# ECON 390 ssignment 2 SID 2

## Problem Set Preparation

```
set.seed(123)
```

## Logic

We're going to "test" DeMorgan's laws empirically. Draw 100 simulations from N(2, 1) and call these "norm draws". Likewise, draw 100 simulations from Exp(2) called "exp draws". For the latter, use the function "rexp()" with a lambda of 2. Remember to look at the help documentation if you don't know how this works. Let the statements be defined as follows: • P is the statement norm draws $> 2$. • Q is the statement exp draws $> 1/2$

```
norm_draws = rnorm(100, mean = 2, sd = 1)
exp_draws = rexp(100, rate = 2)
p = norm_draws > 2
q = exp_draws > (1/2)
mean(p)
```

```
## [1] 0.52
```

```
mean(q)
```

```
## [1] 0.43
```

1. Not (P and Q) is the same as (Not P) or (Not Q)

```
mean(!(p & q) == (!p | !q))
```

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

2. Not (P or Q) is the same as (Not P) and (Not Q)

```
mean((!(p | q)) == (!p & !q))
```

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

## If Statements and Functions

1. Create your own absolute value function that takes in a single value and returns it's absolute value. Call it "my abs". Note: Do not use the abs() function.

```
my_abs = function(input) {
  return(if(input < 0) -1*input else input)
}
my_abs(-42)
```

```
## [1] 42
```

```
my_abs(25)
```

```
## [1] 25
```

2. Now, create your own absolute value function that takes in a vector and returns it's absolute value. Call it "my abs vec". Note: Do not use the abs() function.

```
my_abs_vec = function(inputVec) {
  count = 1;
  output = rep(0, length(inputVec))
  for(i in inputVec) {
    output[count] = my_abs(i)
    count = count + 1
  }
  return(output)
}
my_abs_vec(c(2,-5, 7, 9, -11, 10, -99))
```

```
## [1]  2  5  7  9 11 10 99
```

3. Create a function that returns "the sign" of a single number. That is, if the number is positive, return 1, if the number is negative, return -1, and if the number if 0, return 0. Call this function "my sign"

```
my_sign = function(num) {
  if(num == 0) {
    return(0)
  } else if (num > 0 ) {
    return(1)
  } else {
    return(-1)
  }
}
my_sign(0)
```

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

```
my_sign(-32)
```

```
## [1] -1
```

```
my_sign(5)
```

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

4. Create a function that returns the sign of a vector of numbers. Call this function "my sign vec".

```
my_sign_vec = function(input) {
  count = 1;
  output = rep(0, length(input))
  for(i in input) {
    output[count] = my_sign(i)
    count = count + 1
  }
  return(output)
}
my_sign_vec(c(2,-5, 0, 7, 9, -11, 10, -99))
```

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

5. Write a function called CRRA that takes in two inputs c and eta (spelled eta) and returns Note, if eta $< 0$, the function should error out. Likewise, you should write this function to be able to take in multiple values of c but only one value of eta.

```
CRRA = function(c, eta) {
  if(eta < 0 | !(length(eta) == 1)) {
    stop("invalid input")
  }
  count = 1;
  output = rep(0, length(c))
  for(i in c) {
    if(eta == 1) {
      output[count] = (i^(1-eta) - 1)/(1 - eta)
    } else {
      output[count] = log(i)
    }
    count = count + 1
  }
  return(output)
}
```

6. Create a function called "my funct" that calculates the following:

```
my_funct = function(x) {
  count = 1;
  output = rep(0, length(x))
  for(i in x) {
    if(i < 0) {
      output[count] = i^2 + 2*i + my_abs(i)
    } else if (i >= 0 & i < 2) {
      output[count] = i^2 + 3 + log(i + 1)
```

```
    } else {
      output[count] = i^2 + 4*i - 14
    }
    count = count + 1
  }
  return(output)
}
my_funct(c(1,2,3,4,5,6,7,-9))
```

```
## [1]  4.693147 -2.000000  7.000000 18.000000 31.000000 46.000000 63.000000
## [8] 72.000000
```

7. Create the following object and calculate the mean, median, min, max and standard deviation of each row and call. Name each output "x y" where x is the statistic that is being calculated and y is either row or column.

```
my_mat = matrix(c(rnorm(20,0,10), rnorm(20,-1,10)), nrow = 20, ncol = 2)
#row
mean_row = apply(my_mat, 1, mean)
median_row = apply(my_mat, 1, median)
min_row = apply(my_mat, 1, min)
max_row = apply(my_mat, 1, max)
sd_row = apply(my_mat, 1, sd)
#column
mean_column = apply(my_mat, 2, mean)
median_column = apply(my_mat, 2, median)
min_column = apply(my_mat, 2, min)
max_column = apply(my_mat, 2, max)
sd_column = apply(my_mat, 2, sd)
```

## An Economic Application of Loops: Auction Simulation

```
Nsim = 1000
Nbidders = 2
count = 1
winner = rep(0, Nsim)
winning_bid = rep(0, Nsim)
valuation = rep(0, Nsim)
while(count <= Nsim) {

  Delta = 0.1
  bid = Delta #start bidding with minimum bid
  bidder = as.numeric(runif(1) > 0.5) + 1 #decide who bids first
  Vs = exp(rnorm(Nbidders,mean = 1, sd = 1)) #draw values

  while(Vs[bidder] > bid + Delta){
    #continue to increase bid by minimum bid increment
    #until someone is no longer willing to bid more.
    bid = bid + Delta
    bidder = 3 - bidder
```

```
  }

  winner[count] = 3-bidder    #store winner
  winning_bid[count] = bid    #store winning bid
  valuation[count] = Vs[bidder]
  count = count + 1

}
```

1. Show me summary statistics of the winning bids i.e. use the summary() function on the winning bids and also show me the variance. Compare these to the mean and variance of the log normal distribution. Can you confidently say the distribution of the valuations and the distribution of the bids are different5?

```
summary(winning_bid)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   0.100   0.900   1.600   2.046   2.600  16.200
```

```
var(winning_bid)
```

```
## [1] 3.150291
```

2. Create a data set called "winners" that contains three variables for only winning bidders:

(a) The bidder number: called "bidder num,"
(b) Her valuation: called "valuation,"
(c) Her bid: called "bid."

```
winner = data.frame("bidder_num" = winner, "valuation" = valuation, "bid" = winning_bid)
```

3. Run a regression of "bid" on "valuation" (that is, bid is the dependent variable and valuation is the independent variable). Print the summary of the regression and interpret the coefficients.

```
attach(winner)
```

```
## The following objects are masked _by_ .GlobalEnv:
##
##     bid, valuation
```

```
model = glm(bid ~ valuation, data = winner)
summary(model)
```

```
##
## Call:
## glm(formula = bid ~ valuation, data = winner)
##
## Deviance Residuals:
##       Min         1Q     Median         3Q        Max
## -0.102957  -0.047632   0.002705   0.047014   0.098841
```

```
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 0.003419   0.002760   1.239    0.216
## valuation   0.999456   0.001020 979.729   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.003275317)
##
##     Null deviance: 3147.1406  on 999  degrees of freedom
## Residual deviance:    3.2688  on 998  degrees of freedom
## AIC: -2879.5
##
## Number of Fisher Scoring iterations: 2
```