# Applied Statistical Programming - Apply

Cassandra Custis, Berta Diaz, Zion Little, Alma Velazquez

2/16/2022

**Write the R code to answer the following questions. Write the code, and then show what the computer returns when that code is run. Thoroughly comment your solutions.**

You have until the beginning of class 2/21 at 10:00am to answer all of the questions below. You may use R, but not any online documentation. Submit the Rmarkdown and the knitted PDF to Canvas. Have one group member submit the activity with all group members listed at the top.

## A simulation experiment using `apply` & `plyr`

For this assignment, **you cannot use looping structures**. You will also need to create *arrays* to work with this problem. If you can imagine matrices layered on top of each other, this is an array. Use the following example to familiarize yourself with making arrays and referencing their values.

```r
# Create two vectors of different lengths
vector1 <- c(5, 9, 3)
vector2 <- c(10, 11, 12, 13, 14, 15)

# Put these vectors into an array of two 3x3 matrices.
result <- array(c(vector1, vector2), dim = c(3, 3, 2))
print(result)
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    5   10   13
## [2,]    9   11   14
## [3,]    3   12   15
```

```r
# Reference the 3rd column in matrix 2
result[, 3, 2]
```

```
## [1] 13 14 15
```

1. Make a three dimensional array with `dim=c(20,5, 1000)` and fill it with random data. Think of this as 1000 random datasets with 20 observations and 5 covariates.

```
# Create an array from 100 random values from 1:1000, recycled with the desired
# dimensions
a <- array(sample(1000, 100, replace = TRUE), dim = c(20, 5, 1000))

# Print the first entry to check it worked
a[, , 1]
```

```
##       [,1] [,2] [,3] [,4] [,5]
##  [1,]  173  470  874  463  195
##  [2,]  332  924  544  214  843
##  [3,]  392  157  352  409  761
##  [4,]  985   55  145  856  958
##  [5,]  875   22  423  637  340
##  [6,]  532  390  765  555  993
##  [7,]  734  439  324  304  383
##  [8,]  659  142  812  632  569
##  [9,]  429  776   36  114  565
## [10,]  205  111  226  435  200
## [11,]  451  967  793  690  560
## [12,]  595  873  437  186  785
## [13,]  782   99  317  283  449
## [14,]  430  318  180  626  528
## [15,]  486  723  638   49  890
## [16,]  635  685  728  460  331
## [17,]  656  398  976  462  918
## [18,]  173  377  225  811  719
## [19,]  988  835  637  581  252
## [20,]  265  671  875  781  501
```

2. Use the provided vector of linear model coefficients `Beta`. Make a function to create $Y$ values for a linear model. The $Y$ values should be a linear combination of the $X$'s plus some random noise. The output should be a 20 by 1000 array.

```
# Remove the eval=FALSE header from this code block before continuing
Beta <- matrix(c(1, 2, 0, 4, 0), ncol = 1)
X <- matrix(rnorm(100), ncol = 5, nrow = 20)
X[1, ]
```

```
## [1] -1.8353873 -0.3168628 -0.3276813  1.2948407  0.5709918
```

```
Beta
```

```
##      [,1]
## [1,]    1
## [2,]    2
## [3,]    0
## [4,]    4
## [5,]    0
```

```
X %*% Beta  # No noise has been included yet
```

```
##             [,1]
##  [1,]  2.7102499
##  [2,] -0.7440584
##  [3,] -4.0939691
```

```
##  [4,]  6.7346836
##  [5,] -0.6891670
##  [6,] -9.1834395
##  [7,]  0.4938458
##  [8,] -0.6506535
##  [9,] -0.2140471
## [10,] -0.1259487
## [11,] -5.3453218
## [12,]  8.8187272
## [13,]  1.7983921
## [14,] -6.8553482
## [15,]  2.2960476
## [16,]  4.8647075
## [17,] -4.0495211
## [18,]  4.2800416
## [19,] -0.2247824
## [20,]  1.3716074
```

```r
# Define a function that takes in a matrix of x values
make_output <- function(xmat) {
    # Create vector of noise whose length is the number of rows of the matrix
    # Noise will be Gaussian
    noise <- rnorm(nrow(xmat))

    # Define y as a linear combination of the matrix and Beta
    y <- xmat %*% Beta

    # Add noise to y
    y <- y + noise

    # Return the y
    return(y)

}

# Test it on the given X matrix,
make_output(X)
```

```
##              [,1]
##  [1,]  2.75771057
##  [2,] -0.57089426
##  [3,] -4.89210757
##  [4,]  4.84657022
##  [5,] -0.18287329
##  [6,] -9.92347065
##  [7,]  0.57905139
##  [8,]  0.00510494
##  [9,]  0.99594569
## [10,] -0.75650166
## [11,] -4.99661907
## [12,]  7.46827627
## [13,]  2.23164604
## [14,] -4.71271667
## [15,]  3.77647314
## [16,]  4.76452183
```

```
## [17,] -2.62637681
## [18,]  5.61626945
## [19,] -0.21996160
## [20,]  0.56915728
```

```
# and on the first entry of the array
make_output(a[, , 1])
```

```
##           [,1]
##  [1,] 2964.860
##  [2,] 3036.829
##  [3,] 2343.023
##  [4,] 4520.010
##  [5,] 3466.837
##  [6,] 3532.381
##  [7,] 2828.191
##  [8,] 3473.240
##  [9,] 2436.351
## [10,] 2164.969
## [11,] 5145.282
## [12,] 3083.973
## [13,] 2113.742
## [14,] 3570.248
## [15,] 2127.235
## [16,] 3844.252
## [17,] 3299.894
## [18,] 4170.577
## [19,] 4983.447
## [20,] 4730.183
```

```
# Use apply to make a Y vector for each 20x5 matrix in the array To the third
# dimension of a (each matrix), apply the make_output function
Y <- apply(a, 3, make_output)

# Check dimensions of Y
dim(Y)
```

```
## [1]   20 1000
```

```
# Look at the Y vector generated for the first matrix in a
Y[, 1]
```

```
##  [1] 2965.932 3036.423 2342.047 4519.687 3466.739 3532.069 2828.928 3470.276
##  [9] 2436.864 2165.130 5145.040 3085.314 2112.203 3569.621 2129.378 3844.178
## [17] 3300.176 4170.894 4982.979 4731.262
```

3. Run 1,000 regressions across all of this simulated data. Have as the output a 1000 by 6 matrix of estimated regression coefficients.

```
# We want to specify a regression for each index of the second dimension of Y
# and the third dimension of our array, a.
dim(a)
```

```
## [1]   20    5 1000
```

```
dim(Y)
```

```
## [1]   20 1000
```

```r
# Each regression will look like this test_lm <- lm(Y[,1] ~ a[,,1])

# Define a function that takes in an index
make_reg <- function(i) {
    # It will define an x matrix based on this index for the third dimension of
    # a,
    x <- a[, , i]
    # and a y vector based on this index for the second dimension of y
    y <- Y[, i]

    # Extract the coefficients from the lm object
    coeffs <- lm(y ~ x)$coefficients
    # Return the coefficients in a row-wise matrix
    return(t(matrix(coeffs)))

}

# Test the function on a random index between 1 and 1000
test <- make_reg(991)

test
```

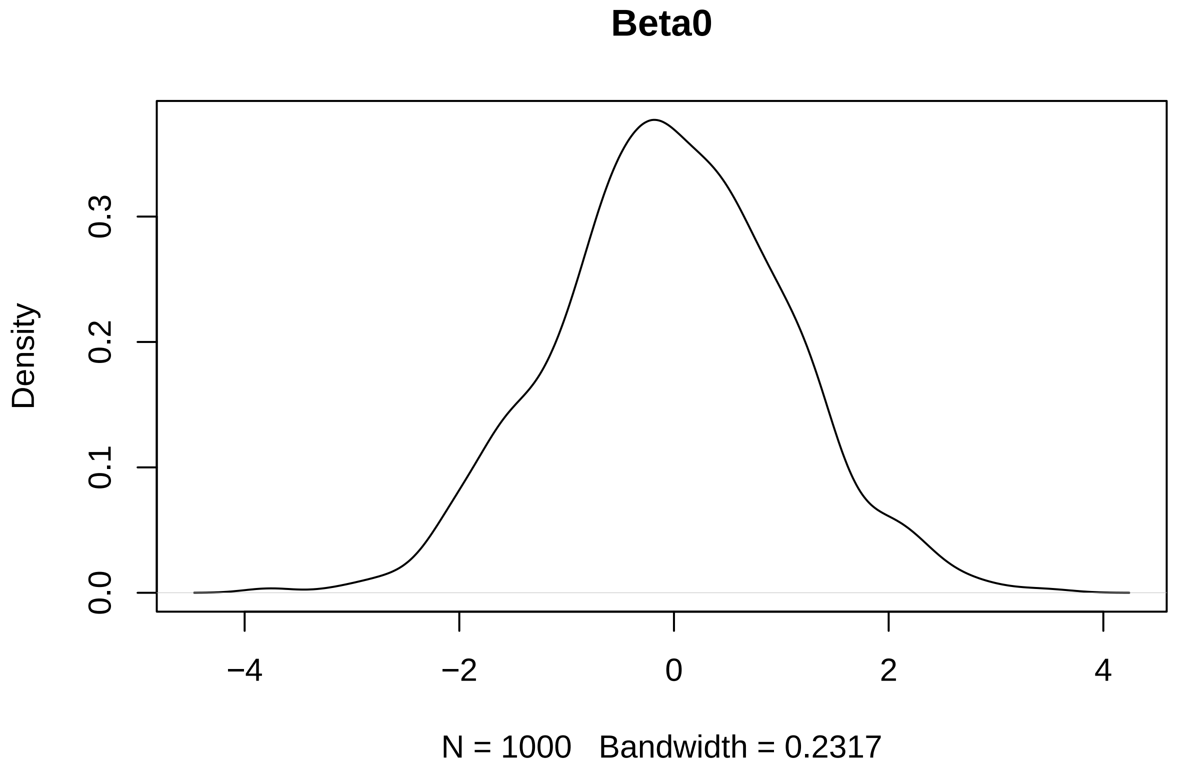
```
##           [,1]     [,2]     [,3]         [,4]     [,5]         [,6]
## [1,] -1.951808 1.002058 2.000363 0.0004878944 4.000213 0.0004111628
```

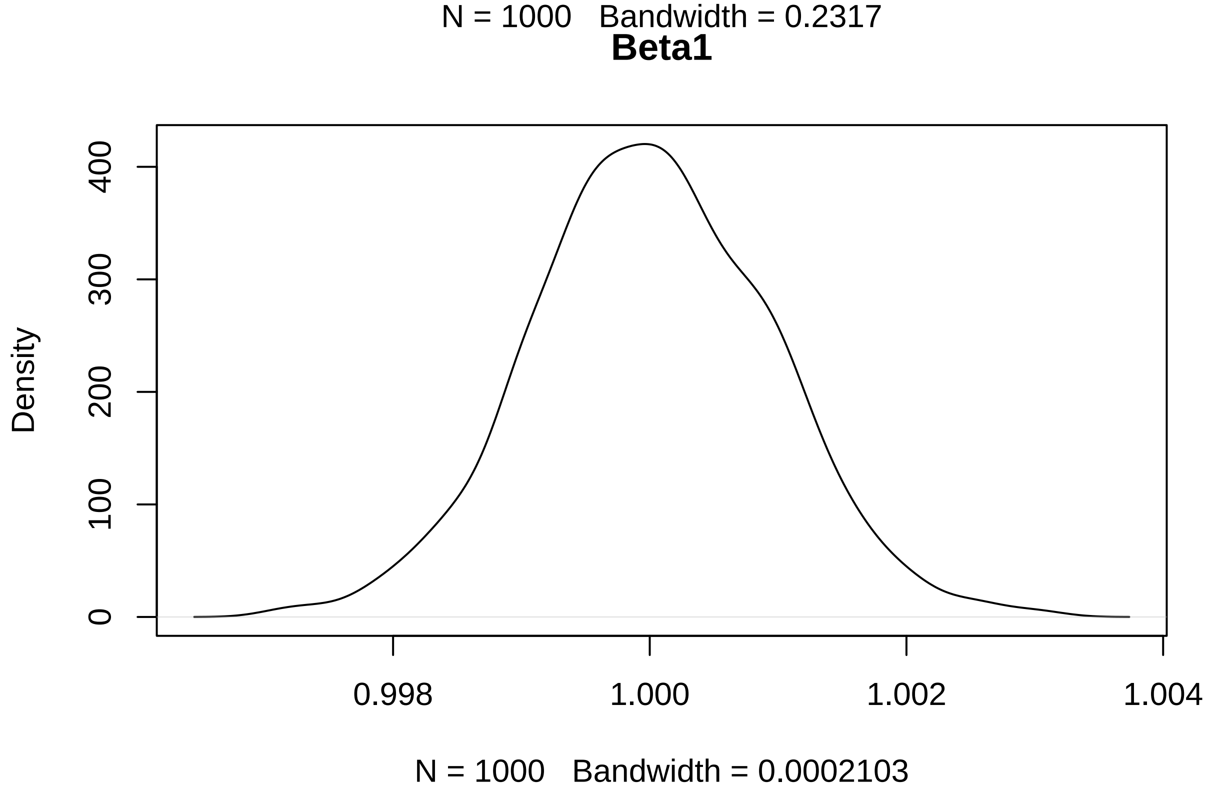
```r
# Since our function takes index inputs, we will be applying it to a vector of
# indices; we need lapply

# This lapply call generates a list of row matrices containing the 6
# coefficients for each regression

# lapply(c(1:1000), make_reg)


# Wrap this output in unlist and matrix to put it in the desired form
all_coeffs <- matrix(unlist(lapply(c(1:1000), make_reg)), ncol = 6, nrow = 1000,
    byrow = TRUE)

# Name the columns of the matrix informatively
colnames(all_coeffs) <- c("B0", "B1", "B2", "B3", "B4", "B5")
```

4. Create a density plot for each of the 6 coefficients, each of which should have been estimated 1,000 times in the previous step. Describe what the density plot represents.

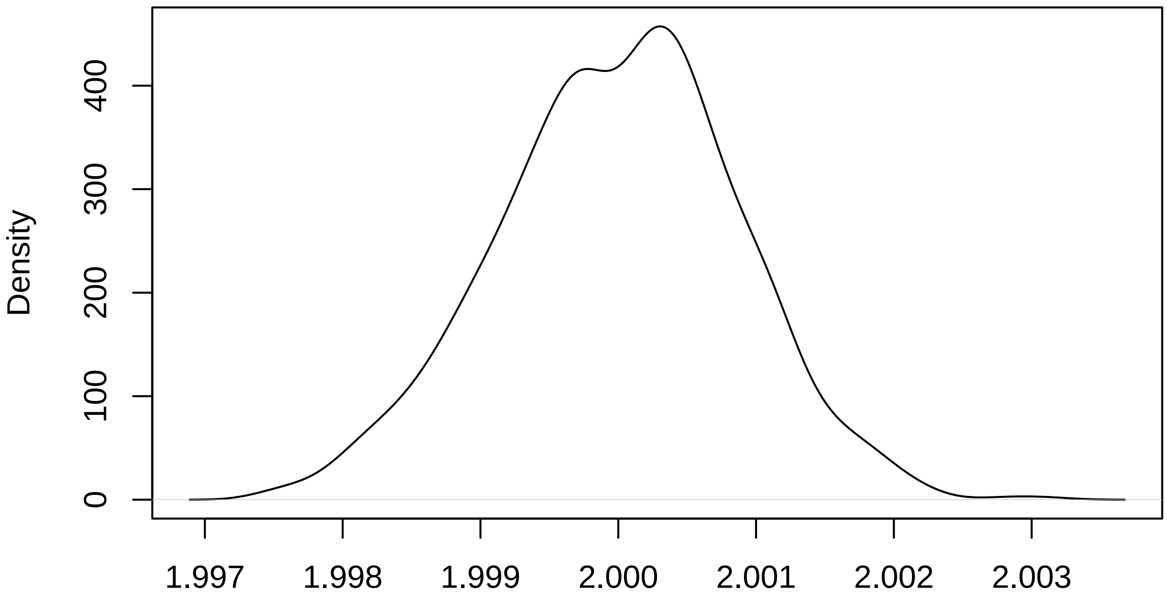
```r
# Test the plot call for one coefficient plot(density(all_coeffs[,6]))

# Define a function that again takes index inputs,
make_plot <- function(i) {
    coeff <- all_coeffs[, i]
    # plots densities of each coefficient, with named coefficient in title
    plot(density(coeff), main = paste0("Beta", (i - 1)))

}

# Again use lapply to make the function call for each of the 6 indices
```

```
lapply(c(1:6), make_plot)
```



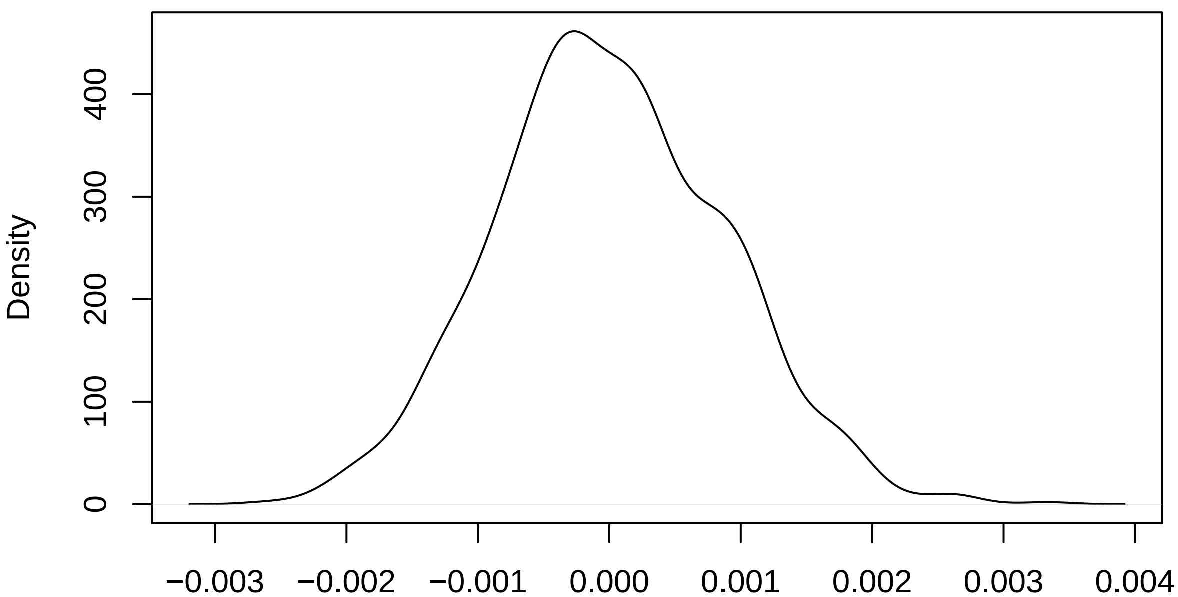



**Beta2**



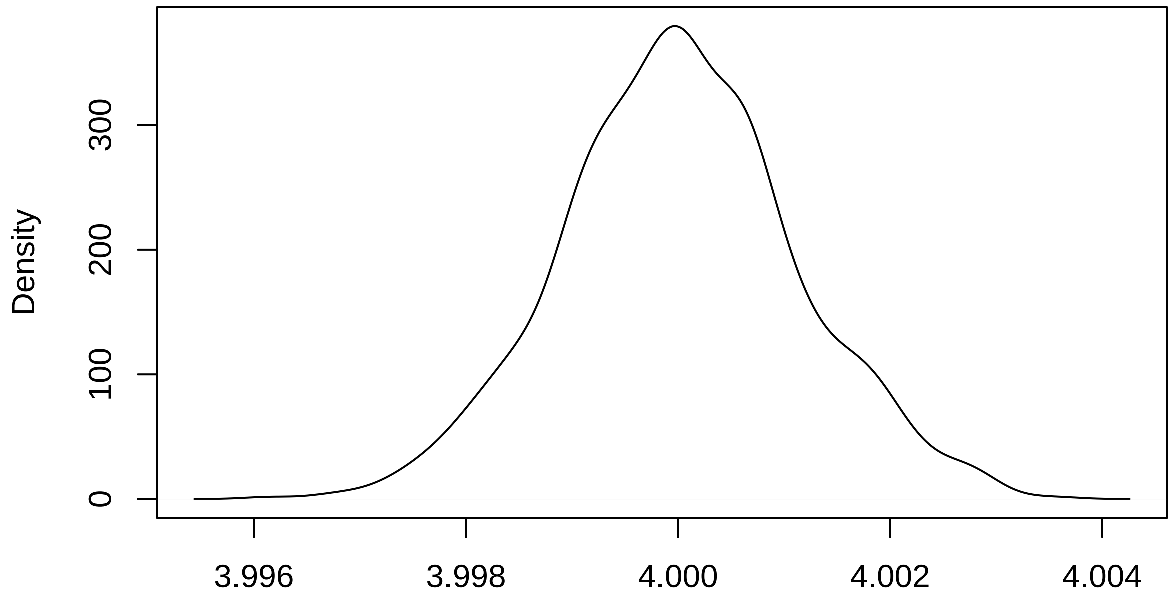

N = 1000 Bandwidth = 0.000196

**Beta3**



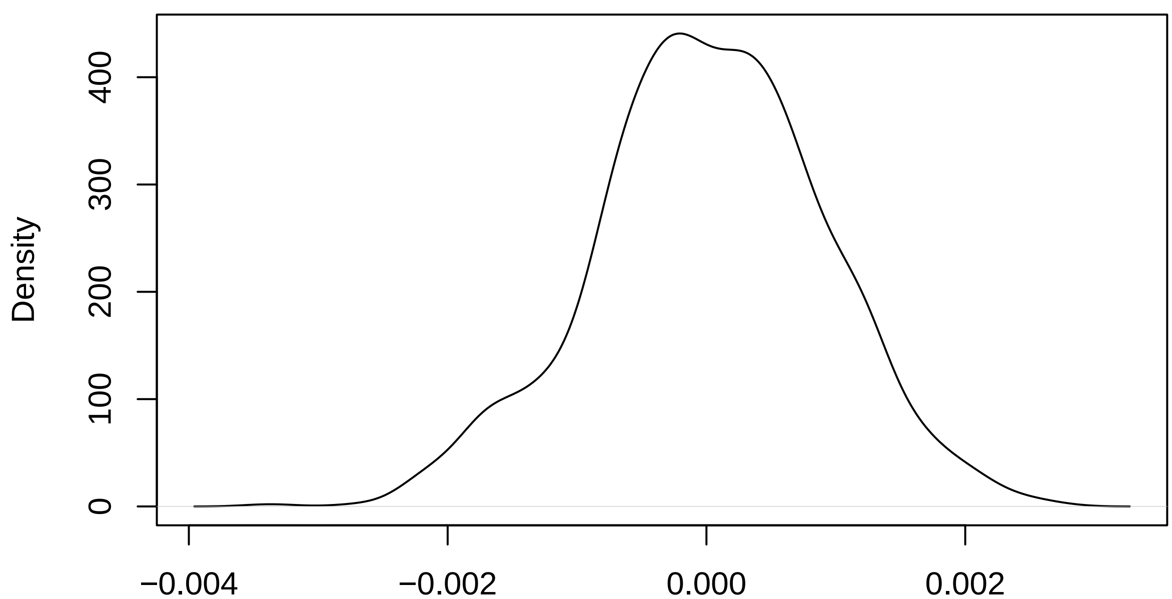

N = 1000 Bandwidth = 0.0001963

# Beta4




N = 1000   Bandwidth = 0.0002381

# Beta5




N = 1000   Bandwidth = 0.0001962

```
## [[1]]
## NULL
##
## [[2]]
## NULL
##
```

```
## [[3]]
## NULL
##
## [[4]]
## NULL
##
## [[5]]
## NULL
##
## [[6]]
## NULL
```

```
# Recall our Beta vector, out of which we made a linear combination of each x
# matrix Beta <- matrix(c(1,2,0,4,0), ncol=1)

### Interpretation: we added Gaussian noise to Y as a linear combination of X
### and Beta. Because of this, our coefficients are normally distributed around
### the original Beta values.
```

5. Re-run that code in parallel. Calculate the differences in run time.

```
library(plyr)
library(doMC)
```

```
## Loading required package: foreach

## Loading required package: iterators

## Loading required package: parallel
```

```
# See how long it took to run our 1000 regressions
system.time(matrix(unlist(lapply(c(1:1000), make_reg)), ncol = 6, nrow = 1000, byrow = TRUE))
```

```
##    user  system elapsed
##   0.810   0.008   0.829
```

```
# Now we'll split this up into more cores
registerDoMC(cores = 4)  # This was as many as my computer would let me do
system.time(matrix(unlist(lapply(c(1:1000), make_reg)), ncol = 6, nrow = 1000, byrow = TRUE))
```

```
##    user  system elapsed
##   0.803   0.007   0.819
```

```
# make_plot <- function(i){ coeff <- all_coeffs[,i] plot(density(coeff),
# main=paste0('Beta',(i-1))) } lapply(c(1:6), make_plot)
```