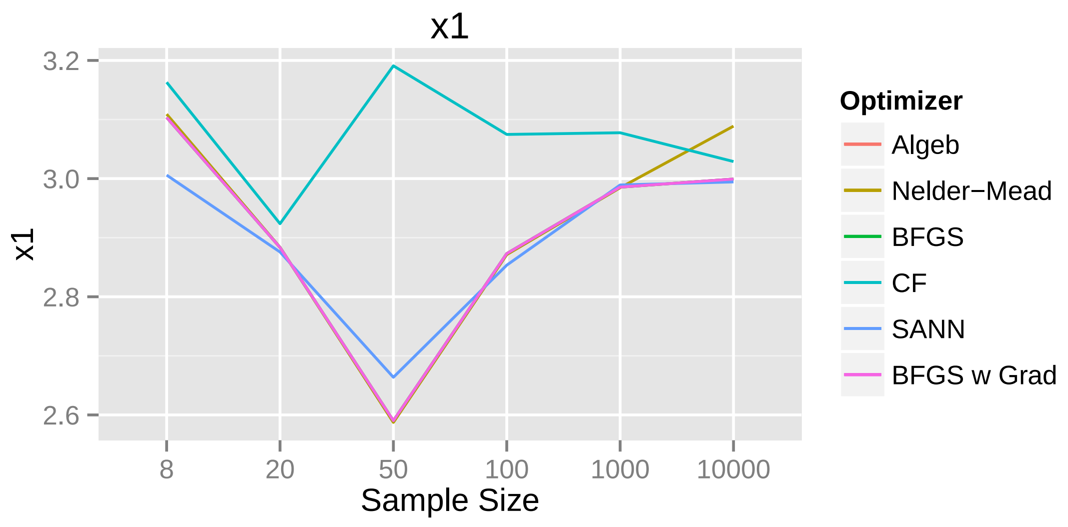
[2]Part f is quite a bit harder than parts a - e

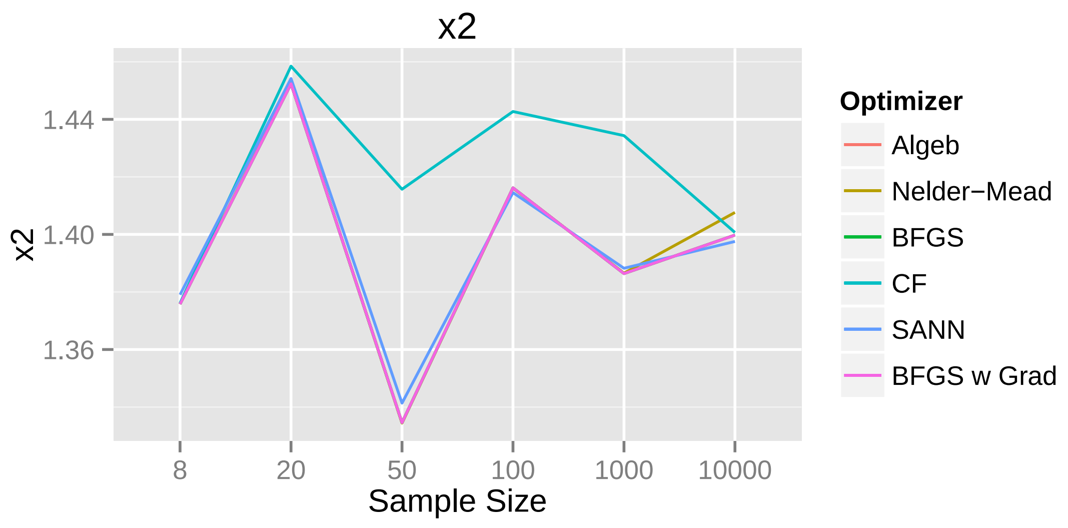
```r
sample.sizes <- c(8, 20, 50, 100, 1000, 10000)
algorithms <- c("BFGS", "CG", "SANN")
data.list <- sapply(sample.sizes, function(n) GenerateData(n))
X.list <- data.list[1, ]
Y.list <- data.list[2, ]

# ===============================
```

```r
# Run Regressions ===============================

# This code is poorly written a less lazy
# programmer would have used a loop.

beta.ot <- array(0, dim = c(3, 6, 6))
# Using algebraic approach
beta.ot[, , 1] <- sapply(1:6, function(i) OLS(X.list[[i]],
    Y.list[[i]], method = 1))
# Using Nelder-Mead
beta.ot[, , 2] <- sapply(1:6, function(i) OLS(X.list[[i]],
    Y.list[[i]], method = 2, optim.method = "Nelder-Mead"))
# Using BFGS
beta.ot[, , 3] <- sapply(1:6, function(i) OLS(X.list[[i]],
    Y.list[[i]], method = 2, optim.method = "BFGS"))
# Using CG
beta.ot[, , 4] <- sapply(1:6, function(i) OLS(X.list[[i]],
    Y.list[[i]], method = 2, optim.method = "CG"))
# Using SANN
beta.ot[, , 5] <- sapply(1:6, function(i) OLS(X.list[[i]],
    Y.list[[i]], method = 2, optim.method = "SANN"))
# Using BFGS w analytic gradient
beta.ot[, , 6] <- sapply(1:6, function(i) OLS(X.list[[i]],
    Y.list[[i]], method = 2, optim.method = "BFGS",
    gradient = OLS.gradient))

# ===============================
```

```r
# Plot Regressions Coeffficents
# ===============================

names <- c("Algeb", "Nelder-Mead", "BFGS", "CF", "SANN",
    "BFGS w Grad")

## Draw up the plots for each of the coefficients
graph1 <- ggplot(data = data.frame(beta.ot[1, , ]),
    aes(x = (1:6))) + geom_line(aes(y = X1, color = "X1")) +
    geom_line(aes(y = X2, color = "X2")) + geom_line(aes(y = X3,
    color = "X3")) + geom_line(aes(y = X4, color = "X4")) +
    geom_line(aes(y = X5, color = "X5")) + geom_line(aes(y = X6,
    color = "X6")) + xlab("Sample Size") + ylab("Intercept") +
    scale_x_discrete(labels = sample.sizes) + scale_color_discrete(name = "Optimizer",
    labels = names) + ggtitle("Intercept")
print(graph1)

graph2 <- ggplot(data = data.frame(beta.ot[2, , ]),
    aes(x = (1:6))) + geom_line(aes(y = X1, color = "X1")) +
    geom_line(aes(y = X2, color = "X2")) + geom_line(aes(y = X3,
    color = "X3")) + geom_line(aes(y = X4, color = "X4")) +
    geom_line(aes(y = X5, color = "X5")) + geom_line(aes(y = X6,
    color = "X6")) + xlab("Sample Size") + ylab("x1") +
    scale_x_discrete(labels = sample.sizes) + scale_color_discrete(name = "Optimizer",
    labels = names) + ggtitle("x1")
print(graph2)

graph3 <- ggplot(data = data.frame(beta.ot[3, , ]),
    aes(x = (1:6))) + geom_line(aes(y = X1, color = "X1")) +
    geom_line(aes(y = X2, color = "X2")) + geom_line(aes(y = X3,
    color = "X3")) + geom_line(aes(y = X4, color = "X4")) +
    geom_line(aes(y = X5, color = "X5")) + geom_line(aes(y = X6,
    color = "X6")) + xlab("Sample Size") + ylab("x2") +
    scale_x_discrete(labels = sample.sizes) + scale_color_discrete(name = "Optimizer",
    labels = names) + ggtitle("x2")
print(graph3)

# ================================
```





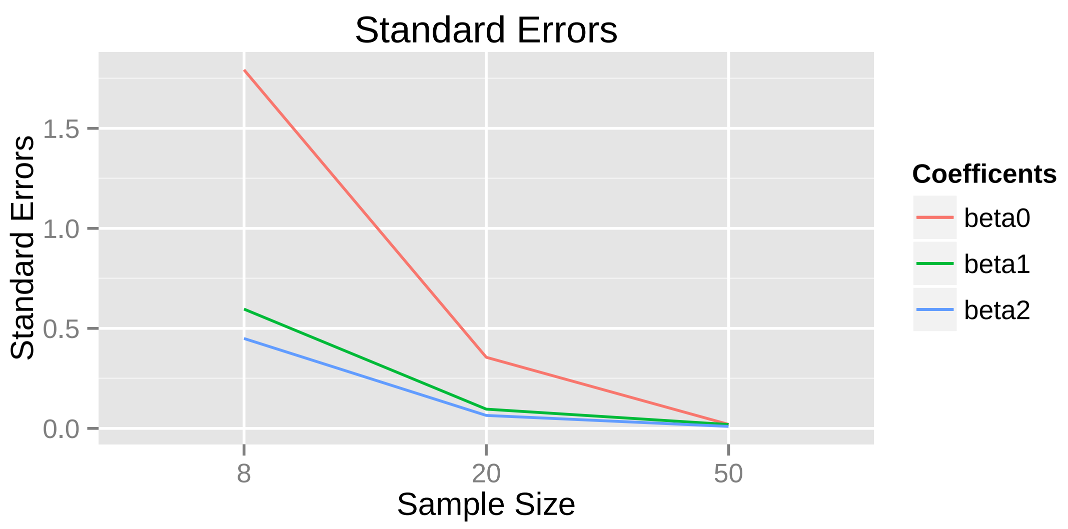


Standard Errors

Use the "hessian=T" option in optim() to return the hessian of the optimization. Use it to construct standard errors for the coefficients in the MLE approach using the "BFGS" optimizer. Make a plot for each parameter with the standard error on the Y axis and the sample sizes on the x-axis (only one line needed for each plot). Discuss briefly how standard error change with sample size (no more than 5 sentences).

```r
# Calculate Standard Errors
# ===============================

hessians <- sapply(1:6, function(i) OLS(X.list[[i]],
    Y.list[[i]], method = 2, optim.method = "BFGS",
    hessian = TRUE)[[2]])
se <- sapply(1:6, function(i) sqrt(diag(solve(matrix(hessians[,
    i], nrow = 3)))))
```

```r
graph4 <- ggplot(data = data.frame(se), aes(x = (1:3))) +
    geom_line(aes(y = X1, color = "X1")) + geom_line(aes(y = X2,
    color = "X2")) + geom_line(aes(y = X3, color = "X3")) +
    xlab("Sample Size") + ylab("Standard Errors") +
    scale_x_discrete(labels = sample.sizes) + scale_color_discrete(name = "Coefficents",
    labels = c("beta0", "beta1", "beta2")) + ggtitle("Standard Errors")
print(graph4)

# ===============================
```



5 Probit Estimation

Suppose we have the same model is before:

$$Y = X\beta + \epsilon$$

, but now Y is the gain in utility from a particular choice (say the choice to work or not). Inherently, we do not observe utility, but only observe if the individual chooses to work or not. We will use a “threshold model”, where an individual chooses to work if their unobserved utility is above a particular threshold:

$$D = \begin{cases} 1, & \text{if } Y \geq T, \\ 0, & \text{if } Y < T. \end{cases}$$

We will set $T = 0$. Using our linear-in-parameters form above, we can rewrite the decision process as:

$$D = \begin{cases} 1, & \text{if } \epsilon \geq (-X\beta), \\ 0, & \text{if } \epsilon < (-X\beta). \end{cases}$$

Assume $\epsilon \sim N(0, 1)$ and answer the following questions:

1. Why can we set $T = 0$ and $\sigma^2_\epsilon = 1$ without consequence?
   T and σ^2 are unidentified. There are an infinite number of values for these which are

consistent with this model. To see this we can set $T = 0$ without loss of generality because this problem is equivalent to the problem where we rename $\hat{\beta}_0 = \beta_0 - T$ and $\hat{T} = 0$. Similarly σ_ϵ^2 is unidentified because the probablity that a mean zero normal variable is greater than zero is independant of it's variance.

2. Write out the likelihood contribution for an individual

$$\log \mathcal{L}_i(\beta) = y_i \ln(\Phi(x_i'\beta)) + (1 - y_i)\ln(1 - \Phi(x_i'\beta))$$

3. Write out the likelihood contribution for the population

$$\log \mathcal{L}(\beta) = \sum_{i=1}^{n} \big(y_i \ln(\Phi(x_i'\beta)) + (1 - y_i)\ln(1 - \Phi(x_i'\beta))\big)$$

4. Modify the simulated data above to create a new dataset for a binary outcome. To do this, replace β_0 to be -45 rather than 2. Also adjust the variance of the error to be 1.

5. Create a function which calculates the likelihood of the data.

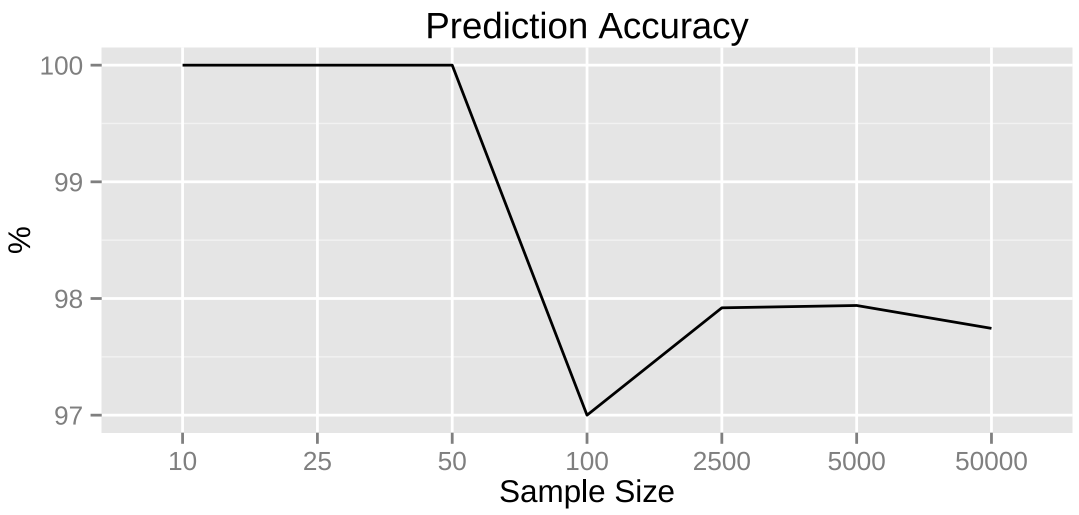
```r
# Generate Probit Data
# ================================

GenerateProbit <- function(n) {
    x1 <- rbinom(n, 10, 0.5)
    x2 <- rnorm(n, 20, 10)
    X <- cbind(1, x1, x2)
    theta2 <- 5
    eps <- rnorm(n, 0, 1)
    beta <- matrix(cbind(-45, 3, 1.4), 3, 1)
    Y <- X %*% beta + eps
    D <- (Y > 0)
    return(list(X, D))
}

# ================================
```

```r
# Probit Likelihood ================================

LogLikProbit <- function(beta, X = X, D = D) {

    L1 <- -log(pnorm(X %*% beta))
    L1[D != 1] <- 0
    L0 <- -log((1 - pnorm(X %*% beta)))
    L0[D != 0] <- 0
    return(sum(L1, L0))
}

ProbitMLE <- function(X = X, D = D, method = "Nelder-Mead") {

    beta.hat.prob <- optim(c(0, 0, 0), LogLikProbit,
        method = method, hessian = TRUE, X = X, D = D)$par
    return(beta.hat.prob[1:3])
}

n <- 10^5
probit.data <- GenerateProbit(n)
D <- probit.data[[2]]
X <- probit.data[[1]]
beta.hat <- ProbitMLE(X, D)

# ============================
```

6. Estimate $\hat{D}$ using $\hat{\beta}$ and the observed estimates. What proportion of the decisions do you get correct? Calculate this proportion with a sample size of 10, 25, 50, 100, and 5000.

```
# Probit MLE w sample sizes
# ============================

# Sample Sizes:
sample.sizes <- c(10, 25, 50, 100, 2500, 5000, 50000)
# Generate Probit Data for each sample size
probit.data <- sapply(sample.sizes, function(n) GenerateProbit(n))
# Estimate Probit Model for each sample size
beta_hat <- sapply(1:length(sample.sizes), function(n) ProbitMLE(probit.data[[1,
    n]], probit.data[[2, n]]))
beta_hat

##          [,1]     [,2]    [,3]    [,4]    [,5]
## [1,]  1.1333 -212.406 -447.06 -33.011 -43.629
## [2,] -0.3667   21.608   33.15   2.247   2.863
## [3,] -1.8000    4.531   13.53   1.010   1.371
##          [,6]    [,7]
## [1,] -48.473 -45.324
## [2,]   3.188   3.025
## [3,]   1.519   1.408

# Calculates predicted outcomes based on predicted
# beta for each sample size
D.hat <- sapply(1:length(sample.sizes), function(n) probit.data[[1,
    n]] %*% beta_hat[, n] > 0)
# Calculates the percentage of outcomes correctly
# predicted
sapply(1:length(sample.sizes), function(n) sum(D.hat[[n]] ==
    probit.data[[2, n]])/sample.sizes[n])

## [1] 1.0000 1.0000 1.0000 0.9700 0.9792 0.9794
## [7] 0.9774

pcnt <- sapply(1:length(sample.sizes), function(n) sum(D.hat[[n]] ==
    probit.data[[2, n]])/sample.sizes[n])
# pcnt

# =============================
```

```r
# Plot prediction accuracy
# ==============================

graph <- qplot(1:length(sample.sizes), pcnt * 100,
    geom = "line") + xlab("Sample Size") + ylab("%") +
    scale_x_discrete(labels = sample.sizes, breaks = (1:length(sample.sizes))) +
    ggtitle("Prediction Accuracy")
print(graph)
```



Potential Side Projects

- Read about the bootstrap algorithm for calculating standard errors. Add an option to calculate the standard errors of one of the OLS methods above using bootstrap. Write no more than a page summarizing how the bootstrap algorithm is implemented. Upload that description as a page on the wiki for the final half-point. (2.5 points)
- Complete your problem set in knitr. Create a sample knitr file that can be shared with your classmates and write a page on using knitr in the class wiki. (1.5 points)
- Rewrite at least a portion of regression code above in Julia[3] or C++ using the Rcpp package. Compare how long the new code takes to run in comparison with your R code. (3 points)
- Rewrite a portion of regression code in python. (1.5 points)
- Add a fifth method to your code to estimate the regression using Generalized Method of Moments. Include a short write-up of how you implemented GMM. (2.5 points)
- Make a meaningful contribution to the class wiki, start an issue and ask a valuable question, provide a detailed and useful answer to a classmate's question. Include 2-3 sentences in your homework stating your contributions. (1 point)

[3]Julia is a very promising new programming language for statistical computing. It is still very new, but I believe it may eventually be a quality replacement for R or python and some early investment now could be beneficial later. It is fast, has a simple syntax, is open source, and has a large community for such a young language.