Multivariate delta method example

Sean Anderson

Here we will use the delta method to calculate a standard error on a derived quantity involving 2 parameters:

Imagine we have a bunch of fish lengths:

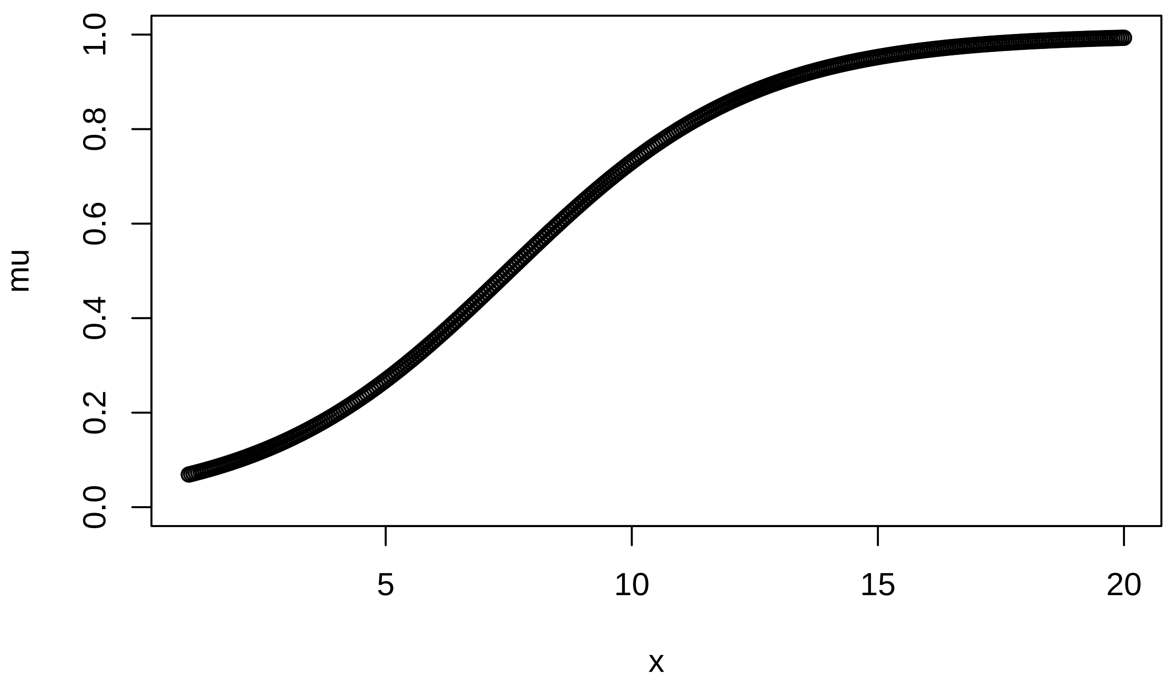
```
x <- seq(1, 20, length.out = 400)
```

And a true intercept and slope parameter from which we will simulate:

```
b0 <- -3
b1 <- 0.4
```

Form the linear predictor:

```
mu <- plogis(b0 + b1 * x)
plot(x, mu, ylim = c(0, 1))
```

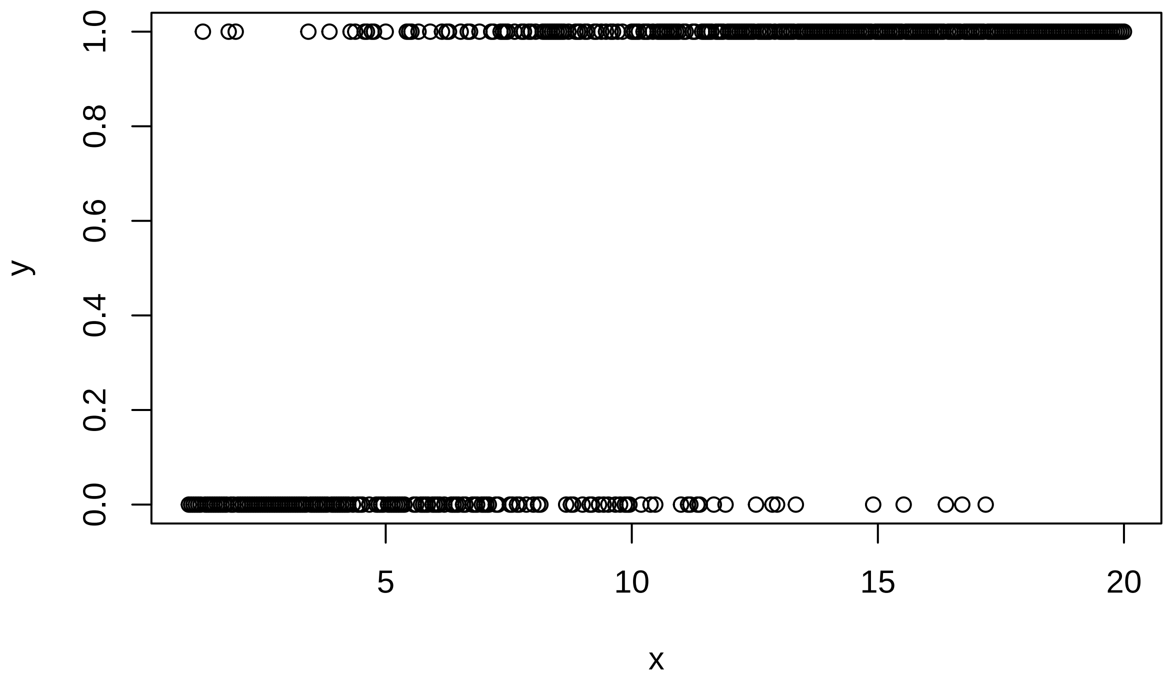


Apply observation error:

```
set.seed(1)
y <- rbinom(length(x), size = 1, prob = mu)
```

Plot it:

```
plot(x, y)
```



Fit a GLM to it:

```
fit <- glm(y ~ x, family = binomial())
coef(fit)
```

```
## (Intercept)           x
##  -3.0404584   0.3861907
```

```
est <- predict(fit) # linear predictor
```

What's the length at 50% maturity? We can calculate that as:

```
calc_length_p50 <- function(x1, x2, prob_threshold = 0.5) {
  -(log((1/prob_threshold) - 1) + x1) / x2
}
(b0_hat <- coef(fit)[[1]])
```

```
## [1] -3.040458
```

```
(b1_hat <- coef(fit)[[2]])
```

```
## [1] 0.3861907
```

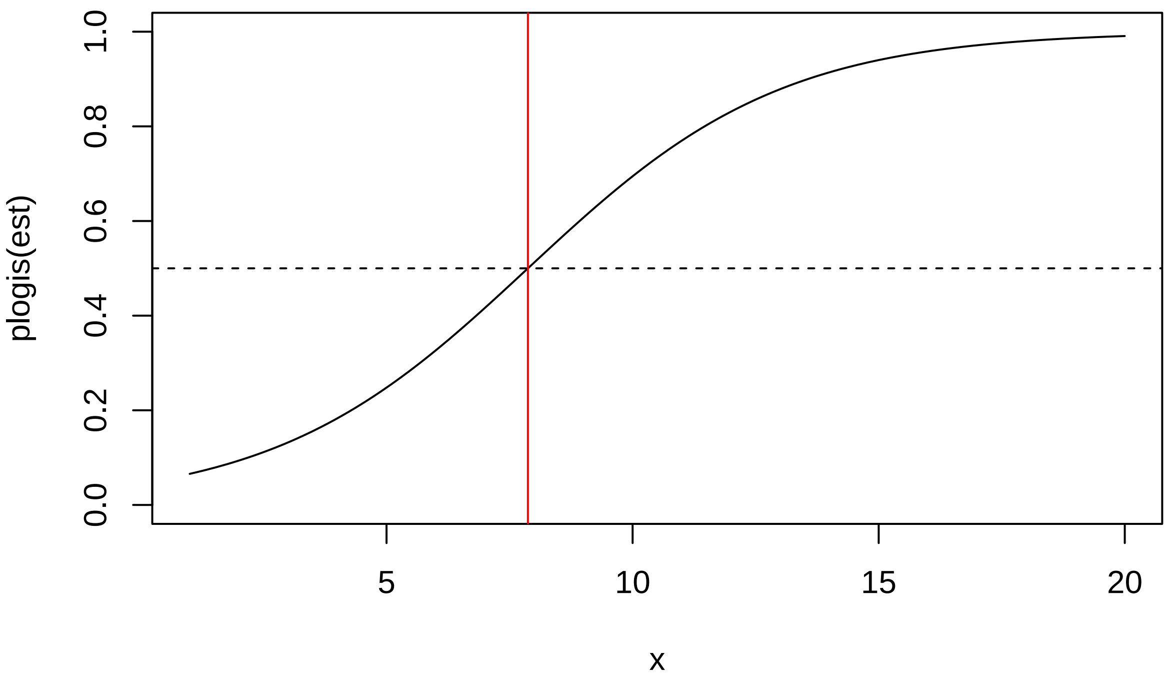
```
p50 <- calc_length_p50(b0_hat, b1_hat)
p50
```

```
## [1] 7.872946
```

So this transformation above is our 'g' function in the delta method.

Plot it:

```
plot(x, plogis(est), ylim = c(0, 1), type = "l")
abline(v = p50, col = "red")
abline(h = 0.5, lty = 2)
```



OK, but what about the standard error on p50? We can use the multivariate delta method to get that.

```r
# Grab our covariance matrix:
cov <- vcov(fit)

# Define our transformation of interest:
g <- ~ -(log((1/0.5) - 1) + b0_hat) / b1_hat

# Create a function that calculates the partial derivatives:
deriv_function <- deriv(g, c("b0_hat", "b1_hat"))
deriv_function
```

```
## expression({
##     .expr4 <- log(1/0.5 - 1) + b0_hat
##     .value <- -.expr4/b1_hat
##     .grad <- array(0, c(length(.value), 2L), list(NULL, c("b0_hat",
##         "b1_hat")))
##     .grad[, "b0_hat"] <- -(1/b1_hat)
##     .grad[, "b1_hat"] <- .expr4/b1_hat^2
##     attr(.value, "gradient") <- .grad
##     .value
## })
```

```r
# Evaluate that function:
e <- eval(deriv_function)

# Note that this gives us the same thing as we did above to
# calculate p50, but it also gives us the partial derivatives:
e
```

```
## [1] 7.872946
## attr(,"gradient")
##         b0_hat    b1_hat
## [1,] -2.589395 -20.38616
```

```r
# We can grab those partial derivatives:
J <- attr(e, "gradient")
```

These are also known as the Jacobian: a matrix of partial derivatives of the entries in g(theta) with respect to the entries in theta itself (theta is our parameter vector).

We then matrix multiply the Jacobian with the original covariance matrix by the transposed Jacobian to get the variance of the derived quantity:

```r
variance <- J  %*% cov %*% t(J )
variance
```

```
##           [,1]
## [1,] 0.1357803
```

The standard error is the square root of that:

```r
se <- sqrt(diag(variance))
se
```

```
## [1] 0.3684837
```

Plot it:

```r
plot(x, plogis(est), ylim = c(0, 1), type = "l")
abline(v = p50, col = "red")
abline(h = 0.5, lty = 2)
abline(v = p50 + 1.96 * se, col = "blue")
abline(v = p50 - 1.96 * se, col = "blue")
```