Regression

R Exercise

This is a three-part exercise. In all three, the goal is to estimate the parameters of a model.

If you do not have much programming experience, the first part may be a bit difficult; if that is the case, don't get disappointed and invest your time on Parts 2 and 3.

In all three parts, you can use the following lines (and change them to your liking) to create your own data and then see how well your functions are performing compared to the outputs you get from R's built in functions (lm and glm).

```r
library(MASS)
N = 100
#creating independent variables
#first, the covariance matrix for our multi-variate normal
# this determines the correlation of our independent variables.
sig  = matrix(c(2,.5,.25,.5,1,0,.25,0,1) , nrow=3)
# now the variables:
M = mvrnorm(n = N, mu = rep(1,3), Sigma = sig )

y.cont = 1 + 2* M[,1] - 5 * M[,2]  + M[,3] + rnorm(N)
y.bin  = as.numeric ( y.cont > 0 )

# include the intercept in your independent variables
X = cbind (1, M )
y = y.cont
# or y = y.bin , depending on the exercise
```

Part 1: OLS using a recursive function (optional)

We want to write a function (call it reg) that takes two inputs (y: dependent variable, X: a matrix whose columns are the explanatory variables) and returns a list which contains its two outputs (b: estimated betas, and e: residuals). Remeber that the intercept can be seen as a coefficient that corresponds to an all-one column in the data.

$$y = b[1] * x[, 1] + b[2] * x[, 2] + ... + b[k] * x[, k] + e$$

The task is to write this function in a recursive way (an example is provided below). Every time you call the function, it should break the X matrix into two parts and call itself for each part in appropriate ways and then stitch the results back together in a correct way.

The only time when the parameters are directly calculated is when x has only one column. In this case, you can calculate the parameter by simply doing

```r
b = sum(y*x)/sum(x^2)
```

In other situations, you simply use the following rule. b[k] is equal to the result of regressing the residual from regression of y on X[, -k] on the residual from regression of X[,k] on X[, -k].

Example of a recursive function: The following function calculates x^i for integer values of i

```r
pwr = function(x,i){
# if i==0, return 1
# if i>0, return x^(i-1) * x
# if i<0, return x^(i+1) / x
if (as.integer(i) == 0 ) return( 1)
else return(
        ifelse( i>0, pwr(x,i-1)* x , pwr(x,i+1) /x ))
}
# test
pwr(10 , -3)  # should be 0.001
```

```
## [1] 0.001
```

```r
pwr(2 , 5)  # should be 32
```

```
## [1] 32
```

Part 2: OLS using numerical optimization

You have two tasks here.

(1) Generate data for a simple $y = b_0 + \beta_1 x_1 + \beta_2 x_2 + \epsilon$ but in a way that your data violate the homoskedasticity assumption.

(2) Use bootstrap to estimate the model and compare your results with the results you get from `lm` if you ignore the violation of the assumption, and the result you get from robust standard errors.

Hint 1: You are encouraged to use the `boot` package for bootstrapping. You can find adequate description of the procedure in Chapter *7.10* in Field.

Hint 2: For robust standard errors, use the `sandwich` package. For example, use this along the `coeftest` function from the `lmtest` package. Something like `coeftest(your.model, vcov = vcovHC)`

Part 3: Logit

This is similar to Part 2, but we are going to maximize the likelihood function for the logistic regression. You should write a function named logit that takes y (binary dependent variable) and X (a matrix whose columns are the independent variables).

$$\mathrm{pr(y = 1)} = \frac{1}{1 + \exp(-X \times b)}$$

The $X \times b$ part is just $b_0 + \beta_1 x_1 + \cdots + \beta_k x_k$ in matrix notation. You can use `%*%` for matrix multiplication in R.

First, let us go through a simple explanation and example of maximum likelihood estimation (MLE).

Suppose we have a model about how the world works but we do not know the true value of the parameters (b) of the model. There is also some source of randomness in the world.

- We make observations from the world.

- We calculate how likely each observation would be in the world if the parameters were b.
- We take the logarithm of the likelihoods. (only for mathematical convenience).
- We add all of these log-likelihood's for the observations.
- *The logic of MLE is that the best estimate for the unknown parameters are values of b that maximize the total likelihood of observing what we have observed.*

This is quite simple, but may take a while to sink in. The following example should demonstrate the logic. We use `optim` for numerical optimization. See the previous supplementary exercise for more help on `optim`.

Example

Assume that we have a coin. The coin has an unknown parameter called tailiness which is the likelihood of observing tails when we toss it. Let b represent tailiness of our coin.

We make N observations of coin tosses. Each observation is coded as "1" if it is tails and "0" if it is heads. Let `y_i` represent this.

- Likelihood of observing 1 is `b` and Likelihood of observing 0 is `1-b`
- Therefore, we write `Ly = ifelse( y==1, b, 1-b )`
- Log Likelihood of observing the entire vector of y (assuming that coin tosses are independent of each other): `LLy = sum( log(Ly) )`

Now we can find the optimal solution:

```
# create the data first for tailiness=.333:
y = rbinom(n=100,p=.333,size=1)

LLy = function(b, y)
  return(   sum( log(ifelse(y==1, b, 1-b) ) ) )

b0 = 0.5 #starting value
# we are doing maximization with optim this time
# so use fnscale=-1

#ignore the warning that tells you the default method is
# not the best method for the problem at hand.

# If there is convergence, optim's output $par should be
# close to  0.333

result =optim(b0, fn = LLy, y=y, control = list( fnscale = -1 ) )
```

```
## Warning in optim(b0, fn = LLy, y = y, control = list(fnscale = -1)): one-dimensional optimization by
## use "Brent" or optimize() directly
```

```
result$convergence # did it converge
```

```
## [1] 0
```

```r
result$par # what is the result
```

```
## [1] 0.35
```

Now, we want to use MLE to estimate a logistic regression. Assume that the parameter to be estimated is b, independent variables form matrix X and observed binary outcomes are y. Using the following steps, write a function LL(b,y,X) which is then going to be maximized.

- What is the likelihood that y_i is 1?
- What is the likelihood that y_i is 0?
- What is the sum of log-likelihoods of observing all of y?
- Pick a good starting point for b
- Use optim to find the value of b that maximizes the likelihood of observing what we have observed.
- Report this value as b if there is convergence
- **And the most magical thing about MLE:** when using the `optim`, tell it that you want the Hessian back; use `optim(....., hessian=TRUE)`. This is the estimated second derivate of the objective function, which is a matrix because the function has multiple arguments. Then take the inverse of the negative of the Hessian. This is the variance-covariance matrix. You can take the square root of its diagonal to find the standard error for each parameter. You'll need something like `s.e.= sqrt(diag(solve(result$hessian)))`. Do this and compare it to what R reports through `summary(glm(....))`.