This week we'll talk about computing the standard errors of linear and non-linear functions of coefficients using both analytical calculation and the delta method. We'll begin by discussing how to compute the standard error of a sum of two coefficients, followed by the more complex standard error of the prediction. Finally, we'll show how to compute a nonlinear function of coefficients using a quadratic data example.

Standard error of a linear combination of coefficients

Let's return to section 5 to think about wages. Suppose we run the following model:

$$\text{wage}_i = \alpha + \beta_1 \text{EDUC}_i + \beta_2 \text{TENURE}_i + \varepsilon_i$$

We can run this model in `R` as follows (after, of course, setting up our OLS function):

```
OLS <- function(y,X) {
  n <- nrow(X); k <- ncol(X)
  b <- solve(t(X) %*% X) %*% t(X) %*% y
  e <- y - X %*% b; s2 <- t(e) %*% e / (n - k); XpXinv <- solve(t(X) %*% X)
  se <- sqrt(s2 * diag(XpXinv))
  t <- b / se
  p <- 2 * pt(-abs(t),n-k)
  output <- data.frame(b, se, t, p)
  colnames(output) <- c("Estimate","Std. Error", "t statistic", "p-value")
  return(output)
}

library(foreign)
library(xtable)
#data <- read.dta("http://fmwww.bc.edu/ec-p/data/wooldridge/wage2.dta")
data <- read.dta("wage2.dta")
data <- data[ , c("wage", "educ", "tenure")]
data <- na.omit(data)
y <- data$wage
X <- cbind(1,data$educ,data$tenure)
OLS.out <- OLS(y,X)
xtable(OLS.out)
```

	Estimate	Std. Error	t statistic	p-value
1	53.52	79.56	0.67	0.50
2	61.15	5.64	10.84	0.00
3	11.18	2.44	4.58	0.00

Great!

But maybe we don't really care about the coefficients. Maybe what we *really* want to estimate is the average additional wage benefit[1] of three years of education and two years at a company. Let's

[1] Here I ask you to make the appropriate suspension of disbelief w.r.t the causal nature of this model.

call it $d = 3\beta_1 + 2\beta_2$. We can estimate the mean of d using our regression results using the estimates b_1 and b_2.

```
(d <- 3 * OLS.out[2,1] + 2 * OLS.out[3,1])
```

```
[1] 205.7979
```

As an economist, though, we should be suspicious of any estimates presented without standard errors[2]. But how do we compute them?

If we were using `Stata`, the immediate answer is `lincom`. And in fact, there are canned functions in `R` that can do the same thing. But it's more interesting to do it by hand, and it gives us the opportunity to demonstrate two different ways of computing standard errors.

Analytical method

The first way to get the standard errors is by using the analytical method. In this setting, it's fairly straightforward to compute the variance of d using the properties of the variance formula:

$$V(d) = V(3b_1 + 2b_2)$$
$$V(d) = 9V(b_1) + 4V(b_2) + (3 \times 2)\ \mathrm{Cov}(b_1, b_2)$$

We can get estimates for $V(b_1)$, $V(b_2)$, and $\mathrm{Cov}(b_1, b_2)$ from the variance-covariance matrix of the coefficients: $V(\mathbf{b}) = s^2(\mathbf{X}'\mathbf{X})^{-1}$. They will be the entries in (2,2), (3,3), and (2,3), respectively. To make this easier, we'll write a function `get.vcov` that returns this matrix.

```
get.vcov <- function(y, X)  {
  n <- nrow(X); k <- ncol(X)
  b <- solve(t(X) %*% X) %*% t(X) %*% y
  e <- y - X %*% b
  s2 <- as.vector(t(e) %*% e / (n - k))
  XpXinv <- solve(t(X) %*% X)
  vcov <- s2 * XpXinv
  return(vcov)
}
vcov <- get.vcov(y,X)
var.b1 <- vcov[2,2]
var.b2 <- vcov[3,3]
cov.b1b2 <- vcov[2,3]
(d.se <- sqrt(9 * var.b1 + 4 * var.b2 + 3*2*cov.b1b2))
```

```
[1] 17.69076
```

Great! We now know that our estimate of d is 205.8 with a standard error of 17.7. We can verify this in `R` using the `estimable()` command in the `gmodels` package (you may have to install it).

[2] This happens all the time, everywhere. So what I'm really telling you is that you should be suspicious all the time, everywhere.

```
library(gmodels)
lm <- lm(data$wage ~ data$educ + data$tenure)
hm <- c(0,3,2)
estimable(lm, hm)
```

```
        Estimate Std. Error  t value  DF Pr(>|t|)
(0 3 2) 205.7979   17.77496 11.57797 932        0
```

Delta method

The delta method, introduced in lecture last week, is a convenient way to approximate standard errors for a wide variety of functions of the coefficients[3]. The notation can be confusing at first, but hopefully its use will become clear once we do a few examples. First, a brief review of the theorem that drives the method:

Let $\mathbf{A}(\boldsymbol{\beta}) \equiv \frac{\partial \mathbf{a}(\boldsymbol{\beta})}{\partial \boldsymbol{\beta}'}$. From Max's notes, we know the following:

$$\sqrt{N}(\mathbf{x}_N - \boldsymbol{\beta}) \xrightarrow{d} N(\mathbf{0}, \boldsymbol{\Sigma}) \Rightarrow \sqrt{\mathbf{N}}(\mathbf{a}(\mathbf{x_N}) - \mathbf{a}(\boldsymbol{\beta})) \xrightarrow{\mathbf{d}} \mathbf{N}(\mathbf{0}, \mathbf{A}(\boldsymbol{\beta})\boldsymbol{\Sigma}\mathbf{A}(\boldsymbol{\beta})')$$

This looks intimidating, but it's actually fairly straightforward to bend it to our setting. Let $\mathbf{x}_N \equiv \mathbf{b}$ and $\mathbf{a}(\mathbf{x}_N) \equiv d(\boldsymbol{\beta})$. This gives us that $\mathbf{A}(\beta) \equiv \mathbf{D}(\boldsymbol{\beta}) = [0\ \ 3\ \ 2]$.

We know from the proof in section 4.2 of the lecture notes that $\sqrt{N}(\mathbf{b} - \boldsymbol{\beta}) \xrightarrow{d} N(\mathbf{0}, \boldsymbol{\sigma^2}(\mathbf{X}'\mathbf{X})^{\mathbf{-1}})$, so we immediately get that $V(d) \approx \mathbf{D}\sigma^2(\mathbf{X}'\mathbf{X})^{-1}\mathbf{D}'$. That's it! That's the whole method.
In sum, to approximate the variance of our linear combination of coefficients, d, all we have to do is pre- and post-multiply $\sigma^2(\mathbf{X}'\mathbf{X})^{-1}$ by the gradient of d, $\mathbf{D}$. As usual, the standard error is just the square root of the variance. Putting it all together gives us our answer:

```
D <- matrix(c(0,3,2), ncol = 3)
(sqrt(D %*% vcov %*% t(D)))
```

```
         [,1]
[1,] 17.77496
```

Wow! It works! I'm as surprised as you are.

Computing the standard error of the prediction

The standard error the prediction is an important quantity for econometricians who are interested in forecasting. It answers the question "how much variation should I expect to see when I use the coefficients from OLS to predict other values of y?" Modern statistical software has built-routines to calculate the prediction and the standard error of the prediction, but it's useful to know the calculation is performed. In fact, since our model is linear, the prediction is just a linear combination of the coefficients and the error term!
Our model is

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \boldsymbol{\varepsilon}$$

[3]It's actually useful in a broad variety of settings, but we use it almost exclusively for standard errors in econometrics.

We'll demonstrate using the `iris` dataset built into `R` and attempt to predict sepal length from sepal width and petal width[4]. First, the predictions:

```
iris.df <- iris
y <- iris.df$Sepal.Length
X <- cbind(1, iris.df$Sepal.Width, iris.df$Petal.Width)
b <- OLS(y,X)[ ,1]
head(pred.y <- X %*% b)
```

```
         [,1]
[1,] 5.048507
[2,] 4.848972
[3,] 4.928786
[4,] 4.888879
[5,] 5.088414
[6,] 5.402561
```

Getting the predictions for our data $\mathbf{X}$ is the easy part. Now we want to compute $V(e^0) = V(\hat{y}^0 - y^0)$, the variance of the error in prediction for a particular set of covariates $\mathbf{x}^0$.

Analytical

First, we can calculate the variance analytically. This is similar in spirit to the analytical exercise above, but since we are working with more complex objects (coefficients, data, and disturbances), we'll use matrix notation. Still, everything that follows is just algebra and the properties of the variance function.

$$\begin{aligned}
V(\hat{y}^0 - y^0) &= V(\mathbf{X}^0\mathbf{b} - \mathbf{X}^0\boldsymbol{\beta} - \boldsymbol{\varepsilon}^0) \\
&= \sigma^2 + V(\mathbf{X}^0(\mathbf{b} - \boldsymbol{\beta})) \\
&= \sigma^2 + V(\mathbf{X}^0\mathbf{b}) \\
&= \sigma^2 + \mathbf{X}^0 V(\mathbf{b})\mathbf{X}'^0 \\
&= \sigma^2 + \mathbf{X}^0\sigma^2(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'^0 \\
&= \sigma^2(1 + \mathbf{X}^0(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'^0)
\end{aligned}$$

Of course, using the y^0 and $\mathbf{X}^0$ is only to make the point that we are actually computing the standard error of the prediction for a single set of data $\mathbf{X}^0$. Practically, we can tell `R` to compute the standard errors for all of observations all at once. We'll do so using the formula we derived above:

```
e <- y - X %*% b
n <- nrow(X); k <- ncol(X)
s2 <- as.vector(t(e) %*% e / (n - k))
XpXinv <- solve(t(X) %*% X)
pred.se <- diag(sqrt(s2 * (1 + X %*% XpXinv %*% t(X))))
head(pred.se)
```

```
[1] 0.4556677 0.4558185 0.4552114 0.4554239 0.4561838 0.4583894
```

To verify, we can use `lm()` and `predict()`:

[4] Dammit Jim, I'm a doctor, not a botanist!

```
lm <- lm(Sepal.Length ~ Sepal.Width + Petal.Width, data = iris.df)
predict.out <- predict(lm, se.fit = T)
all.equal(as.vector(predict.out$fit),as.vector(pred.y))
all.equal(as.vector(predict.out$se.fit),as.vector(pred.se))
```

```
[1] TRUE
[1] "Mean relative difference: 6.394198"
```

Something's not right. Let's investigate.

```
head(predict.out$se.fit)
head(pred.se)
```

```
[1] 0.06446854 0.06552559 0.06116029 0.06272227 0.06802035 0.08151061
[1] 0.4556677 0.4558185 0.4552114 0.4554239 0.4561838 0.4583894
```

Hm. That looks fishy. I'll spare you the suspense — `R` apparently has a different definition of the standard error of the prediction, and is returning the square root of $V(\hat{y}^0)$, rather than the square root of $V(\hat{y}^0 - y^0)$. This is why you should be careful with canned results! Anyway — we can replicate what they have by dropping the 1 from our equation:

```
pred.se.2 <- sqrt(diag(s2 * (X %*% XpXinv %*% t(X))))
head(pred.se.2)
all.equal(as.vector(predict.out$se.fit),as.vector(pred.se.2))
```

```
[1] 0.06446854 0.06552559 0.06116029 0.06272227 0.06802035 0.08151061
[1] TRUE
```

Whew. Mystery solved. Note that `predict()` *does* know how to return the correct 95% prediction interval, and will do so if we set interval = "prediction". If you think this is confusing, you are not alone.

Delta method

To use the delta method, we'll use that $V(\hat{y}^0 - y^0) = \sigma^2 + V(\mathbf{X}^0\mathbf{b})$. Now we'll use the delta method to compute V($\mathbf{X}^0$ $\mathbf{b}$). Let $\mathbf{a} \equiv X^0\mathbf{b}$ and $\mathbf{A} \equiv X^0$. This gives us the following familiar result:

$$V(\hat{y}^0 - y^0) = \sigma^2 + \mathbf{X}^0\sigma^2(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'^0 = \sigma^2(1 + \mathbf{X}^0(\mathbf{X}'\mathbf{X})^{-1}\mathbf{X}'^0)$$

In this case, the "approximation" provided by the delta method is really the true variance that we calculated earlier using our analytical method! This is because the delta method uses a first-order Taylor approximation, which is only an approximation when our function $\mathbf{a}$ has second-order terms. So there's not much else to do here — the analytical result and the delta method are more or less the same.

Non-linear functions of coefficients

So far, we've only computed linear functions of coefficients. But suppose we're interested in some *non-linear* function of the coefficients. How do we compute the standard errors? Let's take an example[5].

Suppose we are running a quadratic regression model:

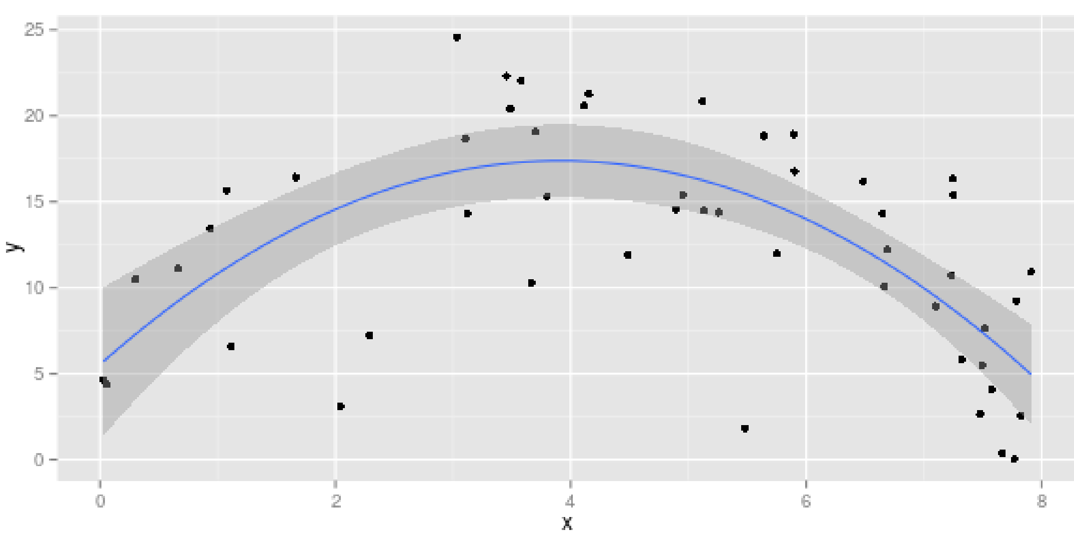
$$y_i = \alpha + \beta_1 x_i + \beta_2 x_i^2 + \varepsilon_i$$

First, we'll simulate some data.

```
set.seed(42)
n <- 50
x <- runif(n,0,8)
x2 <- x^2
eps <- rnorm(n,0,5)
X <- cbind(1, x, x2)
beta <- c(3, 8, -1)
y <- X %*% beta + eps
b <- OLS(y,X)[ ,1]
```

We'll graph this using a quadratic fit line. This is cheating, since normally we'd look at the scatterplot of our data, see the quadratic trend, and *then* fit the line, but you get the idea.

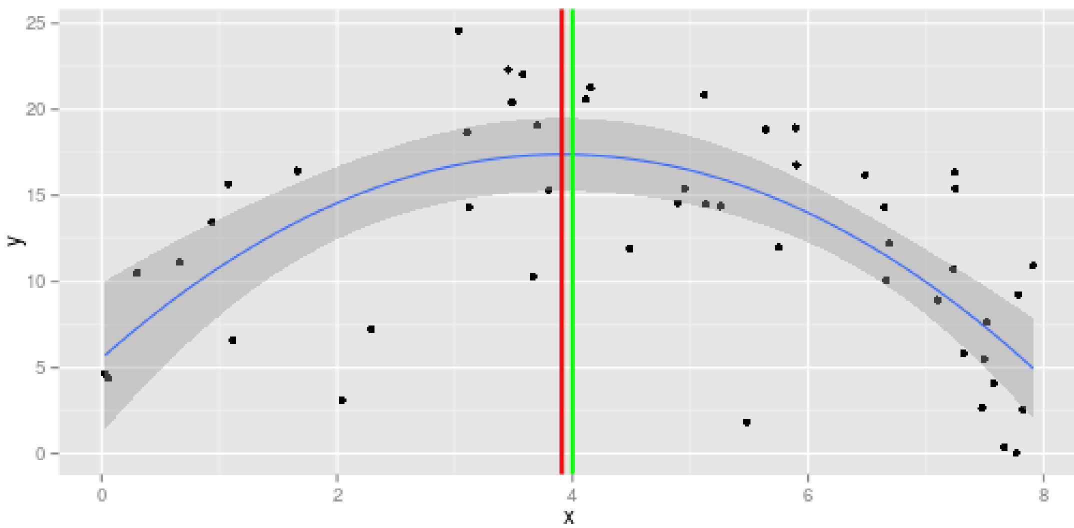
```
library(ggplot2)
curve.df <- data.frame(y,x)
(g <- ggplot(data = curve.df, aes(x=x, y=y)) + geom_point()
      + geom_smooth(method="lm", formula = y ~ x + I(x^2)))
```



This is great. We specified a quadratic model in our simulation and a quadratic line seems to fit! But what we *really* want to know is the value of x for which the maximum of the line is reached. Computing this for a regression is easy enough: the maximum value will always occur at $x_{max} = \frac{-\beta_1}{2\beta_2}$. Let's plot this again, but this time we'll draw two vertical lines: one that shows the true x that gives the maximum value (spoiler alert: it's $x_{max} = 4$) and one that shows the one we can compute using the formula $x_{max} = \frac{-b_1}{2*b_2}$.

[5] Credit for this example goes to Dason Kurkiewicz, whose post at `http://dasonk.github.io/r/2013/02/09/Using-the-delta-method/` inspired this section.

```
true.xmax <- -beta[2] / (2 * beta[3])
est.xmax <- -b[2] / (2 * b[3])
(g<- g + geom_vline(xintercept = true.xmax, colour = "green", size=1)
   + geom_vline(xintercept = est.xmax, colour = "red", size=1))
```



Due to the noise of the data, we haven't quite approximated the true maximum (although this is actually quite good for 50 observations with plenty of variance!). But what about the standard errors?

Analytically

It is, in principle, possible to compute the standard errors of $x_{max} = -\frac{b_1}{2b_2}$ analytically. We can think of the components of our coefficient vector $\mathbf{b}$ as random variables themselves (otherwise how would we be getting their variances), so all $V(x_{max})$ is just the variance of the ratio of two random variables. But this is where I get off the train — the formula is totally horrible[6]. No more will be said of it.

Delta method

Forget using the delta method for sums of variables. Where it *really* shines is in enabling lazy econometricians like myself easily calculate the standard errors of products and quotients (or worse!) of coefficients. As usual, we let $\mathbf{a}(\beta) = x_{max}(\beta_1, \beta_2) = -\frac{\beta_1}{2\beta_2}$, which gives that $\mathbf{A} = [-\frac{1}{\beta_2} \;\; \frac{\beta_1}{\beta_2^2}]$. This gives us our delta method approximation of $V(x_{max}) = \mathbf{A}V(\beta)\mathbf{A}' = \sigma^2\mathbf{A}(\mathbf{X}'\mathbf{X})^{-1}\mathbf{A}'$. As usual, we'll replace the population parameters with their sample equivalents.
Computationally, this is straightforward:

```
b1 <- b[2]; b2 <- b[3]
A <- matrix(c(0, -1 / (2 * b2), b1 / (2* b2^2)), nrow = 1)
vcov <- get.vcov(y,X)
(se.xmax <- sqrt(A %*% vcov %*% t(A)))
```

[6] You really want to know? See "ON THE ARITHMETIC MEANS AND VARIANCES OF PRODUCTS AND RATIOS OF RANDOM VARIABLES" by Fred Fishman, 1971.

```
          [,1]
[1,] 0.191088
```

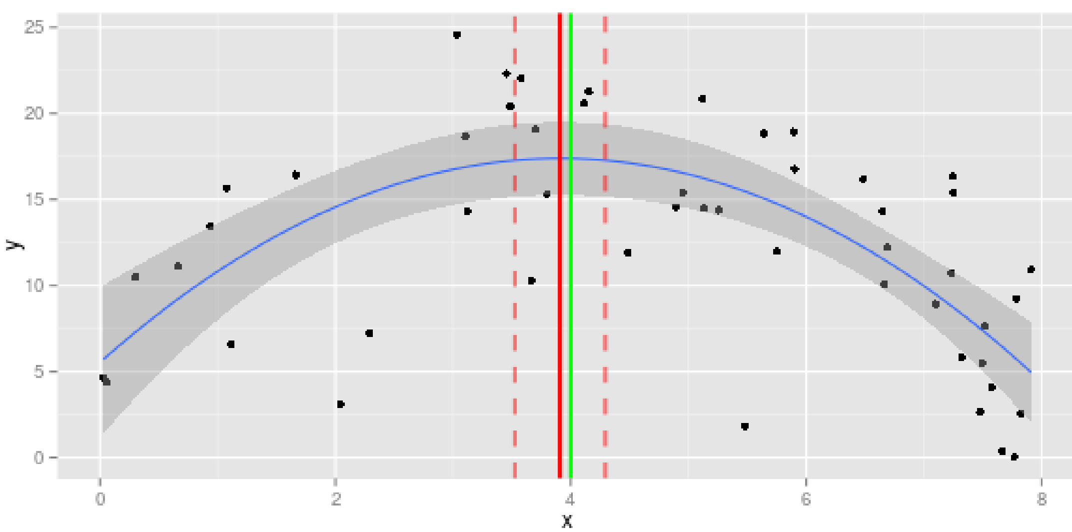
Just to make sure we're doing this right, we'll compute the standard errors using the `deltamethod` function from the `msm` package (which you may need to install):

```
o <- lm(y ~ x + I(x^2))
library(msm)
(standerr <- deltamethod(~-x2/(2 * x3), coef(o), vcov(o)))
```

```
[1] 0.191088
```

Kablammo! For the coup de grace, we'll graph this again, this time with the 95% confidence intervals around our estimate. This is a good time to marvel at how nice it is to progressively build graphs in `ggplot2`.

```
ci <- est.xmax + c(-1, 1) * qt(0.975, n-k) * se.xmax
(g + geom_vline(xintercept = ci[1], colour = "red", size=1, linetype="dashed", alpha=0.5)
   + geom_vline(xintercept = ci[2], colour = "red", size=1, linetype="dashed", alpha=0.5))
```



KAPOWEE! The 95% confidence interval contains the true estimate! And we're done.