Stat 343: Numeric optimization

Problem set up

A recent study of 2700 children randomly selected from all parts of England found that 540 of them were deficient in vitamin D. Let's use these data to estimate the proportion of children in England who are deficient in vitamin D.

Let X = the count of children in our sample who are deficient in vitamin D.

We could use the model

$X \sim \text{Binomial}(2700, p)$

Reminder of previous results

We have previously seen the following results for this model, which you will use below:

The likelihood function is

$L(p|x) = \binom{n}{x} p^x (1-p)^{n-x}$

The log-likelihood function is

$\ell(p|x) = \log\left\{\binom{n}{x}\right\} + x\log(p) + (n-x)\log(1-p)$

The derivative of the log-likelihood with respect to p is

$\frac{d}{dp}\ell(p|x) = \frac{x}{p} + (n-x)\frac{1}{1-p}(-1)$

Setting this equal to 0 and solving for p, we obtain a critical point of

$p = \frac{x}{n}$

The second derivative of the log-likelihood is

$\frac{d^2}{dp^2}\ell(p|x) = \frac{-x}{p^2} + \frac{x-n}{(1-p)^2}$

Since both p^2 and $(1-p)^2$ are positive, and both $-x and $x - n$ are negative (or if one is 0 then the other is negative), the second derivative of the log-likelihood is negative. Thus, the critical point found above is the maximum likelihood estimate.

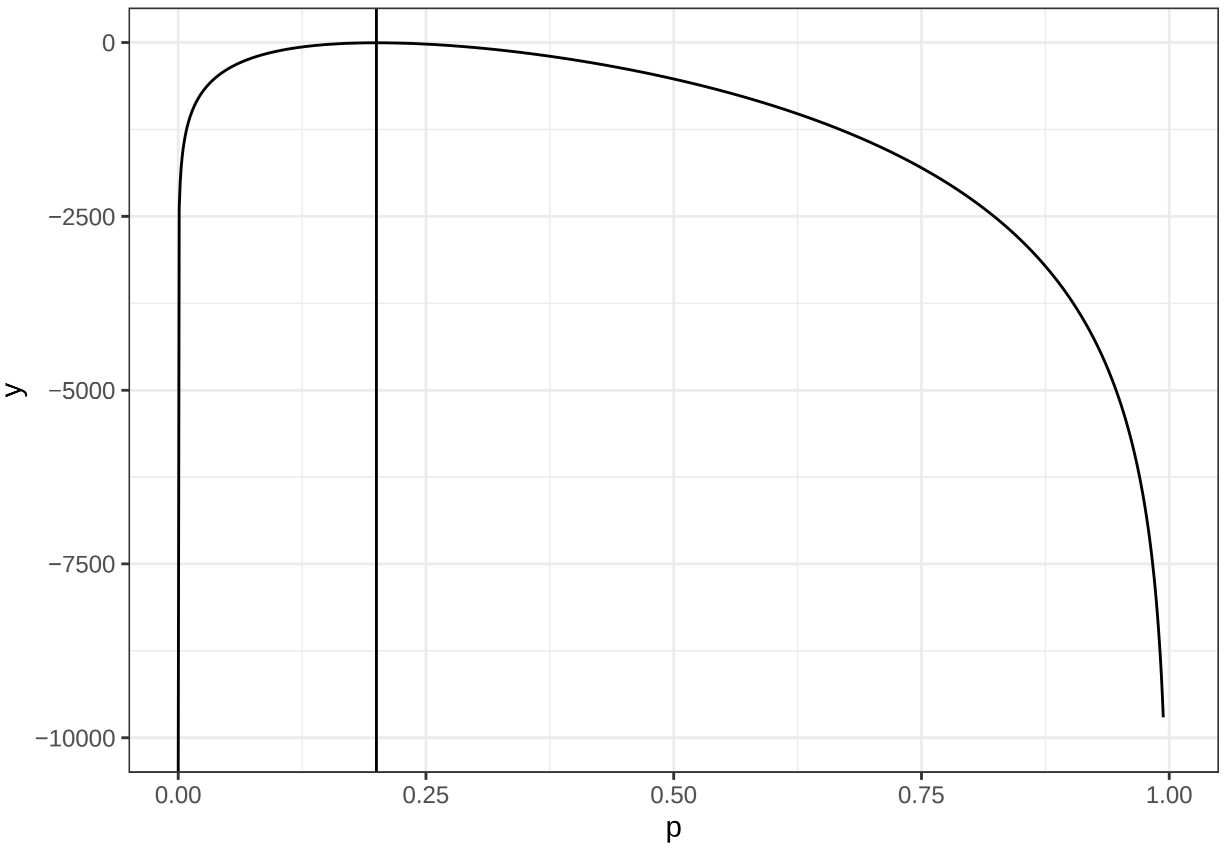
A plot of the log-likelihood and maximum likelihood estimate

The plot below displays the log-likelihood function, with a vertical line at the maximum likelihood estimate of $540/2700 = 0.2$.

```r
loglikelihood <- function(p) {
  result <- vector("numeric", length(p))
  for(i in seq_along(p)) {
    result[i] <- dbinom(540, size = 2700, prob = p[i], log = TRUE)
  }

  return(result)
}

ggplot(data = data.frame(p = c(0, 1)), mapping = aes(x = p)) +
  stat_function(fun = loglikelihood, n = 1001) +
  geom_vline(xintercept = 0.2) +
  ylim(c(-10000, 0)) +
  theme_bw()
```



Estimating p by numeric optimization

Let's see how we could use numeric optimization to estimate *p*. Since we know the maximum likelihood estimate is 0.2, we will know the method is working if our estimates converge to 0.2.

Display initial guess

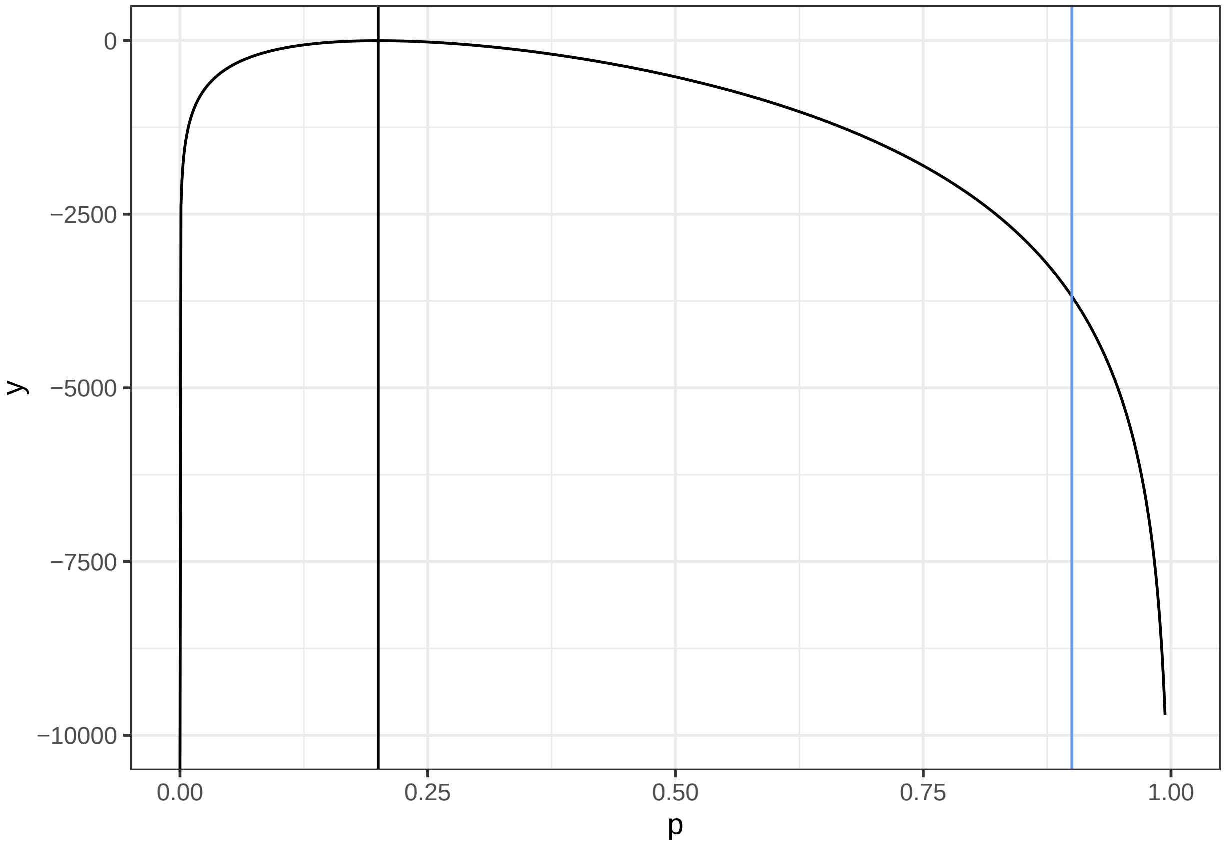
In order to show the method at work, let's start with an initial guess of p = 0.9 (ordinarily, you'd want your initial guess to be as good as possible - but in this case, the parameter estimates converge too quickly for this to be an interesting example if we start much closer to 0.2).

Add a vertical line to the plot below at the initial guess of p = 0.9. Use your favorite color for the line.

```r
loglikelihood <- function(p) {
  result <- vector("numeric", length(p))
  for(i in seq_along(p)) {
    result[i] <- dbinom(540, size = 2700, prob = p[i], log = TRUE)
  }

  return(result)
}
```

```r
ggplot(data = data.frame(p = c(0, 1)), mapping = aes(x = p)) +
  stat_function(fun = loglikelihood, n = 1001) +
  geom_vline(xintercept = 0.2) +
  geom_vline(xintercept = 0.9, color = "cornflowerblue") +
  ylim(c(-10000, 0)) +
  theme_bw()
```



Update guess (first update)

Calculate the first and second derivatives of the log-likelihood function evaluated at the initial guess, and use those values to obtain an updated parameter estimate.

```r
p_old <- 0.9
dl_dp <- (540/p_old + (540 - 2700)/(1 - p_old))
d2l_dp2 <- (-540/(p_old^2) + (540-2700)/((1 - p_old)^2))
p_updated <- p_old - dl_dp / d2l_dp2
p_updated
```

```
## [1] 0.8030769
```

```r
# to start next update, you could set p_old <- p_updated
# and re-use the rest of the code
```

Add a vertical line to the plot below at your updated parameter estimate.

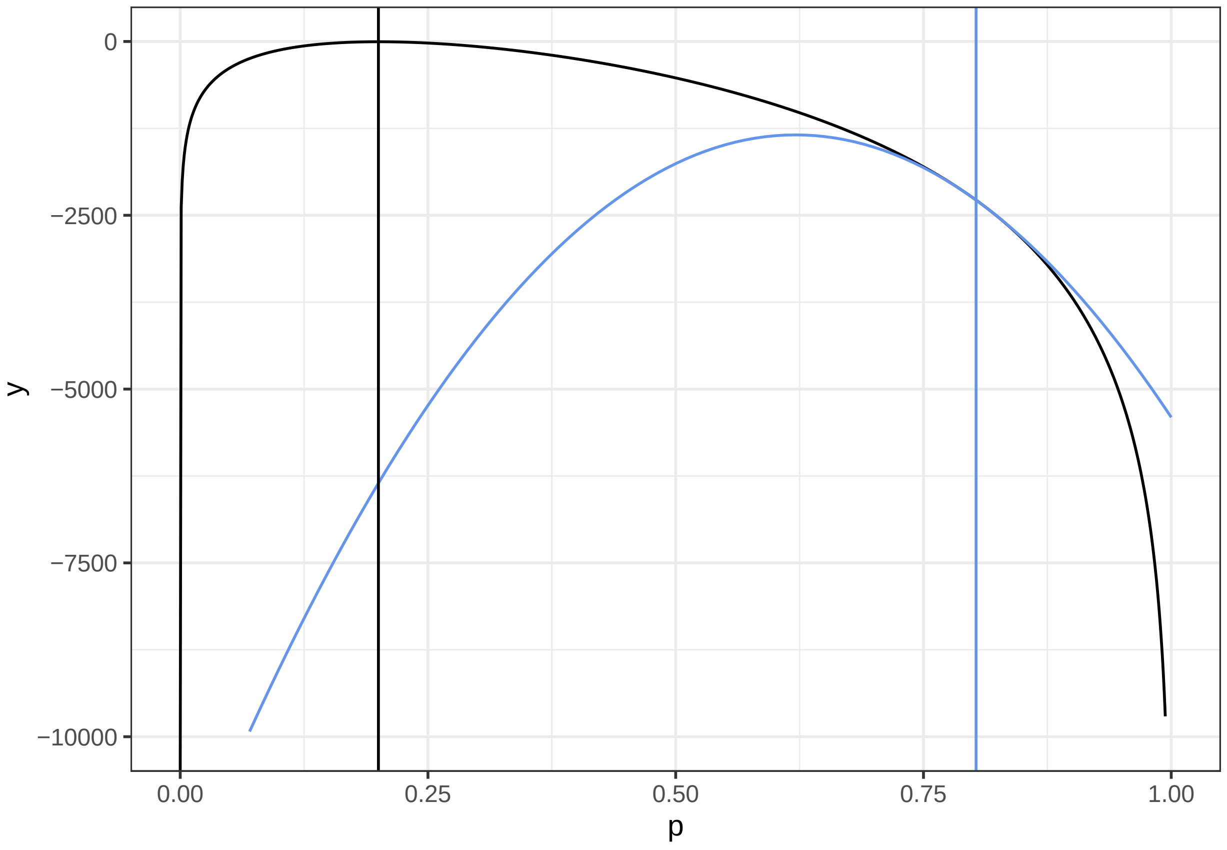
```r
loglikelihood <- function(p) {
  result <- vector("numeric", length(p))
  for(i in seq_along(p)) {
    result[i] <- dbinom(540, size = 2700, prob = p[i], log = TRUE)
  }

  return(result)
}

P_2 <- function(p, p_center) {
  dl_dp <- (540/p_center + (540 - 2700)/(1 - p_center))
  d2l_dp2 <- (-540/(p_center^2) + (540-2700)/((1 - p_center)^2))
  loglikelihood(p_center) +
    dl_dp * (p - p_center) +
    0.5 * d2l_dp2 * (p - p_center)^2
}

ggplot(data = data.frame(p = c(0, 1)), mapping = aes(x = p)) +
  stat_function(fun = loglikelihood, n = 1001) +
  stat_function(fun = P_2,
    args = list(p_center = p_updated),
    color = "cornflowerblue") +
  geom_vline(xintercept = 0.2) +
  geom_vline(xintercept = p_updated, color = "cornflowerblue") +
  ylim(c(-10000, 0)) +
  theme_bw()
```

```
## Warning: Removed 7 rows containing missing values (geom_path).
```



Update guess (second update)

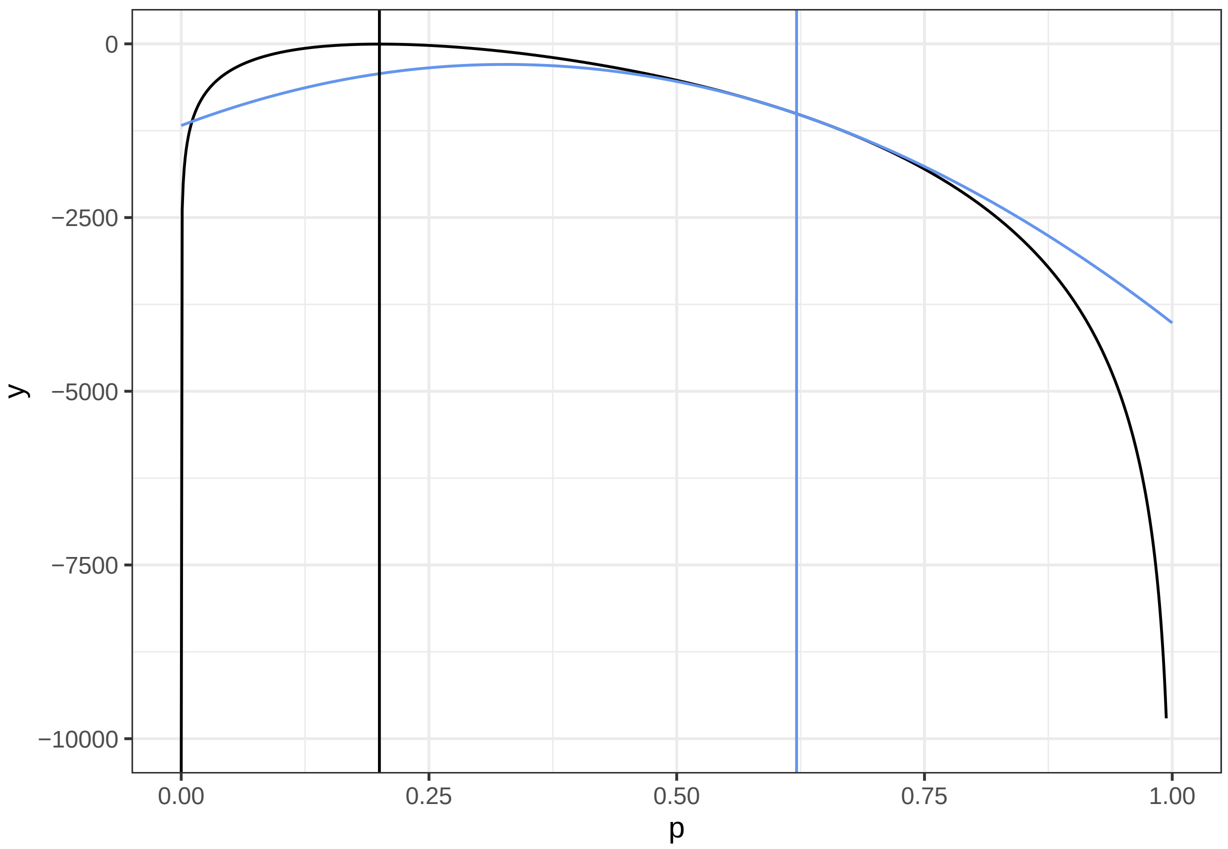
Calculate the first and second derivatives of the log-likelihood function at the updated parameter estimate, and use those values to obtain a new updated parameter estimate.

```r
p_old <- p_updated
dl_dp <- (540/p_old + (540 - 2700)/(1 - p_old))
d2l_dp2 <- (-540/(p_old^2) + (540-2700)/((1 - p_old)^2))
p_updated <- p_old - dl_dp / d2l_dp2
p_updated
```

```
## [1] 0.6209633
```

Add a vertical line to the plot below at your updated parameter estimate.

```r
ggplot(data = data.frame(p = c(0, 1)), mapping = aes(x = p)) +
  stat_function(fun = loglikelihood, n = 1001) +
  stat_function(fun = P_2,
    args = list(p_center = p_updated),
    color = "cornflowerblue") +
  geom_vline(xintercept = 0.2) +
  geom_vline(xintercept = p_updated, color = "cornflowerblue") +
  ylim(c(-10000, 0)) +
  theme_bw()
```



Update guess (third update)

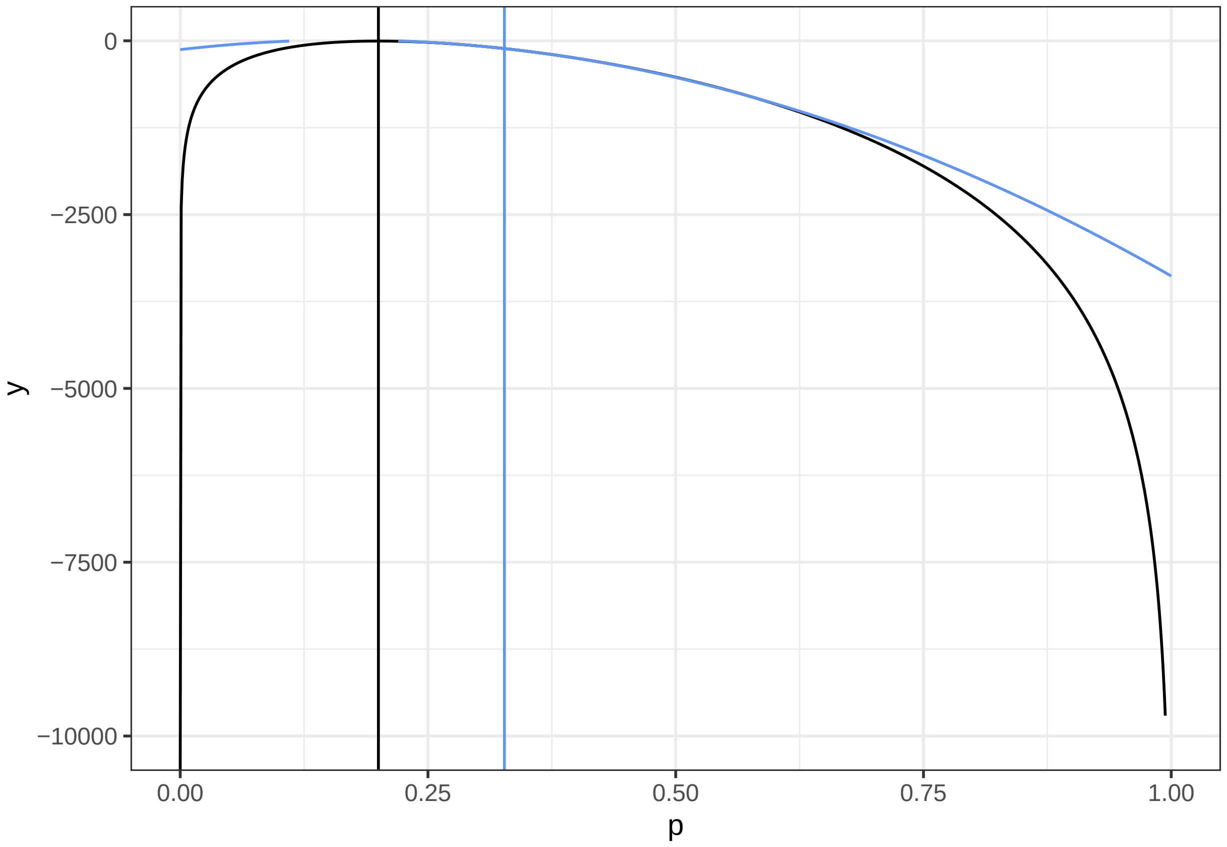
Calculate the first and second derivatives of the log-likelihood function at the updated parameter estimate, and use those values to obtain a new updated parameter estimate.

```
p_old <- p_updated
dl_dp <- (540/p_old + (540 - 2700)/(1 - p_old))
d2l_dp2 <- (-540/(p_old^2) + (540-2700)/((1 - p_old)^2))
p_updated <- p_old - dl_dp / d2l_dp2
p_updated
```

```
## [1] 0.3271369
```

Add a vertical line to the plot below at your updated parameter estimate.

```
ggplot(data = data.frame(p = c(0, 1)), mapping = aes(x = p)) +
  stat_function(fun = loglikelihood, n = 1001) +
  stat_function(fun = P_2,
    args = list(p_center = p_updated),
    color = "cornflowerblue") +
  geom_vline(xintercept = 0.2) +
  geom_vline(xintercept = p_updated, color = "cornflowerblue") +
  ylim(c(-10000, 0)) +
  theme_bw()
```



Update guess (fourth update)

Calculate the first and second derivatives of the log-likelihood function at the updated parameter estimate, and use those values to obtain a new updated parameter estimate.

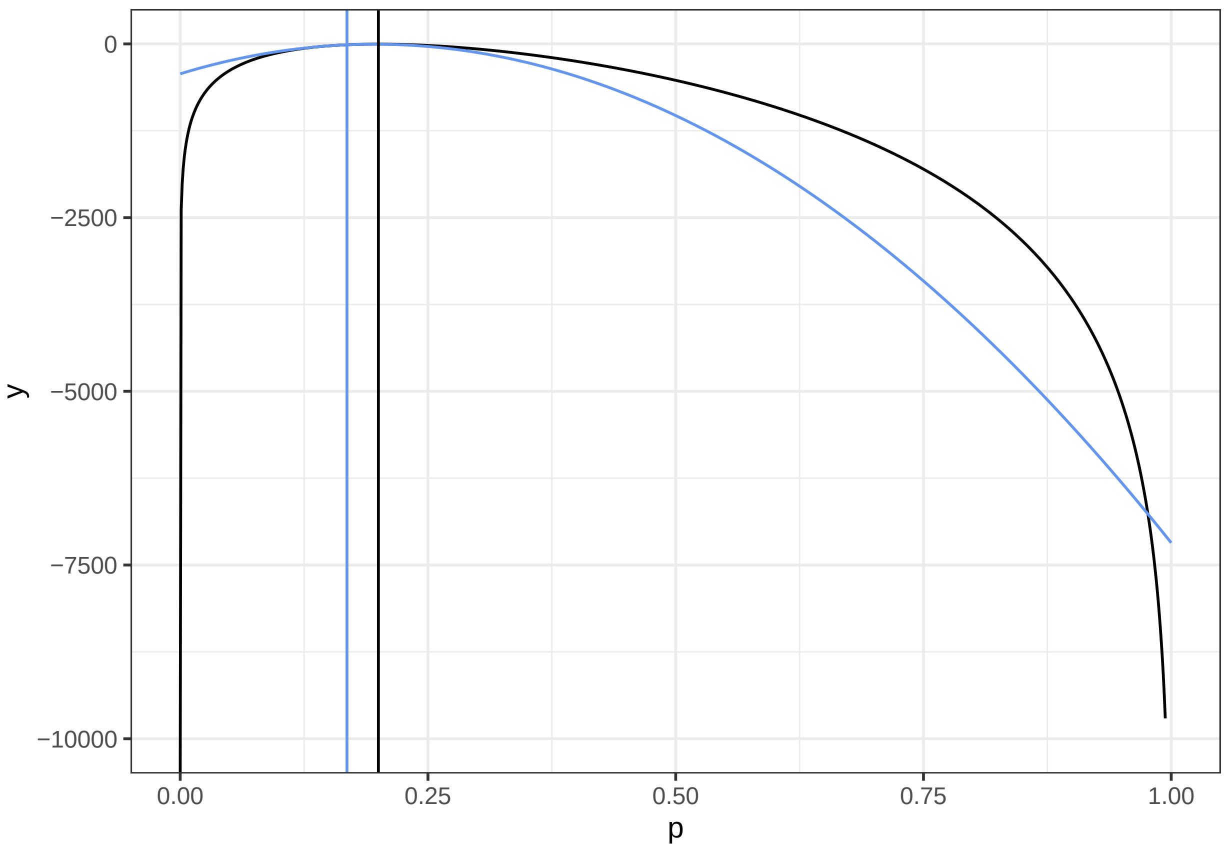
```
p_old <- p_updated
dl_dp <- (540/p_old + (540 - 2700)/(1 - p_old))
d2l_dp2 <- (-540/(p_old^2) + (540-2700)/((1 - p_old)^2))
```

```
p_updated <- p_old - dl_dp / d2l_dp2
p_updated
```

```
## [1] 0.1682781
```

Add a vertical line to the plot below at your updated parameter estimate.

```
ggplot(data = data.frame(p = c(0, 1)), mapping = aes(x = p)) +
  stat_function(fun = loglikelihood, n = 1001) +
  stat_function(fun = P_2,
    args = list(p_center = p_updated),
    color = "cornflowerblue") +
  geom_vline(xintercept = 0.2) +
  geom_vline(xintercept = p_updated, color = "cornflowerblue") +
  ylim(c(-10000, 0)) +
  theme_bw()
```



Update guess (fifth update)

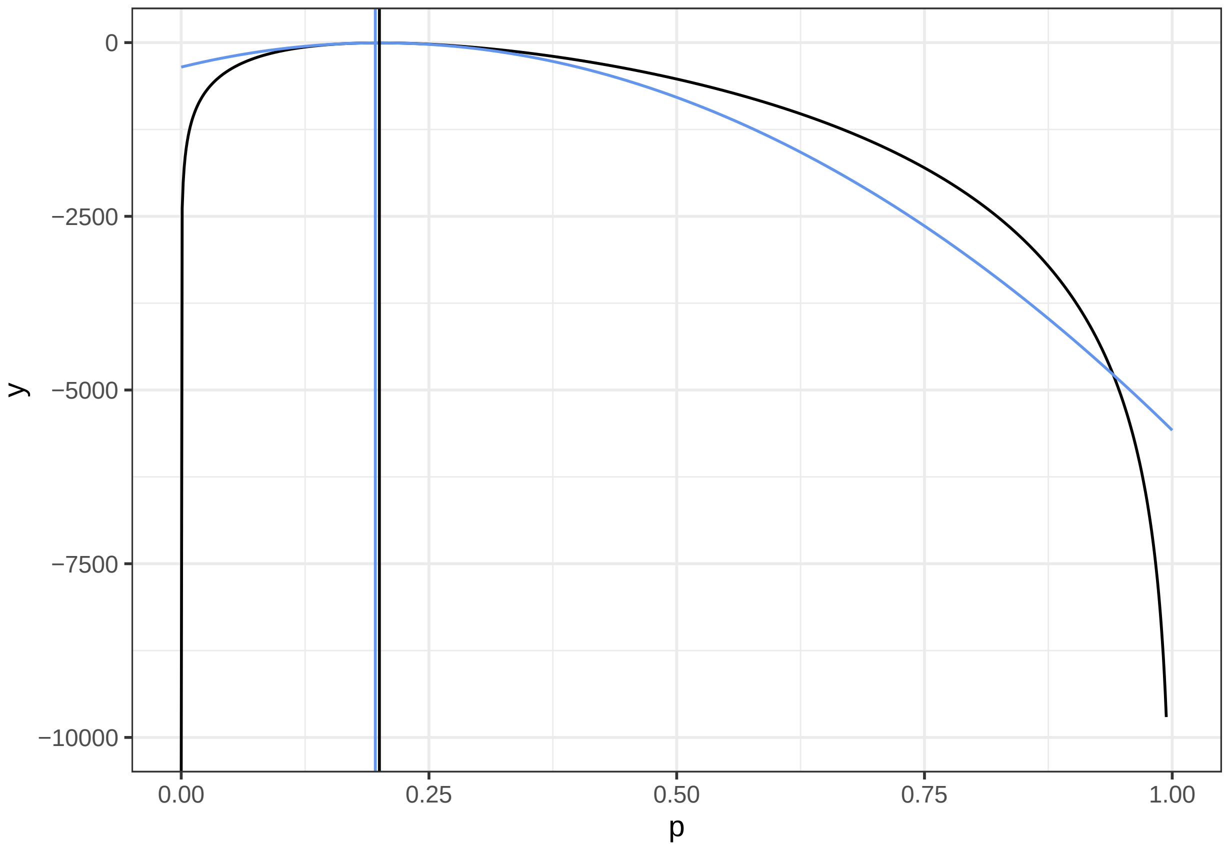
Calculate the first and second derivatives of the log-likelihood function at the updated parameter estimate, and use those values to obtain a new updated parameter estimate.

```
p_old <- p_updated
dl_dp <- (540/p_old + (540 - 2700)/(1 - p_old))
d2l_dp2 <- (-540/(p_old^2) + (540-2700)/((1 - p_old)^2))
p_updated <- p_old - dl_dp / d2l_dp2
p_updated
```

```
## [1] 0.1958535
```

Add a vertical line to the plot below at your updated parameter estimate.

```
ggplot(data = data.frame(p = c(0, 1)), mapping = aes(x = p)) +
  stat_function(fun = loglikelihood, n = 1001) +
  stat_function(fun = P_2,
    args = list(p_center = p_updated),
    color = "cornflowerblue") +
  geom_vline(xintercept = 0.2) +
  geom_vline(xintercept = p_updated, color = "cornflowerblue") +
  ylim(c(-10000, 0)) +
  theme_bw()
```



Update guess (sixth update)

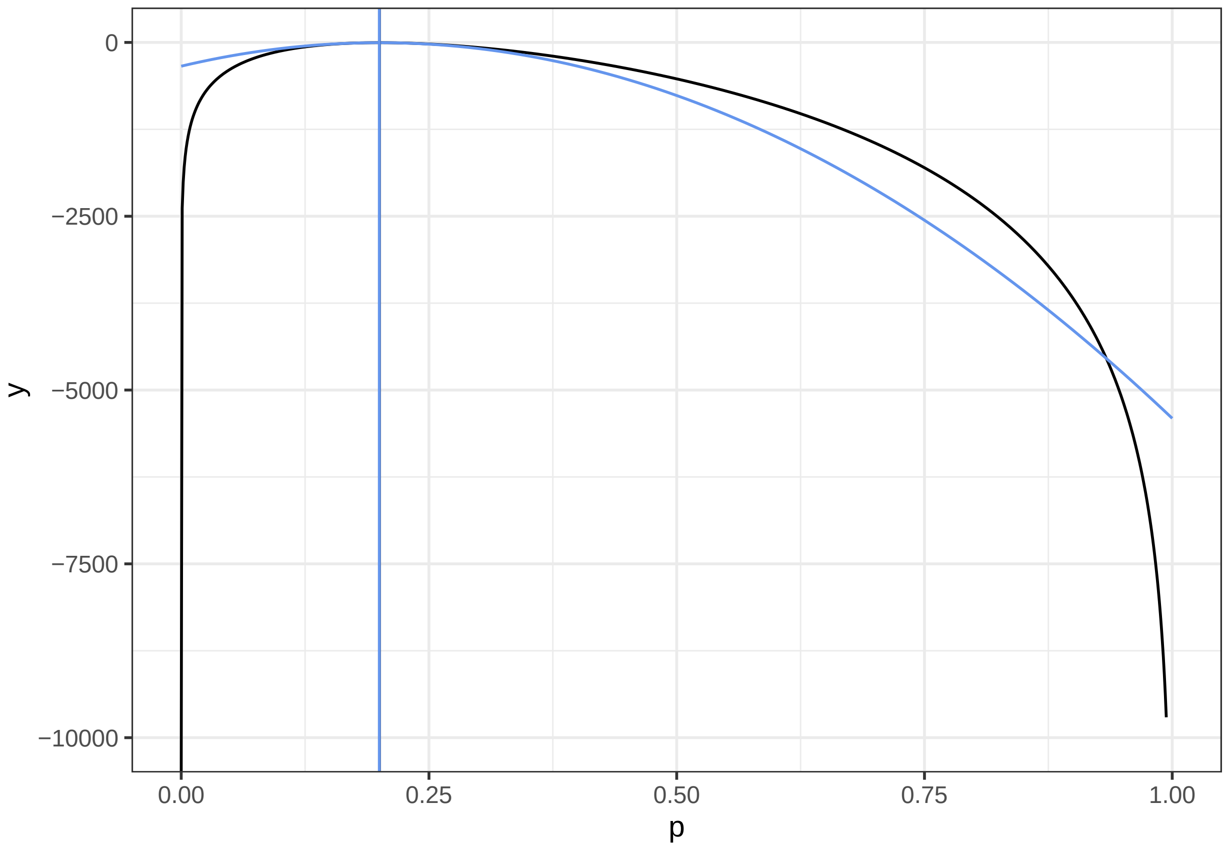
Calculate the first and second derivatives of the log-likelihood function at the updated parameter estimate, and use those values to obtain a new updated parameter estimate.

```
p_old <- p_updated
dl_dp <- (540/p_old + (540 - 2700)/(1 - p_old))
d2l_dp2 <- (-540/(p_old^2) + (540-2700)/((1 - p_old)^2))
p_updated <- p_old - dl_dp / d2l_dp2
p_updated
```

```
## [1] 0.1999346
```

Add a vertical line to the plot below at your updated parameter estimate.

```r
ggplot(data = data.frame(p = c(0, 1)), mapping = aes(x = p)) +
  stat_function(fun = loglikelihood, n = 1001) +
  stat_function(fun = P_2,
    args = list(p_center = p_updated),
    color = "cornflowerblue") +
  geom_vline(xintercept = 0.2) +
  geom_vline(xintercept = p_updated, color = "cornflowerblue") +
  ylim(c(-10000, 0)) +
  theme_bw()
```



Optional, if time

The updates above were obtained by maximizing second-order Taylor series approximations to the log-likelihood. Go back and add a new layer to each of those plots showing the Taylor series approximation at the current estimate that will be used to obtain the next estimate.

I have done this in the code chunks above.

Optional, if even more time

Write a while loop implementing the Newton-Raphson optimization algorithm for this example. Use a tolerance of 1e-8 and set the maximum number of iterations to 100.

```r
tol <- 1e-8
max_iter <- 100

recent_diff <- Inf
```

```r
i <- 0

p_updated <- 0.9

while(recent_diff > tol && i < max_iter) {
  i <- i + 1
  p_old <- p_updated

  dl_dp <- (540/p_old + (540 - 2700)/(1 - p_old))
  d2l_dp2 <- (-540/(p_old^2) + (540-2700)/((1 - p_old)^2))
  p_updated <- p_old - dl_dp / d2l_dp2

  recent_diff <- abs(p_updated - p_old)
}

print(paste0("number of iterations run = ", i))
```

```
## [1] "number of iterations run = 9"
```

```r
print(paste0("last update amount = ", recent_diff))
```

```
## [1] "last update amount = 9.71445146547012e-16"
```

```r
print(paste0("final estimate = ", p_updated))
```

```
## [1] "final estimate = 0.2"
```