MLE Lab 2

Andy Ballard

January 19, 2017

Setting up your workspace

```
## Start with a clean workspace
rm(list=ls())

## Function to load packages
loadPkg <- function(toLoad){
  for(lib in toLoad){
    if(! lib %in% installed.packages()[,1])
      { install.packages(lib, repos='http://cran.rstudio.com/') }
    suppressMessages( library(lib, character.only=TRUE) ) }
}

## Load libraries
pkgs <- c("ggplot2", 'ggthemes', 'MASS', 'arm', 'lmtest')
loadPkg(pkgs)

## Set a theme for ggplot
theme_set(theme_bw())
theme_set(theme_economist())

## Functions that I use frequently
char <- function(x){ as.character(x) }
num <- function(x){ as.numeric(char(x)) }

## Relevant paths
if((Sys.info()['user']=='aob5' | Sys.info()['user']=='Andy')){
  lab2Path <- paste0(path.expand("~"), "MLE_LAB/Lab 2 - OLS and Hypothesis Testing")
}
```

```
## Load data
msrepPath <- paste0(lab2Path, "/Msrepl87.asc")
msrep <- read.table(msrepPath, header=TRUE)

## Create silly logged version of DV
msrep$deaths75ln <- log(msrep$deaths75+1) #Why add 1?

## Create logs of other things
msrep$deaths70ln <- log(msrep$deaths70+1)
msrep$sanctions75ln <- log(msrep$sanctions75+1)
msrep$sanctions70ln <- log(msrep$sanctions70+1)
msrep$energypcln <- log(msrep$energypc+1)


## Running a linear regression
ivs <- c('upper20', 'energypcln', 'intensep',
```

```
      'sanctions70ln', 'sanctions75ln', 'deaths70ln')
olsForm <- formula(
  paste0('deaths75ln ~ ',
         paste(ivs, collapse=' + ') )
  )
mod1 <- lm(olsForm, data=msrep)
```

```
## View model results
summary(mod1)
```

```
##
## Call:
## lm(formula = olsForm, data = msrep)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0584 -0.6402 -0.0818  0.8242  4.4539
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -7.51197    2.06031  -3.646 0.000592 ***
## upper20        0.10878    0.02385   4.561 2.89e-05 ***
## energypcln     0.27170    0.16737   1.623 0.110231
## intensep       1.17761    0.57111   2.062 0.043948 *
## sanctions70ln -0.32637    0.27122  -1.203 0.234005
## sanctions75ln  0.89604    0.23304   3.845 0.000315 ***
## deaths70ln     0.47747    0.11469   4.163 0.000111 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.429 on 55 degrees of freedom
##   (33 observations deleted due to missingness)
## Multiple R-squared:  0.7063, Adjusted R-squared:  0.6743
## F-statistic: 22.04 on 6 and 55 DF,  p-value: 5.034e-13
```

Handmade OLS Estimator

Now let's build an OLS estimator from scratch to estimate the regression we just did with `lm()`.

```
lm(deaths75ln ~ upper20 + energypcln + intensep + sanctions70ln + sanctions75ln + deaths70ln,
   data=msrep)
```

We will estimate the equation $Y = X\beta + \epsilon$, where Y is a vector of values for the dependent variable, X is a matrix of values for the independent variables (columns) for each case (rows), β is a vector of coefficients mapping the independent variables onto the dependent variable, and ϵ is a vector of residuals (also called the error term, or errors) that make up the difference between $X\beta$ and Y for each observation.

First, we will calculate the vector of coefficients $\hat{\beta}$, which is equal to

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

Matrix operators

Some matrix operation commands in R will be helpful:

- `as.matrix()` changes an object to the class 'matrix'.
- `t()` transposes a matrix, which is basically a reflection using the diagonal as an axis, and is denoted by an apostrophe in matrix notation (ex. $X'X$ is the transpose of X multiplied by X).
- `%*%` performs matrix multiplication.
- `solve()` takes the inverse of a matrix.

Estimate coefficients

Now we can find the coefficients, so that $\hat{\beta}$ minimizes the sum of squared residuals.

But first, we have to delete missing values from the relevant variables. NOTE: `lm()` will do this automatically, but what we're building right now isn't that sophisticated. ALSO IMPORTANT: When you're doing your own work, don't just drop observations automatically. Think about how it matters for each case.

```
## Drop NAs for variables

msrep <- msrep[!((is.na(msrep$energypcln) | is.na(msrep$intensep) |
                    is.na(msrep$upper20))),]
```

We deleted 33 observations, the same as lm(). I also cheated a bit, in that I already knew which of the relevant variables had missing values.

Back to our regularly scheduled programming...

```
## Make IV and DV matrices
X <- as.matrix(cbind(1, msrep$upper20, msrep$energypcln, msrep$intensep,
                     msrep$sanctions70ln, msrep$sanctions75ln,
                     msrep$deaths70ln)) #Why the 1?
Y <- as.matrix(msrep$deaths75ln) #Do we need as.matrix() here?


## Compute the coefficient estimates
bhat <- round( solve( t(X)%*%X ) %*% t(X)%*%Y, digits= 3 )
```

Calculate Standard Errors

We've already finished half the battle! We have our coefficient estimates, now we need to compute the standard errors.

To do this, we need the variance-covariance (vcov) matrix, which is:

$$Var(\hat{\beta}|X) = \frac{1}{n-k}\hat{\epsilon}'\hat{\epsilon}(X'X)^{-1}$$

Here, n is the number of observations, k is the number of right-hand variables (IVs plus intercept), and we already know about $\hat{\epsilon}$ and X. We find the standard errors by taking the square root of the diagonals of this k x k matrix.

```
## Define n, k, and residuals

res <- as.matrix(Y - bhat[1] - bhat[2]*X[,2] - bhat[3]*X[,3] - bhat[4]*X[,4] -
                   bhat[5]*X[,5] - bhat[6]*X[,6] - bhat[7]*X[,7])

n <- length(Y)
k <- ncol(X)

## Compute variance-covariance matrix
vcov <- 1/(n-k) * num(t(res)%*%res) * solve(t(X)%*%X)

## Compute standard errors of coefficients
stdErr <- round(as.matrix(sqrt(diag(vcov))), digits=3)
```

We have the coefficients and standard errors. What can we do with them? Let's first put everything in one place.

```
## Data frame for betas
beta.hat <- as.data.frame(cbind(c("Intercept", "Upper20", "EnergyPC.ln", "IntenSep",
                                  "Sanctions70.ln", "Sanctions75.ln", "Deaths70.ln"),
                                bhat))
names(beta.hat) <- c("Coef.", "Est")

## Combine with standard errors
model <- cbind(beta.hat, stdErr)
model
```

```
##            Coef.    Est stdErr
## 1      Intercept -7.512  2.060
## 2        Upper20  0.109  0.024
## 3    EnergyPC.ln  0.272  0.167
## 4       IntenSep  1.178  0.571
## 5 Sanctions70.ln -0.326  0.271
## 6 Sanctions75.ln  0.896  0.233
## 7    Deaths70.ln  0.477  0.115
```

Just for fun, let's compute p-values for a t-test

```
p.vals <- round(rbind(2*pt(abs(bhat[1]/stdErr[1]), df=n-k, lower.tail=F),
                2*pt(abs(bhat[2]/stdErr[2]), df=n-k, lower.tail=F),
                2*pt(abs(bhat[3]/stdErr[3]), df=n-k, lower.tail=F),
                2*pt(abs(bhat[4]/stdErr[4]), df=n-k, lower.tail=F),
                2*pt(abs(bhat[5]/stdErr[5]), df=n-k, lower.tail=F),
                2*pt(abs(bhat[6]/stdErr[6]), df=n-k, lower.tail=F),
                2*pt(abs(bhat[7]/stdErr[7]), df=n-k, lower.tail=F)), digits=3)


## Join with the rest of the model data
print(model <- cbind(model, p.vals)) #What does print() do here?
```

```
##            Coef.    Est stdErr p.vals
## 1      Intercept -7.512  2.060  0.001
```

```
## 2        Upper20  0.109  0.024  0.000
## 3    EnergyPC.ln  0.272  0.167  0.109
## 4       IntenSep  1.178  0.571  0.044
## 5 Sanctions70.ln -0.326  0.271  0.234
## 6 Sanctions75.ln  0.896  0.233  0.000
## 7    Deaths70.ln  0.477  0.115  0.000
```

How does this compare to the model from the `lm()` function?

```
summary(mod1)
```

```
##
## Call:
## lm(formula = olsForm, data = msrep)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0584 -0.6402 -0.0818  0.8242  4.4539
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -7.51197    2.06031  -3.646 0.000592 ***
## upper20        0.10878    0.02385   4.561 2.89e-05 ***
## energypcln     0.27170    0.16737   1.623 0.110231
## intensep       1.17761    0.57111   2.062 0.043948 *
## sanctions70ln -0.32637    0.27122  -1.203 0.234005
## sanctions75ln  0.89604    0.23304   3.845 0.000315 ***
## deaths70ln     0.47747    0.11469   4.163 0.000111 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.429 on 55 degrees of freedom
##   (33 observations deleted due to missingness)
## Multiple R-squared:  0.7063, Adjusted R-squared:  0.6743
## F-statistic: 22.04 on 6 and 55 DF,  p-value: 5.034e-13
```

Hot dang!

Gauss-Markov Assumptions

In order to make any meaningful inferences with an OLS regression model, the model must meet the Gauss-Markov assumptions:

1. $Y = X\beta + \epsilon$, or that the relationship between X and Y is linear.
2. $E[\epsilon|X] = 0$, or that the residuals are distributed evenly about zero, so that there is no bias in the estimates due to the residuals. This implies $E(Y) = X\beta$.
3. $Var(\epsilon|X) = E[\epsilon'\epsilon] = \sigma^2 I_n$, which means that there cannot be any correlation in the variance of the residuals across observations.
4. X is of full rank, or that none of the covariates are perfectly collinear.

How well does the model meet the G-M assumptions?

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

This is the most important of the mathematical assumptions for a linear model, and sensibly so. If the relationship between the IVs and the DV is not linear, our predictions from a linear model are likely to be seriously incorrect.

Oh god. How do I test for this?

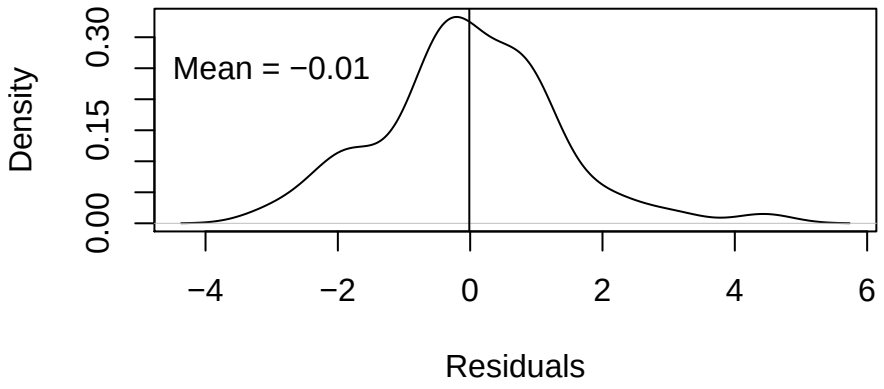
Luckily for all of us, linear models in R come with diagnostics in their output to test for violations of linearity. Simply `plot()` the model object:

```
plot(mod1)
```

Plots 1 (Fitted vs. Residual values) and 2 (Q-Q plot) are the most important here. For plot 1, we want residuals distributed about 0, relative to the fitted values. And for the second plot, we want the values to be as close to the diagonal as possible. For each, the model actually does pretty well (no models we encounter in the wild will ever be perfect in this regard).

2. $E[\epsilon|X] = 0$

The residuals should be distributed about zero, and they basically are!



Does that also mean that the expectation of Y is equal to XB minus the residuals? By definition, yes. How can we check if two quantities are equal?

```
## Difference in E(Y) and E(XB) is the same as E(residuals)
identical(round(mean(Y) - mean(X %*% bhat), 5), round(mean(res), 5))
```

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

or

```r
round(mean(Y) - mean(X %*% bhat), 5) == round(mean(res), 5)
```

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

3. $Var(\epsilon|X) = E[\epsilon'\epsilon] = \sigma^2 I_n$

Serial autocorrelation is generally indicative of a misspecified model, sometimes due to a lack of linearity, sometimes due to over/underfitting your data. It is a more serious problem for time series analysis than any other type of social science modeling.

One way to test for autocorrelation is to create another OLS model using the residuals from the model of interest. If there is a significant relationship, that is evidence of autocorrelation

```r
summary(mod2 <- lm(res[-n] ~ res[-1]))
```

```
##
## Call:
## lm(formula = res[-n] ~ res[-1])
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.0578 -0.5891  0.0047  0.7782  4.5088
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -0.004017   0.175045  -0.023    0.982
## res[-1]      0.125701   0.129825   0.968    0.337
##
## Residual standard error: 1.367 on 59 degrees of freedom
## Multiple R-squared:  0.01564,    Adjusted R-squared:  -0.001043
## F-statistic: 0.9375 on 1 and 59 DF,  p-value: 0.3369
```

Another way is to use a Durbin-Watson Test, `dwtest()` in the `lmtest` package. Again, a significant relationship is evidence of autocorrelation.

```r
dwtest(mod1)
```

```
##
##  Durbin-Watson test
##
## data:  mod1
## DW = 1.7361, p-value = 0.1061
## alternative hypothesis: true autocorrelation is greater than 0
```

We good.

4. No perfect collinearity among IVs

This is easy to test for, just look at how correlated the IVs are.

```
cor(X[,2:ncol(X)]) #Leaves out the first column (could also say X[,-1])
```

```
##             [,1]        [,2]        [,3]        [,4]       [,5]
## [1,]  1.00000000 -0.53781026  0.03672114 -0.12802055 -0.1500299
## [2,] -0.53781026  1.00000000 -0.18262410  0.05329668  0.1414829
## [3,]  0.03672114 -0.18262410  1.00000000  0.40389952  0.3586674
## [4,] -0.12802055  0.05329668  0.40389952  1.00000000  0.8284819
## [5,] -0.15002986  0.14148295  0.35866736  0.82848189  1.0000000
## [6,]  0.07588335 -0.35715825  0.47230890  0.65588469  0.4763908
##             [,6]
## [1,]  0.07588335
## [2,] -0.35715825
## [3,]  0.47230890
## [4,]  0.65588469
## [5,]  0.47639081
## [6,]  1.00000000
```

The two sanctions variables are pretty highly correlated, which could be a problem if there were no theoretical reason to use a pseudo-lagged variable design as Muller and Seligson have done. Otherwise, we're solid.

Visualizing Regression Results

Tables suck, lets get a visualization of the coefficient results.

```
coefplot(mod1)
```

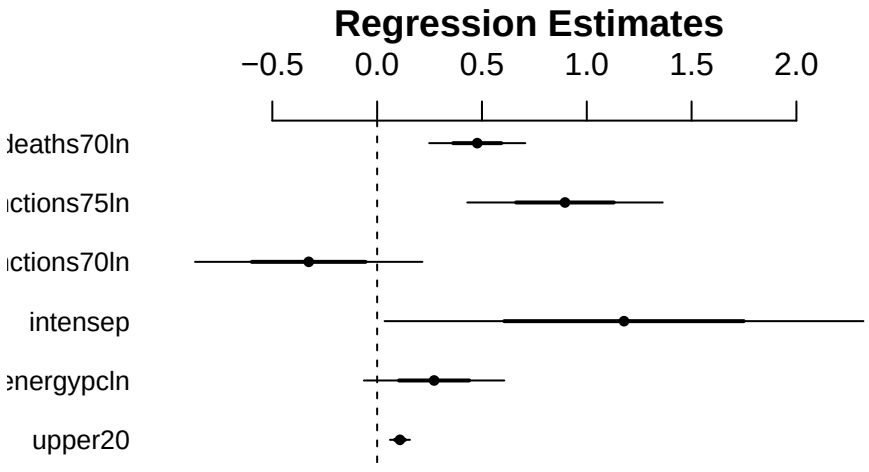


Figure 1: Ugly coefficient plot, and the labels do not even fit (believe me, I tried)

Lets make a nicer looking coefficient plot. First we need to generate the data to plot. What data do we need?

```
# Pull out relevant info from model object
varName <- rownames(summary(mod1)$coefficients)
varName <- c('Intercept', 'Upper 20% Income Share, 1970',
          'Ln(Energy Cons.), 1970', 'Separatism, 1975',
          'Ln(Sanctions), 1968-72', 'Ln(Sanctions), 1973-77',
          'Ln(Deaths), 1968-72')
mean <- summary(mod1)$coefficients[,1]
s.err <- summary(mod1)$coefficients[,2]

# Calculate confidence intervals around regression estimates
up95 <- mean+qnorm(0.975)*s.err
lo95 <- mean-qnorm(0.975)*s.err
up90 <- mean+qnorm(0.95)*s.err
lo90 <- mean-qnorm(0.95)*s.err

# Combine in dataframe
coefData <- cbind(varName, mean, s.err, up95, lo95, up90, lo90)
coefData <- data.frame(coefData, row.names=NULL)

# Check class of variables in dataframe
class(coefData$up95)
```

```
## [1] "factor"
```

```
# Lets clean this up (factors are often awful to work with)
convNumDcol=function(data, vars){
  for(var in vars){ data[,var]=num(data[,var]) }
    return( data ) }
coefData=convNumDcol(coefData, names(coefData)[2:length(coefData)])

# Lets check to make sure class is correct now
class(coefData$up95)
```

```
## [1] "numeric"
```

Now lets produce a nicer looking coefficient plot.

```
mod1.coefplot <- ggplot(data=coefData[-1,], aes(x=varName))+
  geom_linerange(aes(ymin=lo95, ymax=up95), size=.3) +
  geom_linerange(aes(ymin=lo90, ymax=up90), size=1) +
  geom_point(aes(y=mean)) +
  geom_hline(yintercept=0, linetype=2, color = "red") +
  coord_flip() + xlab('') + ylab('') +
  theme(axis.ticks=element_blank())
mod1.coefplot
```

Meaningfully interpreting marginal effects

Our primary concern should not be whether the variables we throw into a regression are significant. If you deal with big enough datasets, finding significant relationships is trivial (and misleading).

Figure 2: Economist coefficient plot

What we actually care about are the substantive implications of the independent variables that we are testing. For example, in the case of Muller & Seligson's model, it is clear that if the upper 20% have a greater share of income, we can expect some uptick in political deaths, but by how much does this variable need to change for us to see an increase in political deaths?

There are many ways at getting this question. The simplest is to just start looking at marginal effects. Our regression model has the following linear form:

$$deaths75ln \;=\; \beta_0 + \beta_1 upper20 + \beta_2 energypcln + \ldots + \beta_6 deaths70ln$$

To determine the marginal effect of the upper20 variable, we simply take the partial derivative of deaths75ln with respect to upper20 and find:

$$\frac{\partial deaths75ln}{\partial upper20} \;=\; 0.1088$$

Thus a one unit change in upper20, holding all else constant, produces a 0.1088 point change in the number of logged deaths from political violence per 1m. That's great, but we still don't know if this is a meaningful change.

Lets go back to our data. A shift from the 25th to 75th percentile in upper20 is equal to 14.675. This translates to a 1.596276 change in our dependent variable.

How meaningful is this change? Well, here's a boxplot of our dependent variable, so you tell me.

```
dv.boxplot <- ggplot(msrep, aes(x=factor(0), y=deaths75ln)) +
  geom_boxplot()+
```

```
  coord_flip() +
  xlab('') +
  scale_x_discrete(breaks=NULL) +
  ylab('Ln(Deaths from Political Violence per 1m), 1973-77')
dv.boxplot
```

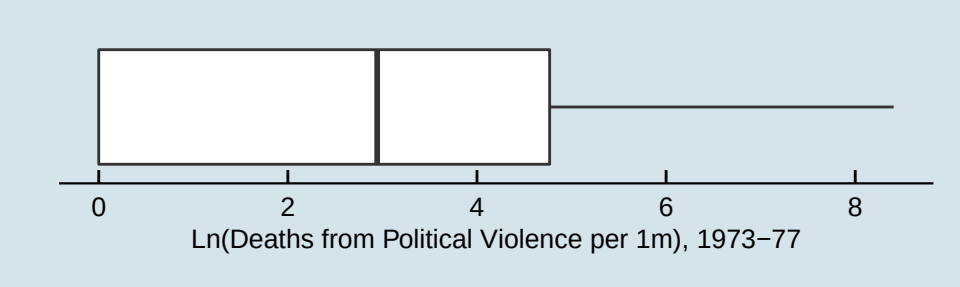


Figure 3: DV boxplot

Meaningfully interpreting coefficient estimates via simulations

In making predictions about the effect of the `upper20%` independent variable, we have completely ignored the standard errors from our coefficients. Additionally, we don't at all account for the uncertainty from the model itself.

We'll use simulations to take uncertainty into account. What are simulations?

```
set.seed(1988)
```

Birthday problem

```
sims <- 1000
people <- 23
days <- seq(1,365,1)
sameday <- 0

for(i in 1:sims){
  birthdays <- sample(days, people, replace=T)
    if(length(unique(birthdays)) < people) sameday <- sameday + 1
}

paste0("Probability at least 2 people with same birthday = ", sameday/sims ,"")
```

```
## [1] "Probability at least 2 people with same birthday = 0.543"
```

Ahem. . . uncertainty

Our regression results contain information about both the point estimates of our parameters and the standard errors of each of those parameters. We can use this information to generate predictions that take into account the uncertainty of our regression coefficients. For this example, we'll look at two examples: a country with high income inequality versus one with low income inequality.

```r
# Scenario 1: High income inequality...other vars at central tendency
# Scenario 2: Low income inequality...other vars at central tendency
means <- apply(msrep[,ivs], 2, function(x){ mean(x, na.rm=TRUE) })
minMaxSep <- quantile(msrep[,ivs[1]], probs=c(0,1), na.rm=TRUE)

scens <- rbind(c(1, minMaxSep[1], means[2:length(means)]),
               c(1, minMaxSep[2], means[2:length(means)]) )

# Simulate additional parameter estimates from multivariate normal
sims <- 1000
draws <- mvrnorm(sims, coef(mod1), vcov(mod1))

# Get predicted values using matrix multiplication
preds <- draws %*% t(scens)

# Plotting sim results
plot(density(preds[,1]), col = "red", bty = "n",
     las = 1, xlim=c(-4, 10), lwd = 3, main='', ylim=c(0,1),
     xlab = "Logged Average Deaths per Million")
lines(density(preds[,2]), col = "blue", lwd = 3)
legend(x=-.4,y=.95,legend=c("Low upper20"),
       text.col=c("red"),bty="n", cex = 0.75)
legend(x=3.7,y=0.95,legend=c("High upper20"),
       text.col=c("blue"),bty="n", cex = 0.75)
```

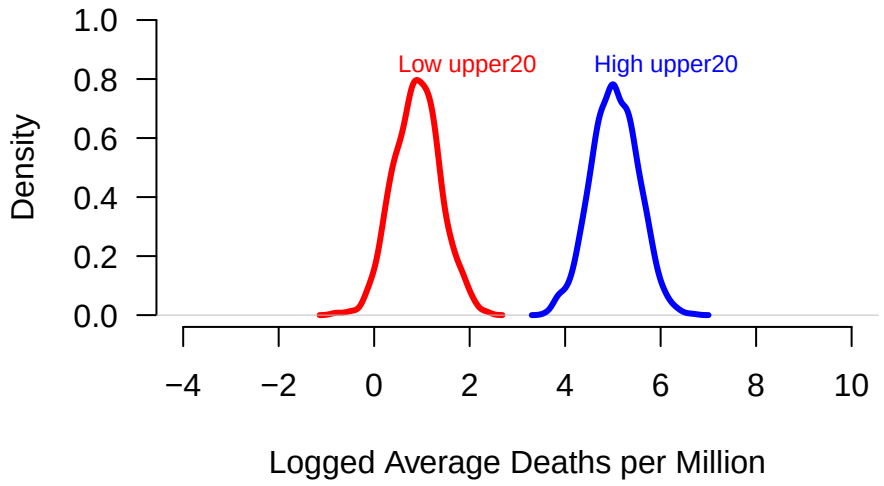


Figure 4: Simulation with Systematic Uncertainty

We have taken into account the systematic component of the uncertainty in our regression model but we should also take into account fundamental uncertainty.

```r
# Model uncertainty
sigma <- sqrt(sum(resid(mod1)^2)/df.residual(mod1))

# Generating expected values
exp <- apply(preds, 2, function(x){
  rnorm(sims, x, sigma)
})

# Repeat code from before
plot(density(preds[,1]), col = "red", bty = "n",
     las = 1, xlim=c(-4, 10), lwd = 3, main='', ylim=c(0,1),
     xlab = "Logged Average Deaths per Million")
lines(density(preds[,2]), col = "blue", lwd = 3)
legend(x=-.4,y=.95,legend=c("Low upper20"),
       text.col=c("red"),bty="n", cex = 0.75)
legend(x=3.7,y=0.95,legend=c("High upper20"),
       text.col=c("blue"),bty="n", cex = 0.75)
# Add lines with fundamental uncertainty
lines(density(exp[,1]), col='coral', lwd=3)
lines(density(exp[,2]), col='cornflowerblue', lwd=3)
```

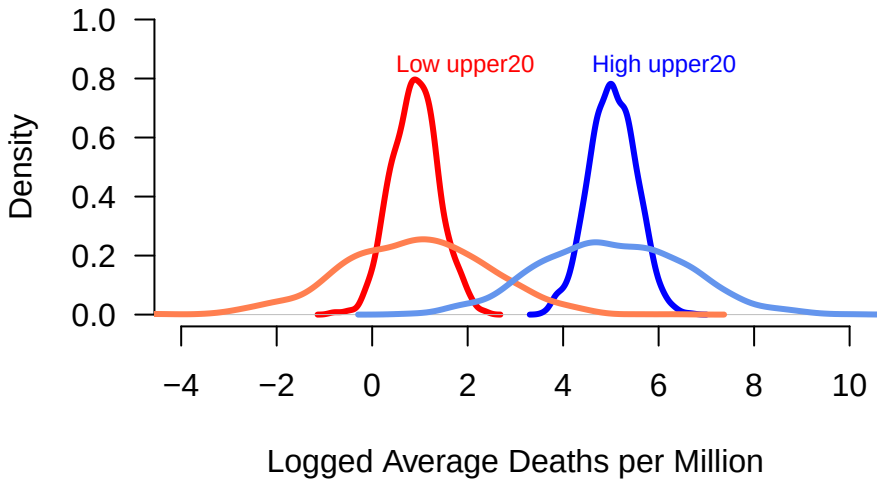


Figure 5: Simulation with Fundamental Uncertainty