This section, we'll look at outliers and Maximum Likelihood estimation.

The grammar of `ggplot2`

Detecting outliers

Maximum Likelihood

In this section, we're going to step away from the world of OLS for a little bit and explore a different family of estimators: the maximum likelihood estimator. We'll work through some examples empirically demonstrating how to conduct maximum likelihood estimation in `R`.

A brief reminder: the general principle behind ML estimation is that we have observe some data vector $\mathbf{z}$, and we assume that $\mathbf{z}$ has a probability distribution characterized by a parameter vector θ such that $f(\mathbf{z};\theta)$. In general, we pick a particular probability distribution (hopefully, but not always, based on sound economic theory) and we attempt to determine the $\hat{\theta}$ that best explains the data $\mathbf{z}$ that we observe. In orer to find $\hat{\theta}$, we simply search for the parameters that maximize the probability of observing the values of our data $\mathbf{z}$.

If you're like me, you found the above explanation mostly baffling the first few times you heard it. Rather than repeating it to you once more, let's dive into the estimation process with some specific examples.

ML on a Bernouilli distribution

The process of flipping a (potentially rigged) coin is described by the **Bernouilli distribution**, which is a special case of the binomial distribution. A Bernouilli random variable is 1 with probability p and 0 with probability $1-p$. Suppose that we observe a sequence of coin tosses that looks something like $\mathbf{x} = [0\ 1\ 0\ 0\ 0\ 0\ 1\ 1\ 0\ 1\ 1\ \ldots]$. We can use ML to determine what the most likely value for our single parameter p is. We'll use a simulation method again so that we can set the "true" data to test our method[1] . First we create our (very long) sequence of flips.

```
set.seed(42)
flips <- 1000
p.true <- 0.59
x <- rbinom(flips, 1, p.true)
head(x)
```

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

Let's first define our likelihood function, which is just the joint probability of observing the particular sequence of $\mathbf{x}$ that we see, given that each flip is identically and independently distributed Bernoulli.

[1] This section is based on a similar tutorial by John Myles White, available at: `http://www.johnmyleswhite.com/notebook/2010/04/21/doing-maximum-likelihood-estimation-by-hand-in-r/`.

$$\mathrm{L} = \prod_{i=1}^{1000} p^{x_i}(1-p)^{1-x_i}$$

We can also take logs of this to produce the often-more-tractable log-likelihood function:

$$\mathscr{L}_n(\theta) = \sum_{i=1}^{1000} x_i \ln(p) + (1-x_i)\ln(1-p)$$

By taking the derivative of the log likelihood function we can show[2] analytically that the best estimate of p is the sample mean of $\mathbf{x}$. This gives us a benchmark against which we can compare the estimates we get from optimization.

```
(analytical.est <- mean(x))
```

```
[1] 0.619
```

We can actually optimize over either the likelihood or the log-likelihood function. Let's create both of them as functions that take in `p`, our guess at the value for p, and `x`, our sequences of 0s and 1s. The output of both functions will be the likelihood (or log-likelihood) of observing `x` given our guess at `p`. Remember, our ultimate goal is to maximize these functions, so we want to find the `p` that makes them largest (i.e. the argmax_p).

```
likelihood <- function(p, x) {
  prod(p^x * (1-p)^(1-x))
}

log.likelihood <- function(p, x) {
  sum(x * log(p) + (1-x) * log(1-p))
}
```

Now, let's calculate the values of the these functions over the range $[0, 1]$.

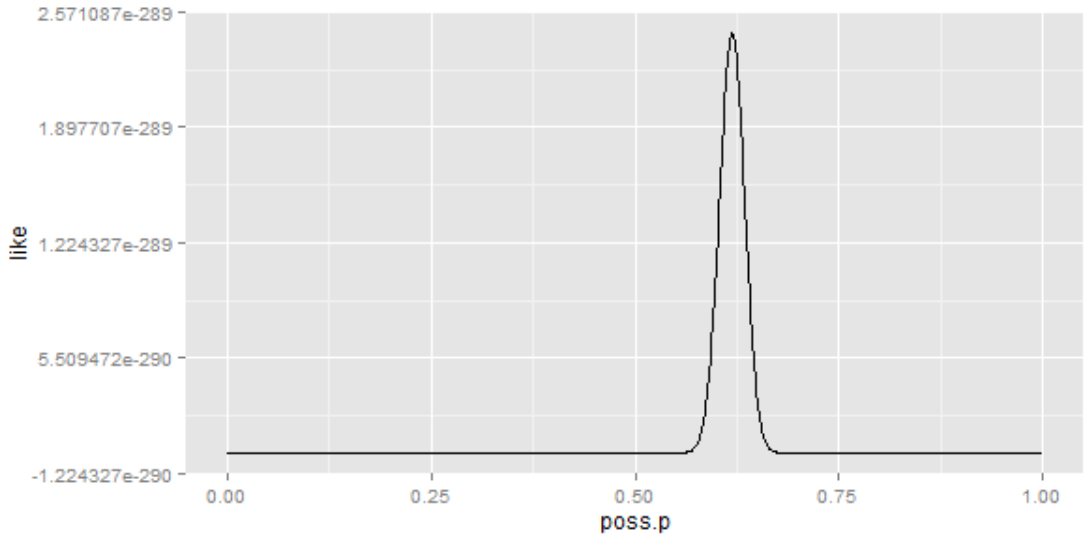
```
poss.p <- seq(0,1,0.001)
like <- sapply(poss.p, likelihood, x = x)
loglike <- sapply(poss.p, log.likelihood, x = x)
```

So what is it that we're trying to maximize, exactly? Let's find out. Using `ggplot2`, we plot the likelihood function:

Figure 1: Likelihood function

```
data <- data.frame(poss.p,like,loglike)
library(ggplot2)
ggplot(data=data, aes(x=poss.p, y=like))  + geom_line()
```

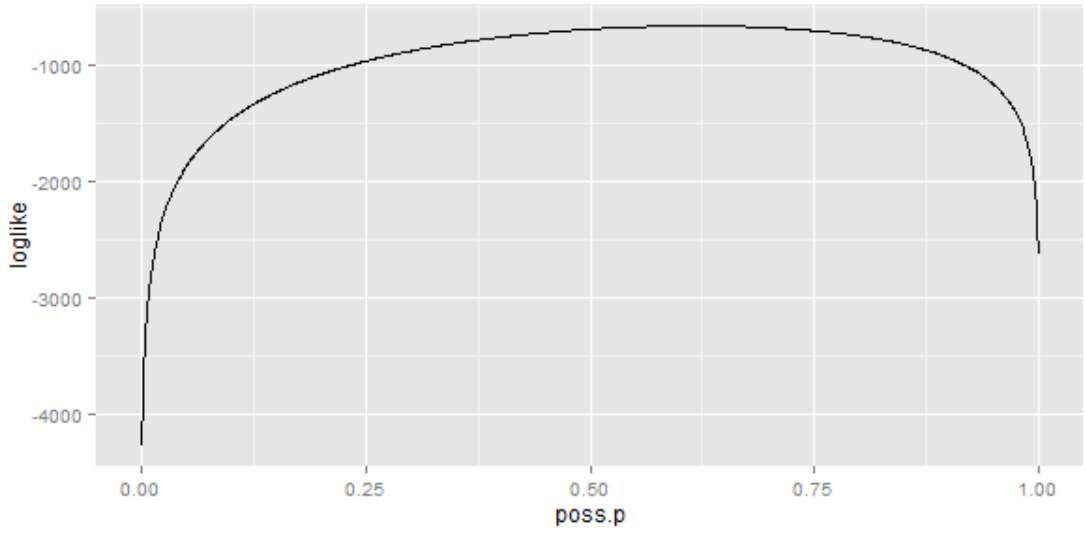
[2] But won't. See `http://mathworld.wolfram.com/MaximumLikelihood.html`.



As you can see, the, function seems to peak sharply around $p = 0.60$. That's pretty good! Since log is a monotone operator[3], we should expect to see the maximum at the same point p:

Figure 2: Log-likelihood function

```
ggplot(data=data, aes(x=poss.p, y=loglike)) + geom_line()
```



And we do! More or less. However, if we want to be more precise about our estimate, we ought to use an optimization algorithm. Let's try it, first with the likelihood function.

```
opt.like <- optimize(f = likelihood, c(0,1), maximum = T, x = x)
cbind(opt.like$maximum, opt.like$objective)
```

```
          [,1]          [,2]
[1,] 0.6189997 2.448654e-289
```

[3] I.E. $x > y \leftrightarrow \ln(x) > \ln(y)$

That's pretty darn close to our analytical estimate, the sample mean, which was `0.619`. How about the log-likelihood?

```
opt.loglike <- optimize(f = log.likelihood, c(0,1), maximum = T, x = x)
cbind(opt.loglike$maximum, opt.loglike$objective)
```

```
          [,1]      [,2]
[1,] 0.6190052 -664.5516
```

Cool. You'll recall from lecture that we use the log-likelihood because it provides analytical convenience — it's typically much simpler to take the derivative over a sum than a product. But you might not know that the log-likelihood has computational advantages as well! To see this, try setting `flips = 100000`. Cover your ears.

Fitting a model using maximum likelihood

Now that we've got a little maximum likelihood experience under our belts, let's see if we can replicate the theoretical results from lecture by fitting a regression model using ML rather than OLS. To use ML, we need to make some sort of distributional assumption that we can use to estimate our parameters. Following the lecture notes (Section 2.7.3), we use the convenient assumption A6, which gives us that $\varepsilon|\mathbf{X} \sim \mathbf{N}(\mathbf{0}, \sigma^{\mathbf{2}}\mathbf{I_n})$. From the linearity of our model, we get that $\mathbf{y}|\mathbf{X} \sim (\mathbf{X}\beta, \sigma^2\mathbf{I}_n)$, which we plug into the pdf for a multivariate normal distribution. Taking logs over the distribution gives us:

$$\log L(\tilde{\mathbf{b}}, \tilde{\sigma}^2) = -\frac{n}{2}\log(2\pi) - \frac{n}{2}\log(\tilde{\sigma}^2) - \frac{1}{2\tilde{\sigma}^2}(\mathbf{y} - \mathbf{X}\mathbf{b})'(\mathbf{y} - \mathbf{X}\mathbf{b})$$

Following Max's notes and solve for $\mathbf{b}$ and σ^2 analytically, we should find that $\mathbf{b}_{ML} = \mathbf{b}_{OLS} = (X'X)^{-1}X'y$ and that $\sigma^2 = \frac{\mathbf{e}'\mathbf{e}}{n}$. **Or**, we could think jointly estimate $\mathbf{b}$ and σ^2 using the `optim` command (which is similar to `optimize`, but can optimize with more than one parameter).

```
n <- 2000
set.seed(42)
X <- cbind(1,runif(n))
eps <- rnorm(n)
b.true <- rbind(5,3)
y <- X %*% b.true + eps
```

```
log.likelihood.optim <- function(theta) {
  sigma2 <- tail(theta, n = 1)
  b <- theta[1:nrow(b)]
  e <- y - X %*% b
  output <- -n/2*log(2*pi) - n/2*log(sigma2) - 1/(2 * sigma2) * t(e) %*% e
  return(-output)
}
optim(par = c(3,2,1), fn = log.likelihood.optim)$par
```

```
[1] 4.988389 3.014805 1.015007
```

The first argument to `optim` is a set of starting parameters for θ. In general, in order to maximize our chances of finding the correct optimum (and to increase the rate of convergence), we want to choose reasonable parameters. The second argument is just the log-likelihood function that we created. We should observe that, in line with theory, $\$\mathbf{b}_{\mathrm{ML}} \approx b_{OLS}$. Let's check.

```
OLS <- function(y,X) { b <- solve(t(X) %*% X) %*% t(X) %*% y }
(b <- OLS(y,X))
```

```
         [,1]
[1,] 4.988142
[2,] 3.015419
```

Whew! But hey — that's pretty neat.