MLE - Lab 6

Andy Ballard

February 17, 2017

House Keeping

- Your assignments are due at 11pm tonight! The system doesn't accept late submissions, so please submit on time
- Your first paper is due March 10th, 3 weeks from today (also the start of spring break)
- Be thinking about your final papers! Be efficient with your final papers!
- Think of ways you can work more exclamation points into your life!

Today

- Simulations. (The period means it's important)
- No autocorrelation? What gives? Daniel wants to do a section on this in lecture before we go over it in lab

As always, we'll start by setting up a clean workspace

Also, let's get y'all some help that I probably should have pointed you toward earlier. here is a PDF cheat sheet with lots of functions you'll want to know in R.

Simulations

What are simulations?

The essential idea of simulation is using repeated random sampling to solve problems. Now that's just vague enough to be confusing; let's see what we really mean. Before we do anything, we need to set a starting point for our random processes: setting a seed.

Why does setting a seed matter?

```r
norm.draws <- rnorm(100,5)
mean(norm.draws)
```

```
## [1] 5.003399
```

```r
norm.draws2 <- rnorm(100,5)
mean(norm.draws2)
```

```
## [1] 5.040433
```

Now those are close, but they aren't the same. Setting a seed ensures that our random processes produce the exact same results if somebody else runs our code (or if we have to run it again).

```r
set.seed(1845)
norm.draws <- rnorm(100,5)
mean(norm.draws)
```

```
## [1] 4.904649
```

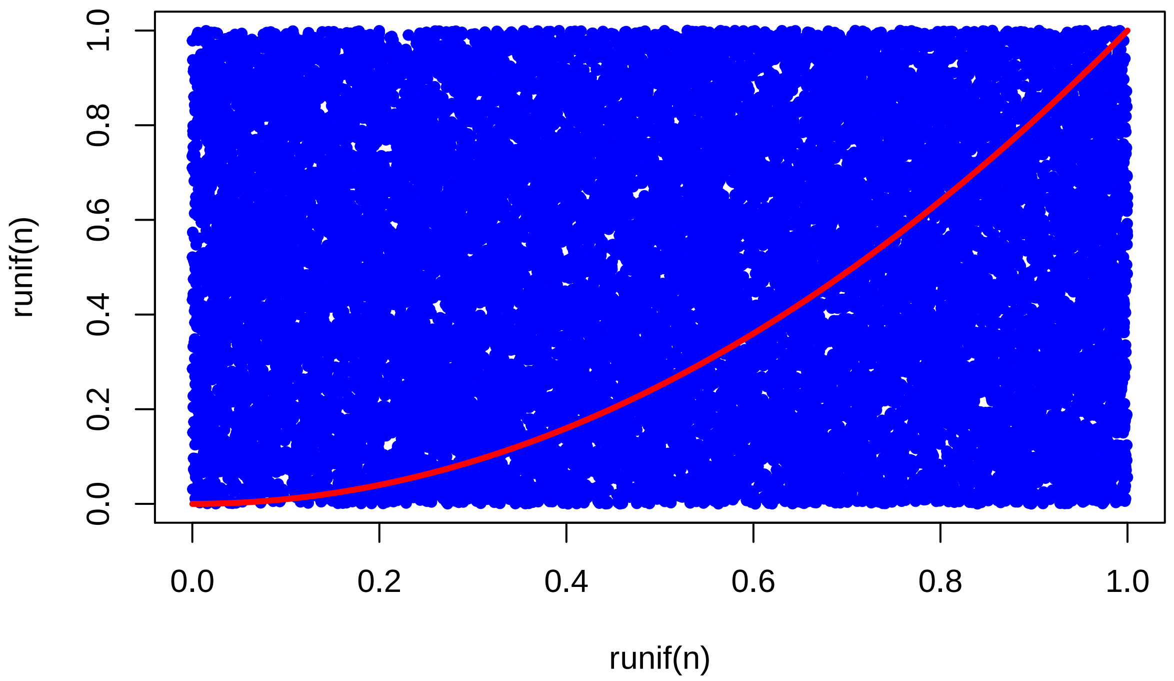
```r
set.seed(1845)
norm.draws2 <- rnorm(100,5)
mean(norm.draws2)
```

```
## [1] 4.904649
```

See? The exact same.

We can use simulations to do all sorts of things. We've already gone through a few examples (birthday problem, Monty Hall problem), but we can estimate just about anything. One common use for simulations is to estimate anlaytically intractable integrals, which some of you may get into if you go deeper into game theory or Bayesian statistics (in fact, just about everything in Bayesian statistics is done via simulation). How would we do this? Let's look at a simple example, say $y = x^2$.

```r
n <- 20000
f <- function(x) x^2
plot(runif(n), runif(n), col='blue', pch=20)
curve(f, 0, 1, n=100, col='red', add=TRUE, lwd=3)
```

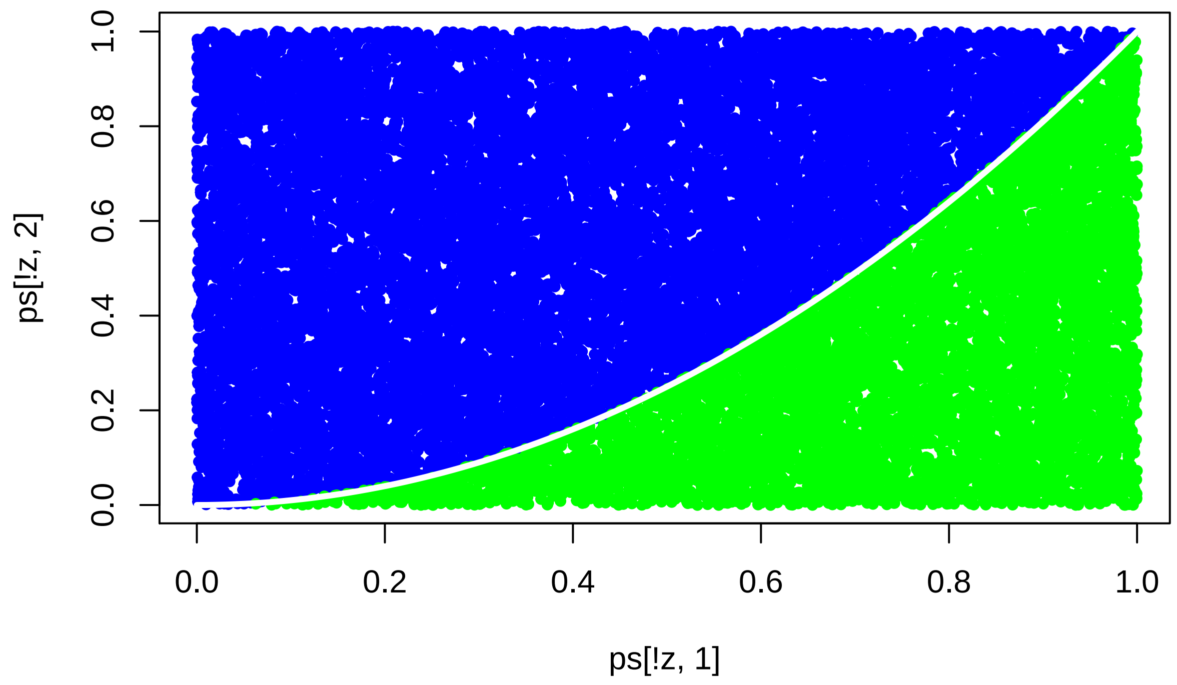


What did we just do?

Okay, so how do we get from points in a box to an approximation of an integral? We've been told before that an integral is the same as the area under a curve. So we need to find a way to separate out the points under the curve $y = x^2$ from those above it.

```r
ps <- matrix(runif(2*n), ncol=2)
g <- function(x,y) {y <= x^2}
z <- g(ps[,1], ps[,2])
plot(ps[!z,1], ps[!z,2], col='blue', pch=20)
```

```r
points(ps[z,1], ps[z,2], col='green', pch=20)
curve(f, 0,1, n=100, col='white', add=TRUE, lwd=3)
```



We've graphically estimated the integral, so how do we get a numerical estimate? Well, we know that $\int x^2 df = \frac{x^3}{3}$, which from 0 to 1 equals $\frac{1}{3}$. To get a numerical estimate of the integral, we can just take the number of green points and divide it by the total number of points to get the integral estimate.

```r
length(z[z])/n
```

```
## [1] 0.33425
```

Pretty good! If we increased the number of iterations, the estimate would be closer to $\frac{1}{3}$.

Oh boy, this is fun. What else can we do?

Let's estimate another quantity, this time π. First, we'll estimate $\pi/4$ (one-quarter of the unit circle) and then we can multiply by 4

```r
g <- function(k) {
  n <- 10^k
  f <- function(x,y) sqrt(x^2 + y^2) <= 1
  z <- f(runif(n),runif(n))
  length(z[z]) / n
}
a <- sapply(1:7, g)
a*4
```

```
## [1] 2.800000 3.120000 3.184000 3.097600 3.149200 3.140260 3.140727
```

These are reasonably close, but we can do better with more iterations. As you can see above, the estimates get better as we go from left to right. That's because each is doing 10^k estimates, and we're varying k from 1 to 7.

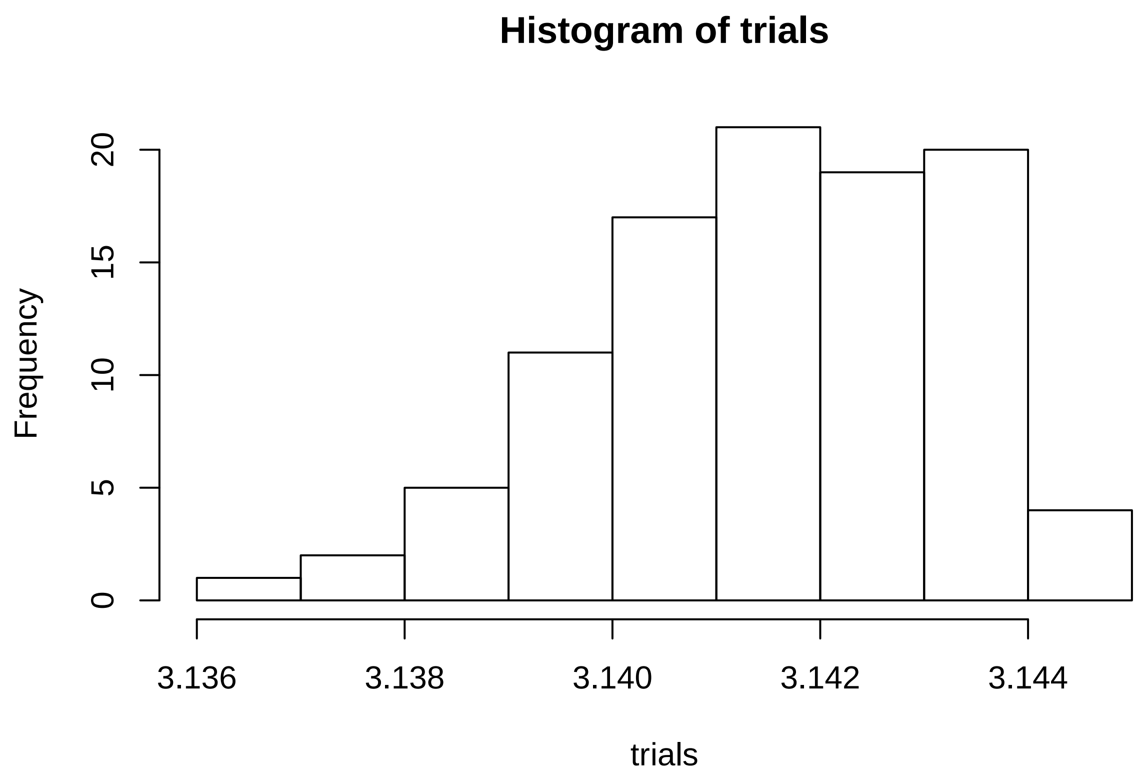
```r
trials <- 4 * sapply(rep(6,100), g)
e <- 1/sqrt(10^6)
mean(trials)
```

```
## [1] 3.141546
```

```r
length(trials[abs(trials - pi)/pi <= e])
```

```
## [1] 96
```

```r
hist(trials)
```



We took 100 samples of size 6 (each of these being samples of 10^6) and then estimated the mean of each of our 100 samples. (What is `e` doing here?)

This is all well and good, but how can it make my life as a social scientist easier?

Let's revisit Muller and Seligson. We re-ran their OLS model looking at political deaths as a function of various characteristics, which is below.

```r
# Load data
msrep <- read.table(paste0(path.expand("~"), "/MLE_LAB/Lab 6/Msrepl87.asc"), header=TRUE)

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

# Create logs of other things
msrep$deaths70ln <- log(msrep$deaths70+1)
```

```r
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)
```

```r
# 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
```

The interpretation of OLS coefficients is straightforward. For a one unit increase in an independent variable, there is a coefficient-sized change in the dependent variable. For instance, for a one unit increase in `upper20`, there is a +0.108 change in `deaths75ln`. The p-value tells us that if this data generating process were repeated infinitely many times, nearly 100 percent of our regressions would show a positive relationship between `upper20` and `deaths75ln`, but that doesn't mean there is no uncertainty about our coefficient estimates. We have standard errors! We can use these standard errors to estimate the systematic uncertainty in our model (the uncertainty around the model coefficients) with simulations.

As before, we will simulate two scenarios and plot them in what we creatively call substantive effects plots. Here is the basic process:

- Find some scenario for which you want to estimate an effect. This could be looking along the length of some continuous variable, comparing specific characteristics of an observation of relevance, comparing the minimum and maximum values of an IV (as we do here), and much more.

- Set up the scenario by creating values of “data” to pass through the simulation.
- Simulate many different estimates with your, using random draws with information from your regression model
- Visualize (or present the result in some way, but visualization is always good)

First, let’s define our scenario. We’ll compare the effect of the minimum income inequality versus the effect of the maximum income inequality on logged deaths.

To make our new “data”, we need to set values for all the other IVs in our model. The convention is to hold everything we aren’t varying to its central tendency (usually median or mean).

```
# 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)]) )
```

Now that we have our scenarios set up, we can make our draws. Here, we’ll take 1000 draws from the multivariate normal distribution, with means equal to the coefficients in our model, and variance equal to the variance-covariance matrix from our model.

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

Now that we have our coefficient draws, we multiply each of our 1000 draws by the data scenarios we’ve set up. Note, this looks an awful lot like $X\beta$!

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

Now that we have our predicted values, we can plot them.

```
#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)
```

We have taken into account the systematic component of the uncertainty in our regression model but we should also take into account stochastic (sometimes called “fundamental”) uncertainty.

Basically, we’re also taking into accoutn how well our model fits the data. First, we’ll calculate another variance parameter based on the square root of the SSR divided by the degrees of freedom.

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

Now we can take this model uncertainty into account with a new set of random draws, this time from a normal distribution centered on the predictions we already drew, but with the variance parameter for the whole model that we just created.

```
# Generating expected values
exp <- apply(preds, 2, function(x){
```

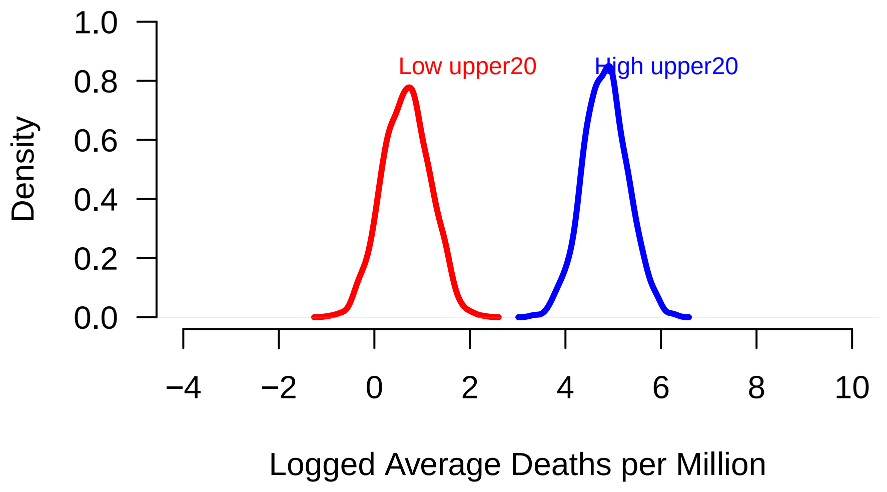
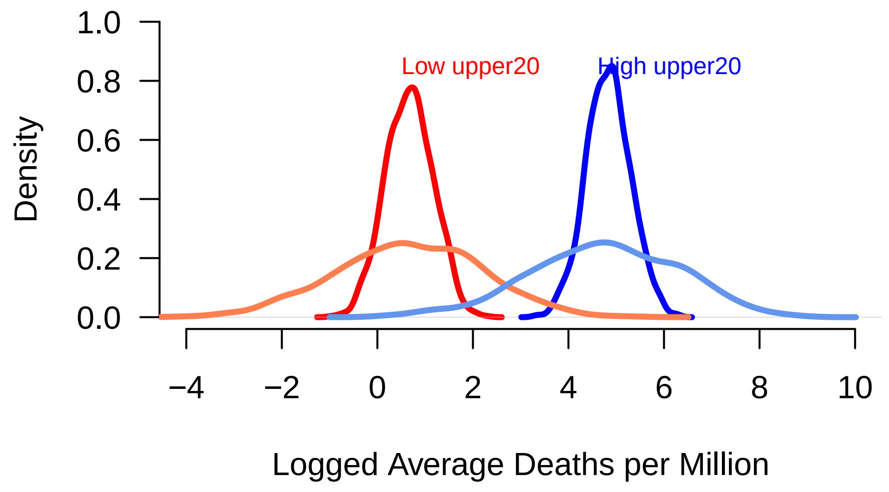


Figure 1: Simulation with Systematic Uncertainty

```r
  rnorm(sims, x, sigma)
})
```

Lastly, we can plot these effects.

```r
# 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)
```



Clearly, when we take model fit into account as well, we are adding more uncertainty to our estimates.

So hopefully now you have a better understanding of simulations and why they make life better.

But wait, isn't this an MLE class?

Yes, yes it is. Let's look at some simulations for MLE.

```
###################################################
# Load Data
data <- read.dta("http://www.indiana.edu/~jslsoc/stata/spex_data/binlfp2.dta")
    # DV: female labor force participation (lfp)
    # IVs: age (age), log wage (lwg), household income (inc)
table(data$lfp)
```

```
##
## NotInLF    inLF
##     325     428
```

```
data$lfp <- as.numeric(data$lfp)-1
```

Here is a data set. Now let's say we want to estimate female participation in the labor force as a function of age, logged wages, and household income. We learned earlier how to write our own MLE estimators, but we can just use a canned function now. We will estimate the model implied above using a logistic regression.

Remember, this means that we will estimate our betas via maximizing a likelihood function (in this case, the binomial I believe), and then passing our $X\beta$s through a logistic CDF link function.

```
### GLM function
m2 = glm(lfp ~ age + lwg + inc, data=data, family=binomial(link="logit"))
summary(m2)
```

```
##
## Call:
## glm(formula = lfp ~ age + lwg + inc, family = binomial(link = "logit"),
##     data = data)
##
```

```
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.6622  -1.2106   0.7918   1.0173   1.9462
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  0.813310   0.442895   1.836 0.066306 .
## age         -0.019783   0.009440  -2.096 0.036111 *
## lwg          0.753688   0.143352   5.258 1.46e-07 ***
## inc         -0.025338   0.006826  -3.712 0.000206 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1029.7  on 752  degrees of freedom
## Residual deviance:  984.0  on 749  degrees of freedom
## AIC: 992
##
## Number of Fisher Scoring iterations: 4
```

It looks like all of our variables are significant, but do they actually matter? Furthermore, the interpretation of logistic regression coefficients is not nearly as straightforward as in OLS. Using the simulation framework we used with the OLS regression above, we can create predicted probabilities for some scenarios we care about, and put our coefficients into terms that are much more easily interpertable. In this instance, let's say we want to know more about the substantive effect of logged wages on female labor force participation.

But first, a note on interpreting logistic regression coefficients:

The coefficients from a logistic regression in `R` are log odds. The logistic regression coefficients give the change in the log odds of the outcome for a one unit increase in the predictor variable. So for our model:

- For every one unit change in `age`, the log odds of labor force participation (versus non-participation) decrease by 0.019
- For every one unit change in `lwg`, the log odds of labor force participation increase by 0.754
- For every one unit change in `inc`, the log odds of labor force participation decrease by 0.025

You can also get the normal odds ratios by exponentiating the coefficients:

```r
m2.OR <- exp(coef(m2))
m2.OR
```

```
## (Intercept)         age         lwg         inc
##   2.2553603   0.9804111   2.1248211   0.9749800
```

Some people think these are more easily interpretable. This changes the interpretation to the following:

- For every one unit change in `age`, the odds of participating in the labor force decrease by a factor of 0.98 (barely a change)
- For every one unit change in `lwg`, the odds of participating in the labor force increase 2.12 times
- For every one unit change in `inc`, the odds of participating in the labor force decrease 0.97 times

Okay, back to our simulations.

We've thought about our scenario, now we need to assign data values.

```r
# We hold age and inc at their mean while varying logged wages
lwgRange <- seq(min(data$lwg), max(data$lwg), length.out=100)
scen <- with(data=data, cbind(1, mean(age), lwgRange, mean(inc)) )
```

Next, we take the scenario we've created and multiply it by the coefficients in our model to get predicted values.

```r
# Calculate predicted log odd values
predVals <- scen %*% coef(m2)
head(predVals) #is something horrible wrong? These aren't between 0 and 1
```

```
##           [,1]
## [1,] -2.086433
## [2,] -2.046289
## [3,] -2.006146
## [4,] -1.966002
## [5,] -1.925859
## [6,] -1.885716
```

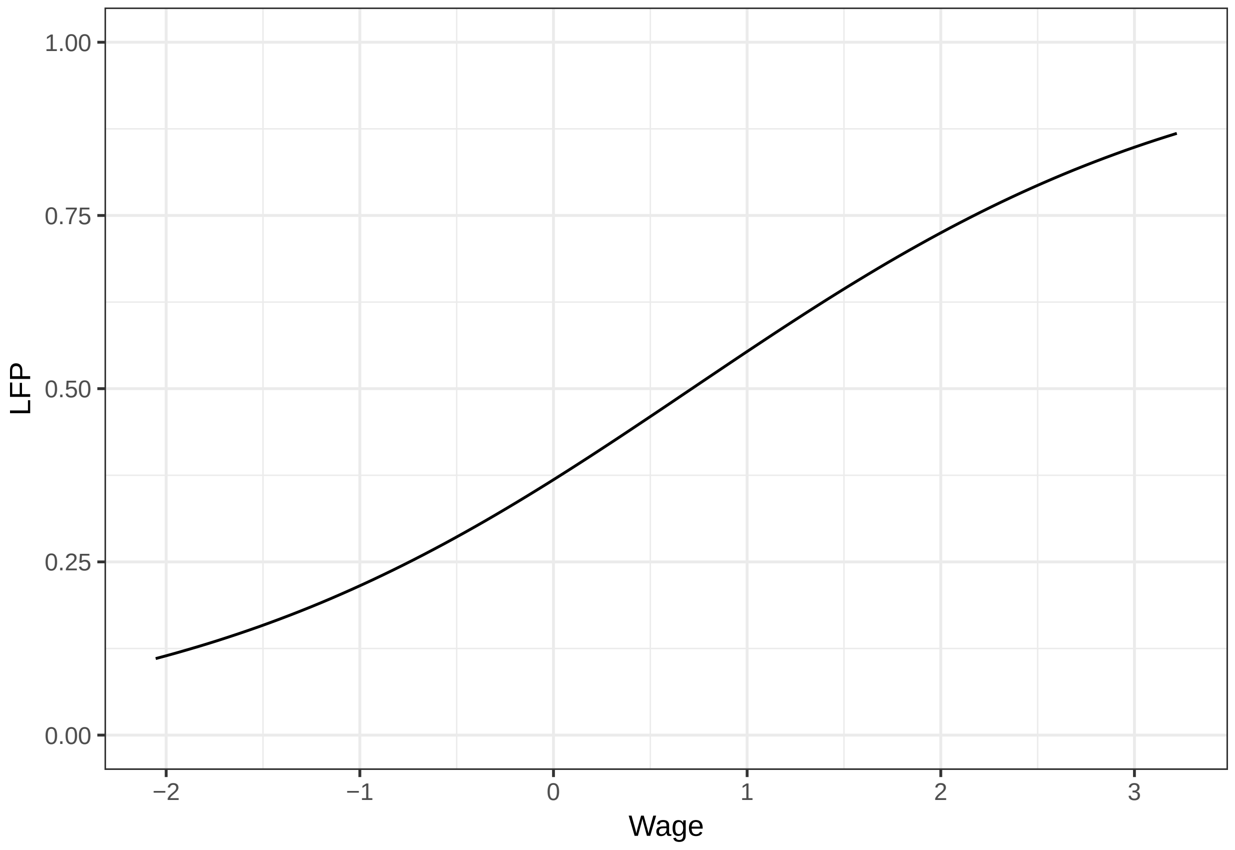
Unlike OLS, we need to pass these predicted values through the link function in order to map the predicted values onto the possible range of values in the logistic CDF, which we are using to estimate the binary data generating process (one either participates in the labor force, or they don't).

```r
# Apply link function to get predicted probabilities
predProbs <- 1/(1 + exp(-predVals))
head(predProbs)
```

```
##           [,1]
## [1,] 0.1104225
## [2,] 0.1144279
## [3,] 0.1185592
## [4,] 0.1228189
## [5,] 0.1272096
## [6,] 0.1317337
```

Much better. Plot time!

```r
# Plot result
ggData <- data.frame(lwgRange, predProbs)
names(ggData) <- c('Wage', 'LFP')
ggplot(ggData, aes(x=Wage, y=LFP)) + geom_line() + ylim(0, 1)
```



Nice! But where the the uncertainty? Let's add some confidence intervals with the `predict()` function, which if we specify `se.fit=TRUE` will help us out with the confidence intervals. If not, the `predict()` function will give us exactly what we plotted above.

```
# Add confidence intervals
predScen <- data.frame(scen)
names(predScen) <- names(coef(m2))
preds = predict(m2,
    type='link', se.fit=TRUE,
    newdata=predScen )
critVal <- qnorm(0.975) #for 95% CIs
predInts <- data.frame(
    LFP=preds$fit,
    up95=preds$fit + (critVal * preds$se.fit),
    lo95=preds$fit - (critVal * preds$se.fit)
    )
```

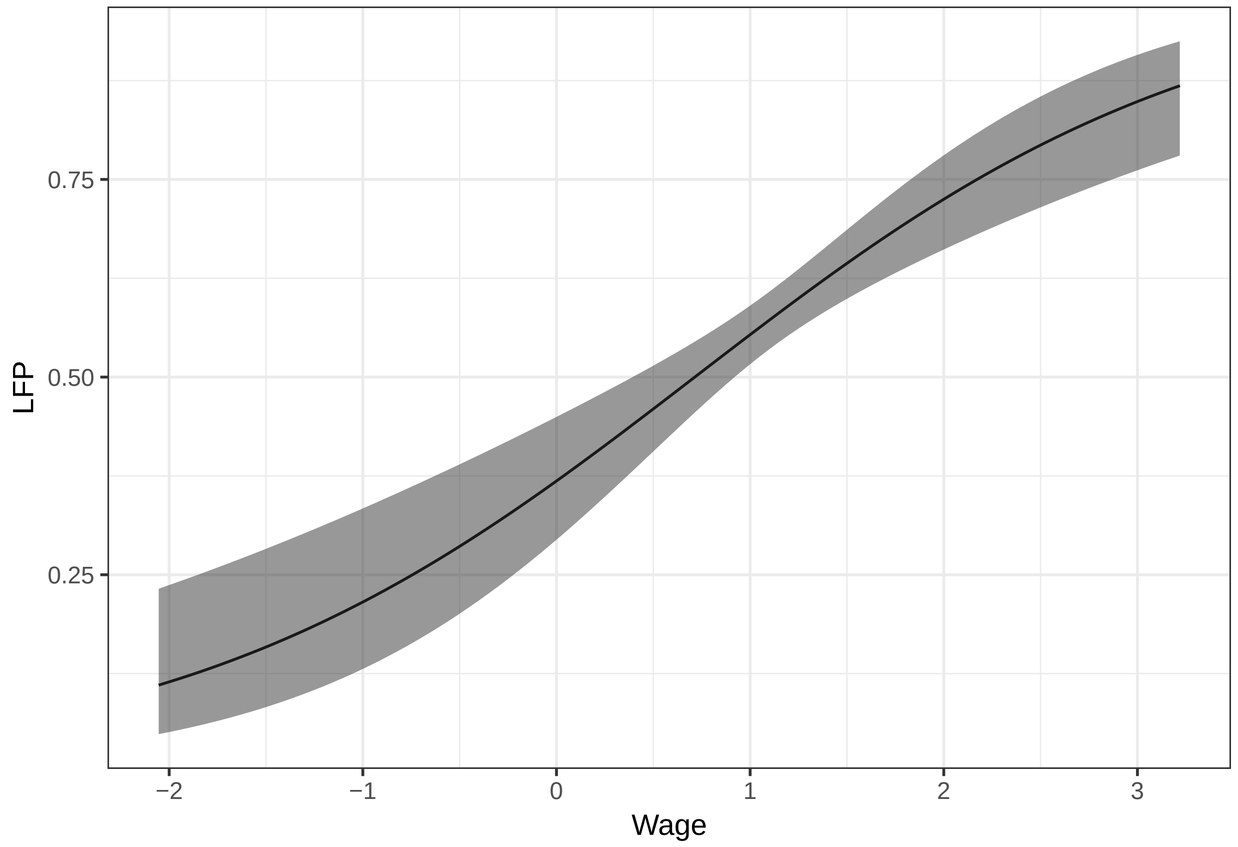
Note that the `newdata=predScen` argument allows us to make predictions from the model `m2` with our new scenario, which we've defined in the data frame `predScen`. Again, these values we've come up with aren't between 0 and 1, so we need to link them.

```
# Convert to predicted probabilities
predInts <- apply(predInts, 2, function(x){ 1/(1 + exp(-x)) })
```

Aaaaand plot.

```
# Now plot with conf int
ggData <- data.frame(Wage=lwgRange, predInts)
```

```
ggplot(ggData, aes(x=Wage, y=LFP, ymin=lo95, ymax=up95)) + geom_line() + geom_ribbon(alpha=.5)
```



Now this is all well and good, and R did some simulations under the hood, but it's good to be able to see exactly what's going on here. So let's do a simulation approach by hand. We've already defined our scenario, so we can get right to taking our multivariate normal draws, using the information from our model.

```
## Simulation based approach
sims <- 1000
draws <- mvrnorm(sims, coef(m2), vcov(m2))
predUncert <- draws %*% t(scen)
```

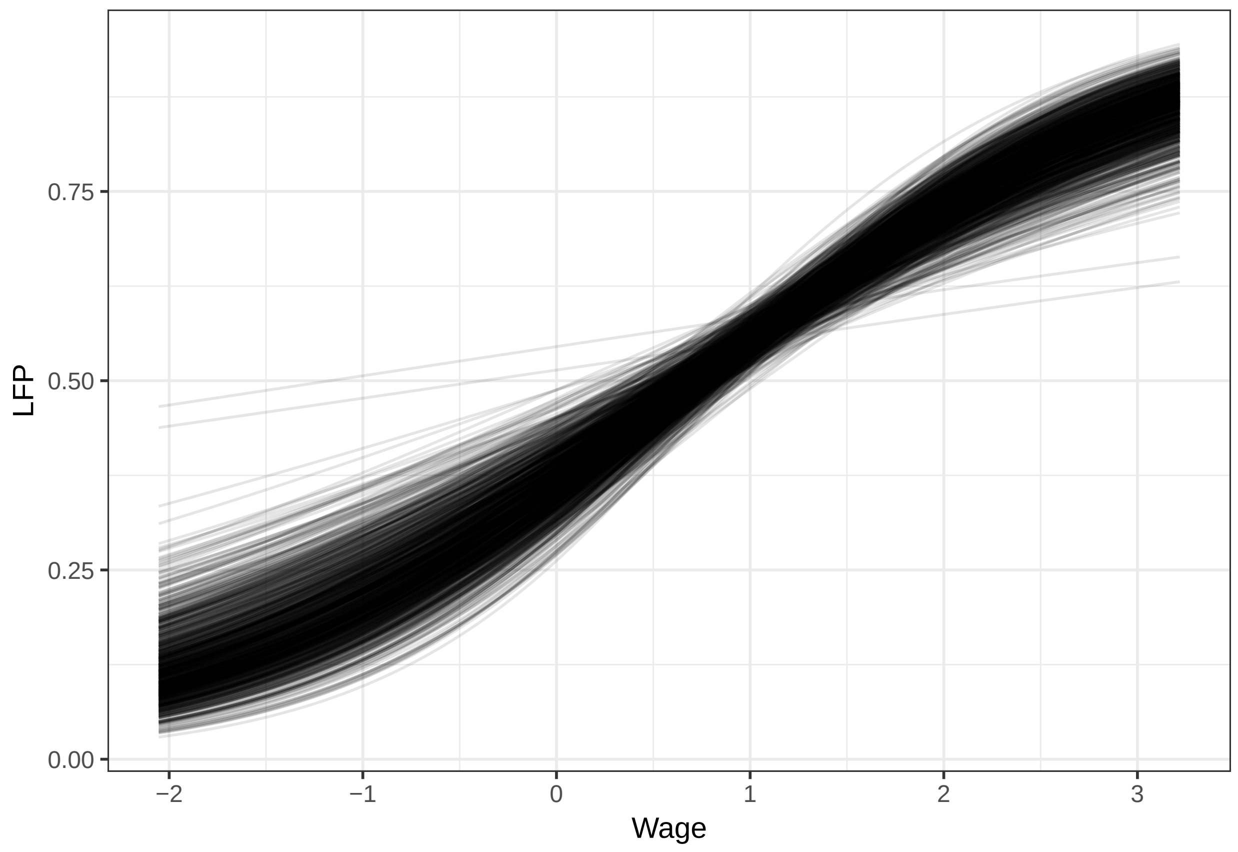
After we get predicted values from our draws, we use the link function to turn them into predicted probabilities.

```
# Convert to predicted probabilities
predUncert <- apply(predUncert, 2, function(x){ 1/(1+exp(-x)) })
```

There are a couple of ways to plot this. The first we'll look at is called a spaghetti plot, because it looks like spaghetti. We are a clever bunch.

To make this plot, we take all of the simulations and plot them as separate lines. As you can see, it looks much the same as the above plot using the `predict()` function.

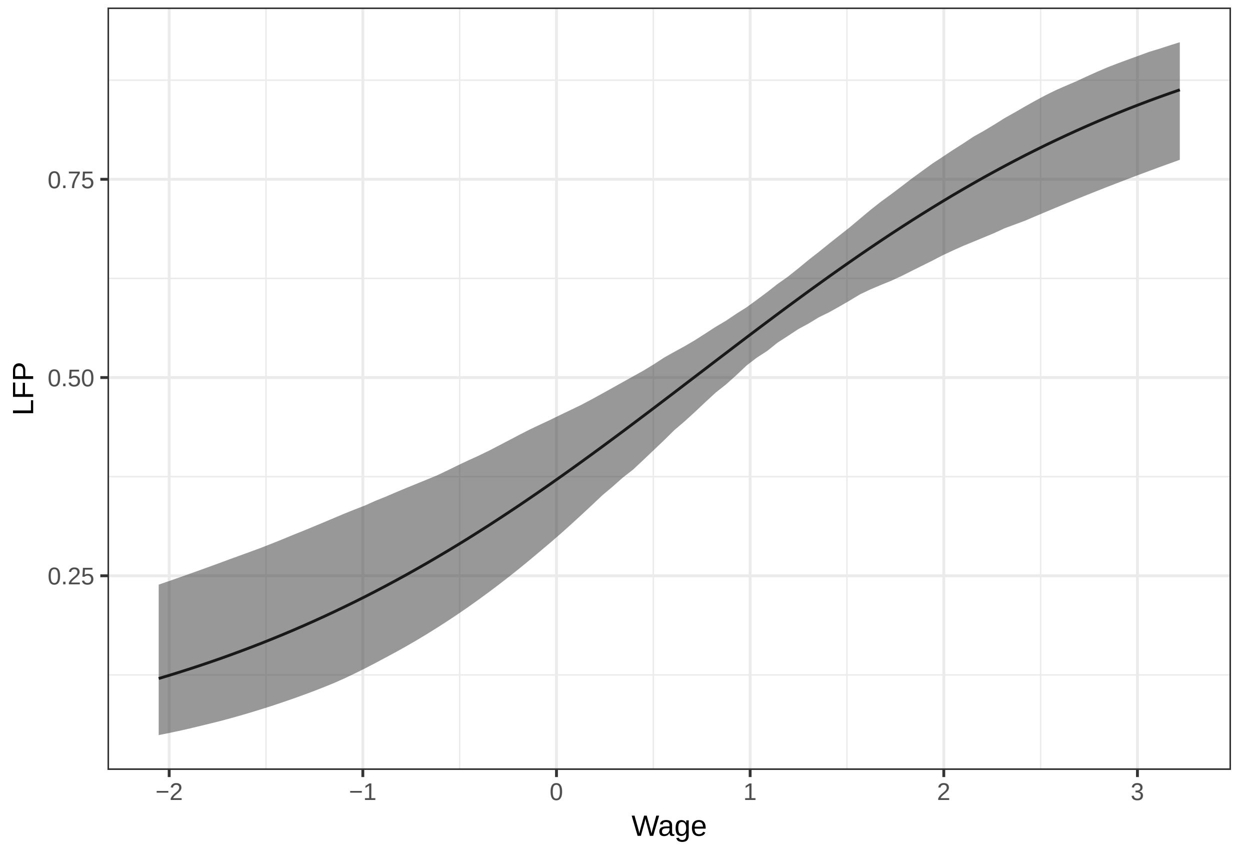
```
# Spaghetti plot
ggData <- melt(predUncert)
names(ggData) <- c('Scenario', 'Wage', 'LFP')
ggData$Wage <- rep(lwgRange, each=sims)
ggplot(ggData, aes(x=Wage, y=LFP, group=Scenario)) + geom_line(alpha=.1)
```



Another way we can do this is to just pull out the mean and 95% confidence intervals from our simulations.

```
ggData <- t(apply(predUncert, 2, function(x){
    mean = mean(x)
    qlo95 = quantile(x, 0.025)
    qhi95 = quantile(x, 0.975)
    rbind(mean, qlo95, qhi95) } ))
ggData <- data.frame(ggData)
colnames(ggData) <- c('LFP', 'Lo95', 'Hi95')
ggData$Wage <- lwgRange

ggplot(ggData, aes(x=Wage, y=LFP, ymin=Lo95, ymax=Hi95)) + geom_line() + geom_ribbon(alpha=.5)
```

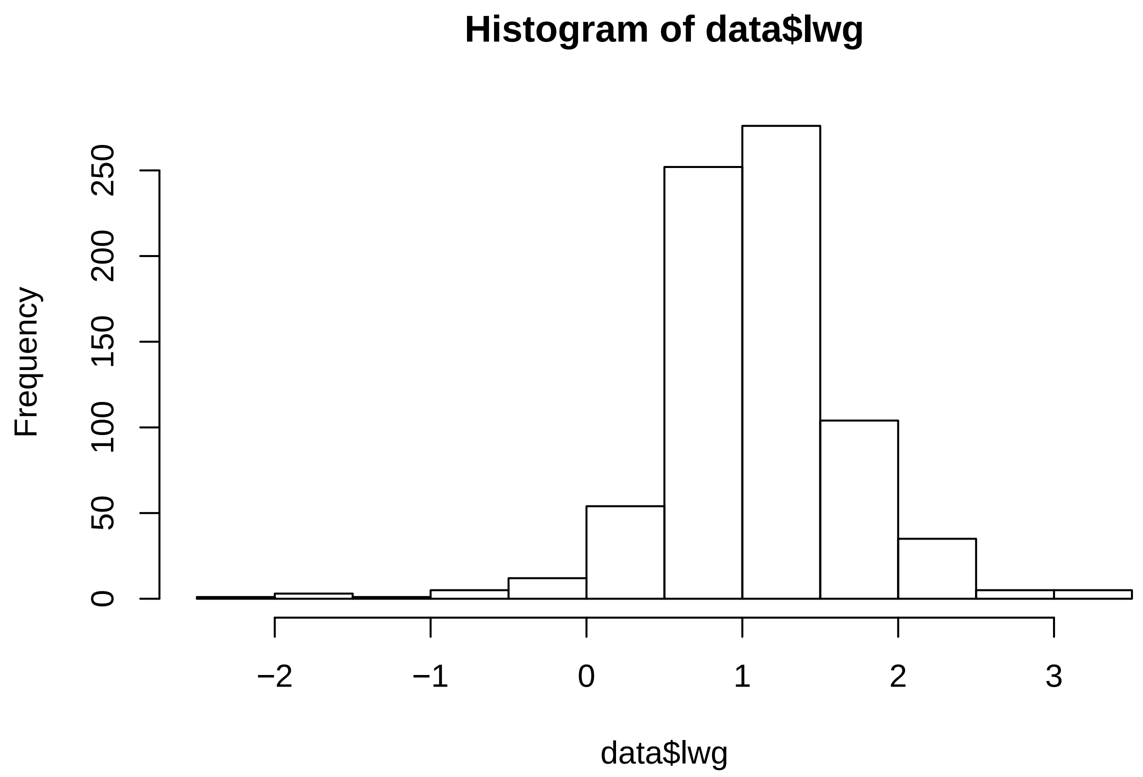


And just like that, we have an identical plot to the one we made with `R`'s canned functions. As we can see, there is a substantively important relationship between `lwg` and `flp`. Which we could also surmise from the fact that the odds ratio for `lwg` is 2.12, and comparing that to the distribution of `lwg`:

```
summary(data$lwg)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.0540  0.8181  1.0680  1.0970  1.4000  3.2190
```

```
hist(data$lwg)
```



While a one unit change (the required amoutn for a 2.12 times increase in the odds of labor force participation) would be rare, movement from the first to third quartile would increase the odds of labor force participation more than 50 percent. (How would you figure that out?)