Introduction to Binomial GLMs

Sam Mason

The Binomial Distribution

We've discussed how the Bernoulli distribution is simply a special case of the binomial distribution were there is only a single trial. Let's dig a bit deeper into the binomial when our data is generated from a process with more than a single trial.

Support: $y \in \{0, 1, 2, ..., n\}$
Parameters: $p \in [0, 1]$ (the probability of success); $n \in \{0, 1, 2, ..., n\}$ (the number of trials); $q = 1 - p$
First Moment: $n * p$
Second Moment: $n * p * q$

I think I said that we wouldn't be spending a ton of time looking at probability mass functions (the functions that define the shape of a distribution describing discrete values), but maybe I lied a little bit.

$$PMF = \frac{n!}{y!(n-y)!}p^y(1-p)^{n-y}$$

Let's ignore the fraction of factorials (!) for now and look at the exponentiated p and q in the right-hand portion of the PMF. We'll take a simple example where n = 3, y = 2 (out of 3 trials, we observed 2 successes), and p = 0.30. The probability of observing, say, trial 1 = success, trial 2 = success, and trial 3 = failure would be $0.3 * 0.3 * (1 - 0.3) = 0.063$. This is equivalent to $0.3^2(0.7)^1 = 0.063$. So. . . shouldn't that just be it? What's all the factorial business? Let's start by working out the value of that fraction in the context of our example.

$$\frac{3!}{2!(3-2)!} = \frac{3 * 2 * 1}{2 * 1 * (1)} = 3$$

This fraction is called the "binomial coefficient" (often represented as $\binom{n}{y}$), and tells us the number of different ways that we could get y = 2 given n = 3. The options are: success|success|failure, success|failure|success, and failure|success|success. Given n = 3, we could generate y = 2 in three different ways. Scenario 1 **OR** scenario 2 **OR** scenario 3. Each scenario has a probability of 0.063, and because we recognize that our y = 2 could be the result of any one of these three options, we add the probabilities together.

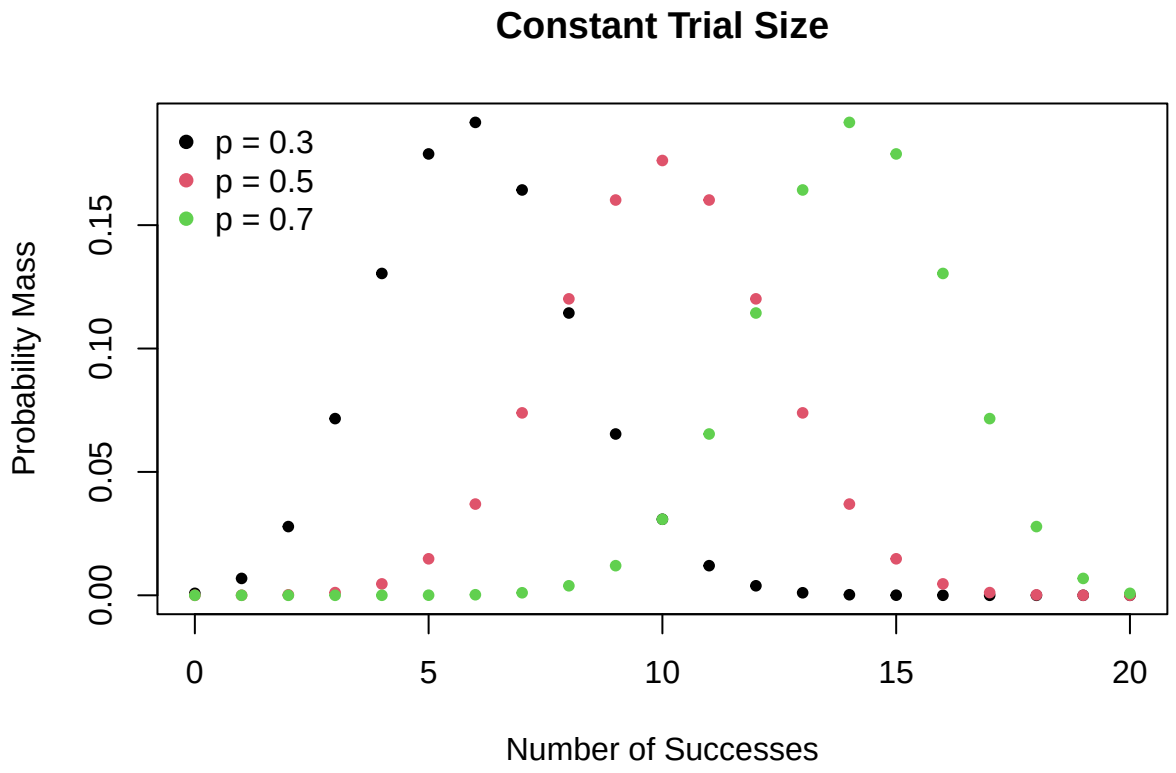
$$0.063 + 0.063 + 0.063 = 3 * 0.063 = 0.189$$

How neat is that?! Aren't you glad we took a look at the binomial PMF? Let's make sure R agrees with our work above.

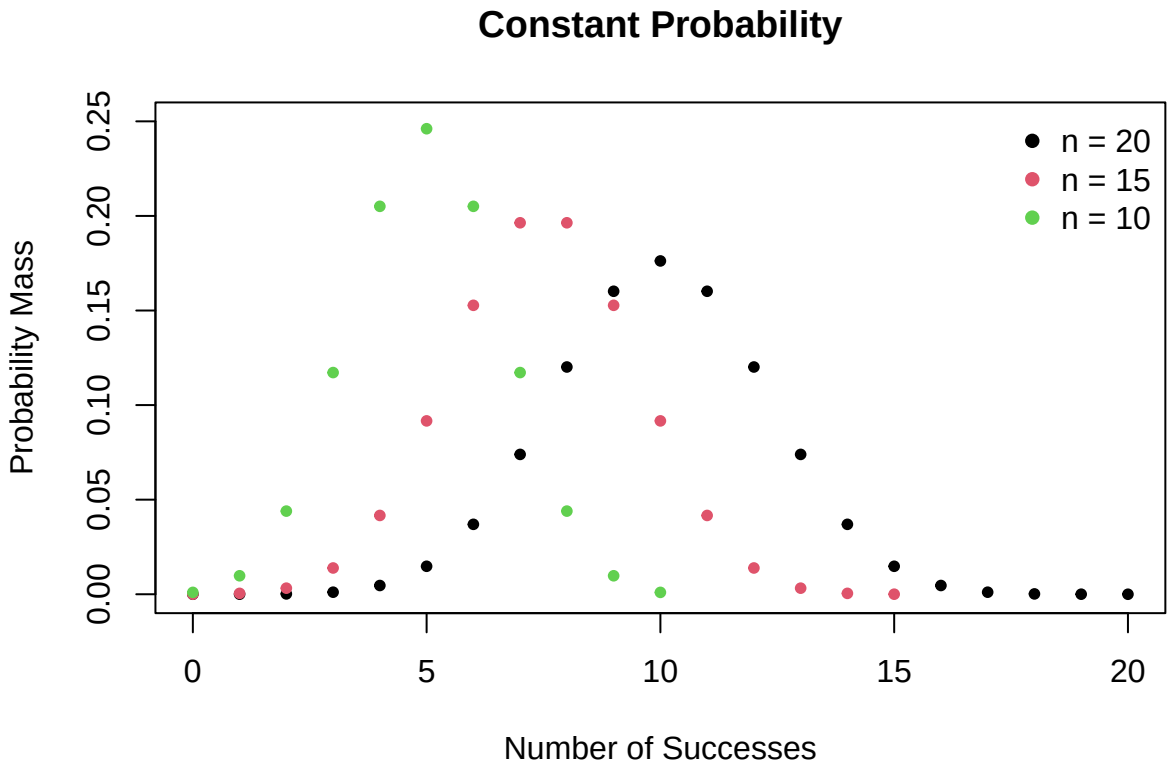
```
dbinom(x = 2, size = 3, prob = 0.3)
```

```
## [1] 0.189
```

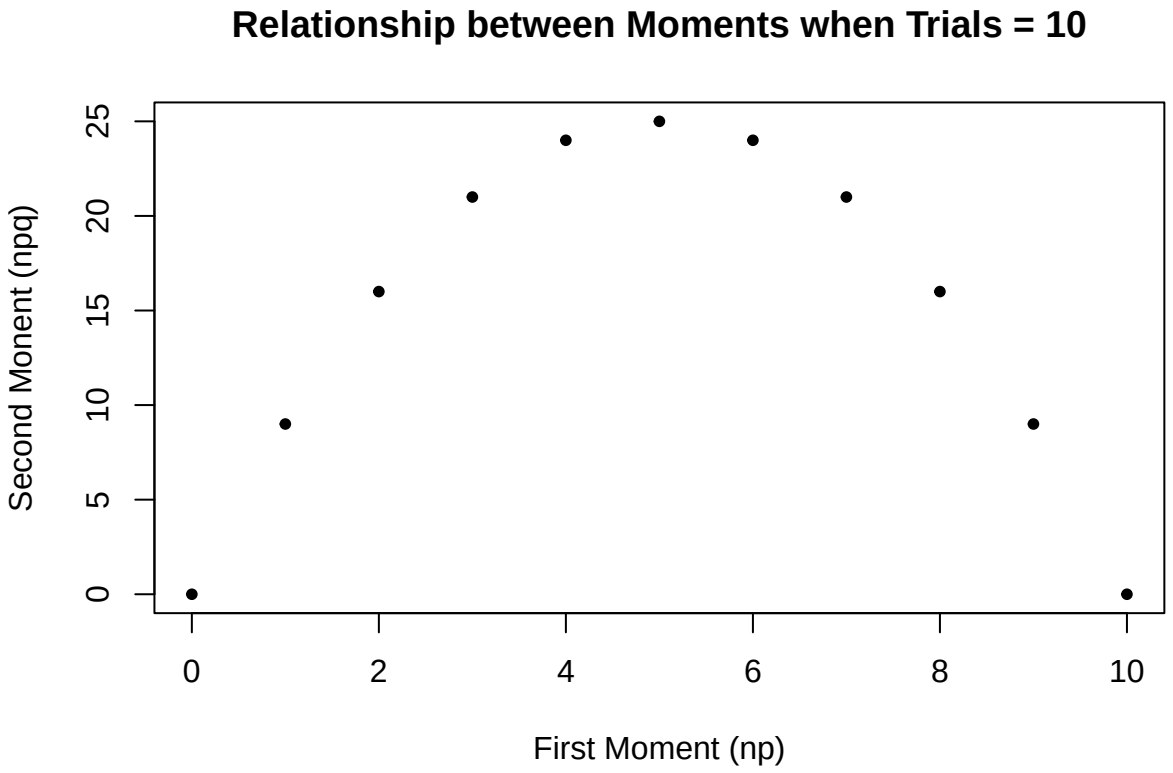
```r
# Constant trial size (n)
plot(x = 0:20, y = dbinom(0:20, 20, 0.3), pch = 20,
     main = "Constant Trial Size", xlab = "Number of Successes", ylab = "Probability Mass")
points(x = 0:20, y = dbinom(0:20, 20, 0.5), pch = 20, col = 2)
points(x = 0:20, y = dbinom(0:20, 20, 0.7), pch = 20, col = 3)
legend("topleft", legend = c("p = 0.3", "p = 0.5", "p = 0.7"), pch = 16,
       bty = "n", col = 1:3)
```




```r
# Constant probability (p)
plot(x = 0:20, y = dbinom(0:20, 20, 0.5), pch = 20,
     main = "Constant Probability", xlab = "Number of Successes",
     ylab = "Probability Mass", ylim = c(0, 0.25))
points(x = 0:15, y = dbinom(0:15, 15, 0.5), pch = 20, col = 2)
points(x = 0:10, y = dbinom(0:10, 10, 0.5), pch = 20, col = 3)
legend("topright", legend = c("n = 20", "n = 15", "n = 10"), pch = 16,
       bty = "n", col = 1:3)
```




```
n <- 10
m1 <- n*seq(0, 1, 0.1)
q <- 1-seq(0, 1, 0.1)
m2 <- n*m1*q
plot(m1, m2, pch=20, xlab = 'First Moment (np)', ylab = 'Second Monent (npq)',
     main = "Relationship between Moments when Trials = 10")
```



Glancing at the figures above reveals that the population mean shifts right as the probability or number of trials increases. The spread of a binomial distribution increases with increasing trial size, or as the probability of success approaches 0.5 from either extreme (0 or 1).

The Binomial GLM

Let y be a vector of length n containing realizations ($y = \{y_1, y_2, y_3, ..., y_n\}$) of a corresponding set of independent random variables $Y = \{Y_1, Y_2, Y_3, ..., Y_n\}$. We can model the variation in the probability of success of the binomial distribution describing each of the random variables as a function of the inverse logit of a linear combination of beta coefficients (β) and model predictors (x) according to the general form described below.

$$Y_i \sim binom(n_i, p_i)$$

$$logit(p_i) = ln\frac{p_i}{1-p_i} = \beta_0 + \beta_1 * x_{1,i} + \beta_2 * x_{2,i} + ... + \beta_k * x_{k,i}$$

$$p_i = \frac{e^{\beta_0+\beta_1*x_{1,i}+\beta_2*x_{2,i}+...+\beta_k*x_{k,i}}}{1+e^{\beta_0+\beta_1*x_{1,i}+\beta_2*x_{2,i}+...+\beta_k*x_{k,i}}}$$

This looks almost identical to the Bernoulli GLM except for the first line where each random Y is described by a binomial distribution with both parameters n and p. We'll need to specify the trial size for each of our random variable realizations y when specifying a binomial GLM in R. Because we're ultimately modeling the log-odds of success when fitting both Bernoulli and binomial GLMs, the link function is the same.

Fitting Binomial GLMs in R

To show you the syntax, I'll create a very simple and arbitrary binomial dataset.

```
y <- c(2, 4, 2, 5, 1, 1, 3, 2, 5, 4)
n <- c(6, 5, 8, 5, 2, 5, 7, 9, 6, 9)
f <- n-y # number of failures
r <- y/n # proportion of successes to trials
mod_syntax1 <- glm(cbind(y, f) ~ 1, family = binomial(link = "logit"))
mod_syntax2 <- glm(r ~ 1, weights = n, family = binomial(link = "logit"))
```

Using the first syntax option, you pass both the response (the number of successes) and the number of failures into the function as a two-column matrix. In the second syntax option, you provide the proportions as the response and set the model weights to the trail size (denominator) for each observation. Both options will fit exactly the same model.

```
summary(mod_syntax1)$coefficients
```

```
##               Estimate Std. Error    z value  Pr(>|z|)
## (Intercept) -0.1292117  0.2545305 -0.5076473 0.6117007
```

```
summary(mod_syntax2)$coefficients
```

```
##               Estimate Std. Error    z value  Pr(>|z|)
## (Intercept) -0.1292117  0.2545305 -0.5076473 0.6117007
```

There are no predictors in this model, so we can interpret the intercept as the average log-odds of success across all random variables from which our observations are drawn. Of course, I didn't actually observe `y` through a binomial process. I just made up some data that looks binomial. Let's generate some response data from an actual binomial process.

Task 1

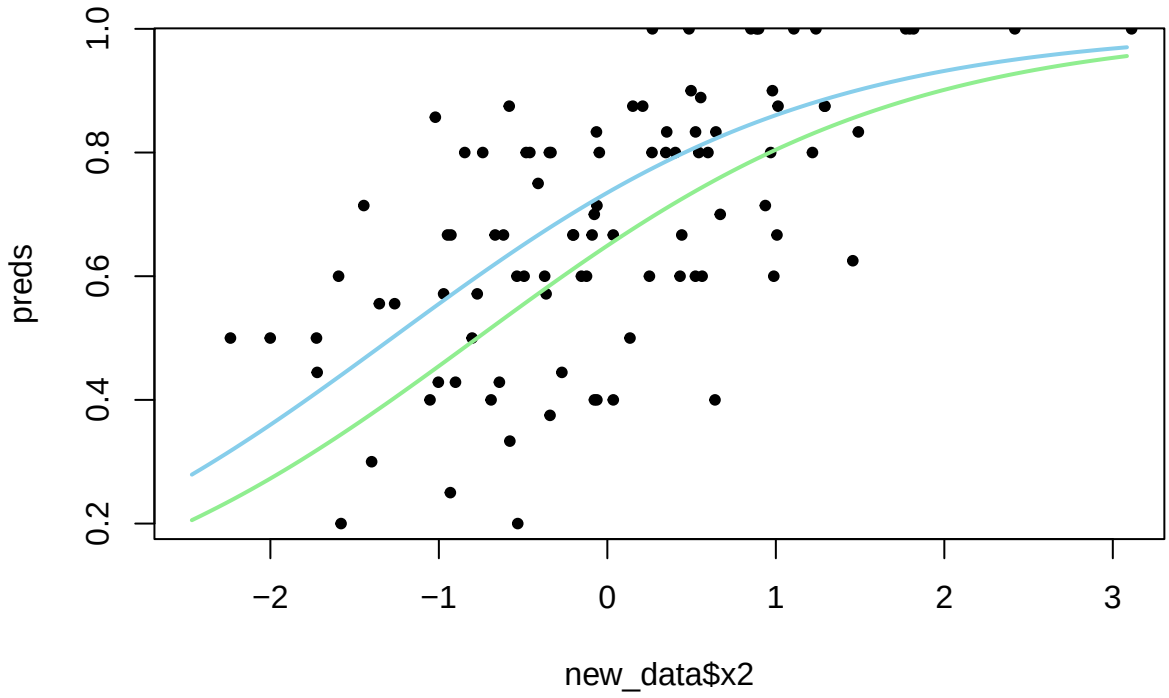
Generate a vector of binomial response data (length = 100) from a model with an intercept, a continuous predictor, and a two-level categorical predictor (factor). Use the following values for your linear predictor, where x_1 is a dummy variable containing 1s when the predictor is equal to its second level, and 0s when the predictor is equal to its first level. To make this easy on yourself, just make a vector containing 50 0s and then 50 1s. You'll need to create a vector of trial sizes as well. Once you've generated the response data, run a model using the `glm()` function to see if you can recover the beta coefficients you used to generated the data (you'll need to make sure the categorical predictor is a factor in the model, so that R doesn't try to fit a slope to it). See the end of the document for my solution.

$$\beta_0 + \beta_1 * x_1 + \beta_2 * x_2 = 1.1 + -0.5 * x_1 + 0.7 * x_2$$

Let's visualize this model together. Walk through my code slowly and make sure everything makes sense to you. We'll interpret the resulting plot together as a group.

```
r <- y/n # the proportions are the individual data points
new_x2 <- seq(min(x2), max(x2), 0.05)
new_data <- data.frame(x1 = factor(c(rep(0, length(new_x2)), rep(1, length(new_x2)))),
```

```
                        x2 = rep(new_x2, 2))
preds <- predict(mod, newdata = new_data, type = "response")
plot(x = new_data$x2, y = preds, type = "n")
points(x = x2, y = r, pch = 20)
lines(x = new_x2, y = preds[new_data$x1 == 0], lwd = 2,
      col = "skyblue")
lines(x = new_x2, y = preds[new_data$x1 == 1], lwd = 2,
      col = "lightgreen")
```



While we're here let's look at the residuals of the model you fit in **Task 1**. We know for a fact that the binomial GLM with predictors x_1 and x_2 is the correct model to fit to this response data because we just generated the data ourselves. Looking at the residual plots will be a helpful exercise in seeing an example of an appropriate model.

Task 2

Create a plot of the Pearson residuals against values of the linear predictor. Add a gray, horizontal dashed line at 0. In my solution, I add a locally weighted polynomial regression line using the `loess()` function to fit a line that helps to visualize trends in the residuals across the domain of the linear predictor. You don't have to do this.

Let's think about this plot some more. Remember, this type of diagnostic plot is meant to help you understand potential problems with your model, particularly non-linear relationships behavior. Non-linear residuals might indicate that your link function is incorrect, but more commonly indicates that your model should be fit with a quadratic term. We'll talk about fitting quadratic predictors later. We also expect

the Pearson residuals, being scaled by the standard deviation of each random response observation, to fall "evenly" on either side of the trend line across the entire domain of the linear predictor. We don't want the residuals to be small for certain values of the linear predictor, and large for others. When residuals behave in this way we say that they are *heteroscedastic*. We're always looking to achieve *homoscedasticity* in our residual plots. Heteroscedastic residuals might be indicative of a poorly parameterized model (missing an important predictor). We can check for heteroscedasticity using the plot you created in **Task 2**. It's easy to catch a glimpse of heteroscedastic behavior at the extremes of the linear predictor, but you should be cautious here. In many models, there is less data at the extremes of the linear predictor, which can make it seem like there is less (or sometimes more) deviation in those locations. Take a look at my solution. The model I used to generate the response data is the same model I specify in `glm()`. I know for a fact that this model is correct, and yet there's still some wonky behavior at the extremes.

Task 3

Another very helpful model diagnostic is the quantile-quantile (Q-Q) plot. The idea here is that the standardized Pearson residuals should be largely normally distributed. That is we should have both positive and negative residuals, with most being close to the trend line and fewer falling far from the line. Let's start this process by getting a feel for what a quantile is. Create a vector containing the sequence of natural numbers from 0 to 100. Call the `quantile()` function on this vector with `probs = seq(0, 1, 0.1)`. Keeping in mind that a quantile defines the proportion of the data falling below a particular point in the ordered vector, interpret the resulting output.

Taks 4

Create another vector containing the standardized Pearson residuals for this model. Create objects holding the mean and standard deviation of these residuals. Create a vector containing 1000 draws from a normal distribution defined by the mean and standard deviation of the standardized Pearson residuals. This vector represents the theoretical shape the residuals would have were they truly normally distributed. Calculate the percentiles (quantiles corresponding the probabilities `seq(0, 1, 0.01)`) of both the observed residuals, and the theoretical residuals. Plot the observed percentiles on the y-axis, and the theoretical percentiles on the x-axis. Draw a straight, red line with intercept = 0 and slope = 1 through this data to represent a perfect 1:1 relationship between quantiles.

The last diagnostic figure we'll look at here addresses the assumption that all observations are independent. That is, the response data is not spatially or temporally auto-correlated. We'll discuss examples of spatial and temporal auto-correlation together as a class. We can check for spatial auto-correlation by aggregating residuals into the discrete spatial units used to collect the response. This simulated example data is not spatial, and so we have no motivation to check for this kind of auto-correlation here. We could, for the sake of demonstration, argue that the data was collected over a time period. The first element of the response vector was collected first, then the second, then the third, etc.

Task 5

Plot the standardized Pearson residuals (y-axis) against the order of data collection (x-axis). Set `type = "l"`. You should end up with a zig-zag.

Solutions

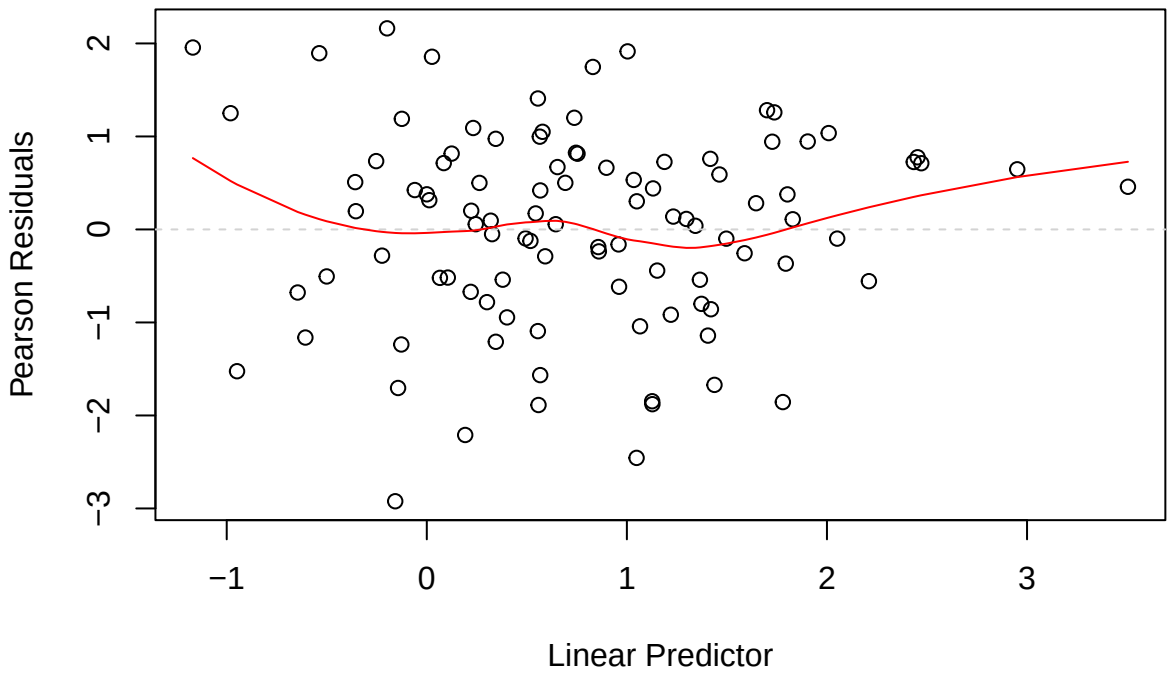
Task 1

```
set.seed(61221)
# Model Parameters
b0 <- 1.1
b1 <- -0.5
b2 <- 0.7

# Data
n <- sample(5:10, 100, replace = T) # trials
x1 <- c(rep(0, 50), rep(1, 50))
x2 <- rnorm(100)

# Linear predictor
lp <- b0 + b1*x1 + b2*x2

# Inverse linear predictor
ilp <- exp(lp)/(1 + exp(lp))

# Observing each random variable
y <- numeric(100)
for(i in 1:100){
  y[i] <- rbinom(1, n[i], ilp[i])
}

# Fitting a model to assess parameter recovery
fails <- n - y
mod <- glm(cbind(y, fails) ~ factor(x1) + x2, family = binomial(link = "logit"))
coefficients(mod)
```

```
## (Intercept) factor(x1)1          x2
##   1.0202830  -0.4038230   0.7983391
```

Task 2

```
# Creating a locally weighted smoothing line
smooth <- loess(residuals(mod, type = "pearson") ~ mod$linear.predictors)
plot(residuals(mod, type = "pearson") ~ mod$linear.predictors,
     xlab = "Linear Predictor", ylab = "Pearson Residuals")
abline(h = 0, lty = 2, col = "lightgray")
lines(sort(smooth$x), smooth$fitted[order(smooth$x)], col = "red")
```



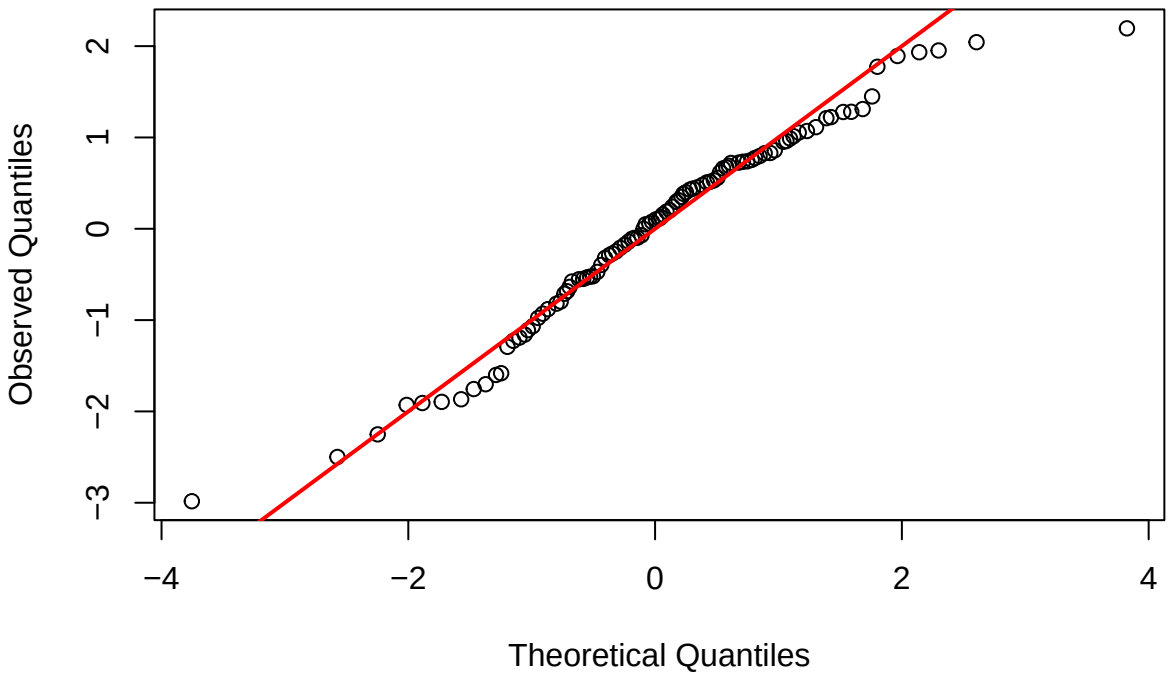
Task 3

```
a <- 0:100
quantile(a, probs = seq(0, 1, 0.1))
```

```
##   0%  10%  20%  30%  40%  50%  60%  70%  80%  90% 100%
##    0   10   20   30   40   50   60   70   80   90  100
```

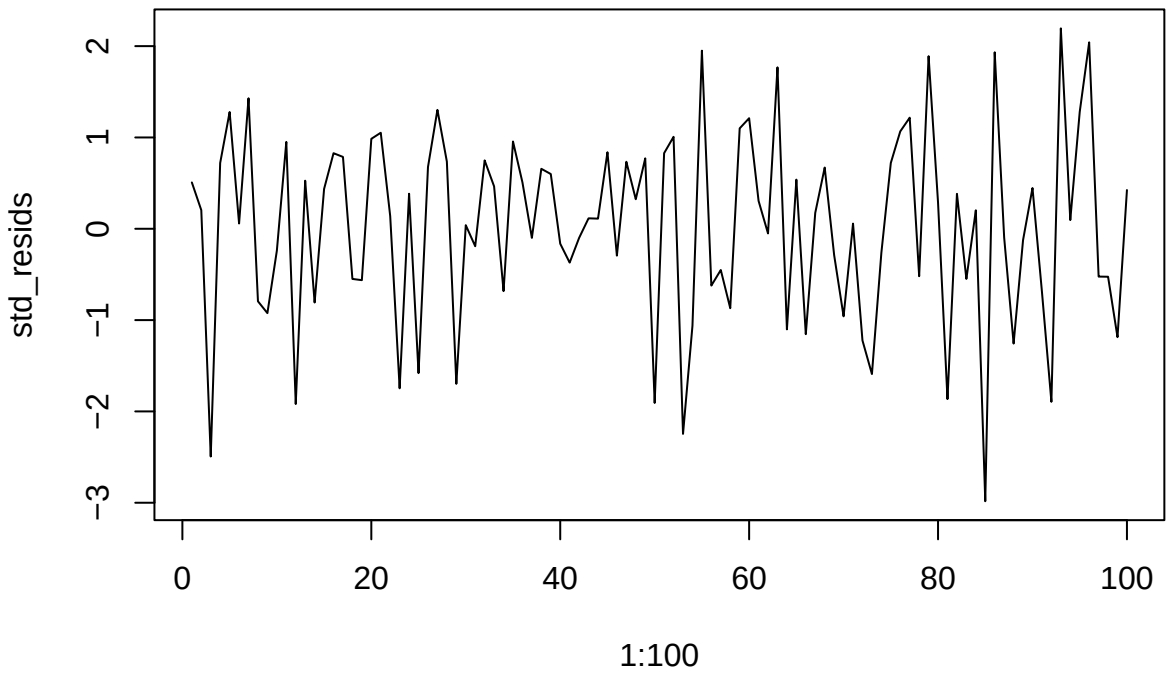
Task 4

```
p_resids <- residuals(mod, type = "pearson")
std_resids <- p_resids/sqrt(1 - hatvalues(mod))
resid_mu <- mean(std_resids)
resid_sd <- sd(std_resids)
norm_resids <- rnorm(1000, mean = resid_mu, sd = resid_sd)
resid_quants <- quantile(std_resids, probs = seq(0, 1, 0.01))
norm_quants <- quantile(norm_resids, probs = seq(0, 1, 0.01))
plot(resid_quants ~ norm_quants, xlab = "Theoretical Quantiles",
     ylab = "Observed Quantiles")
abline(0, 1, lwd = 2, col = "red")
```



Task 5

```
plot(1:100, std_resids, type = "l")
```


std_resids
2
1
0
−1
−2
−3
0
20
40
60
80
100
1:100