Homework Assignment 1

Greg Forkutza
Student ID: 400277514

16 October, 2023

Contents

1

```
set.seed(1234)
# Load data and transform dependent variable into a binary factor variable.
data("kyphosis", package = "rpart")
kyphosis <- transform(kyphosis, Kyphosis = as.numeric(factor(Kyphosis)) - 1)
```

a

`Kyphosis`, is a binary dependent variable. It is therefore appropriate to model the prediction of the outcome using logistic regression. In the `glm` framework and by extension the `gam` framework, the `family =`

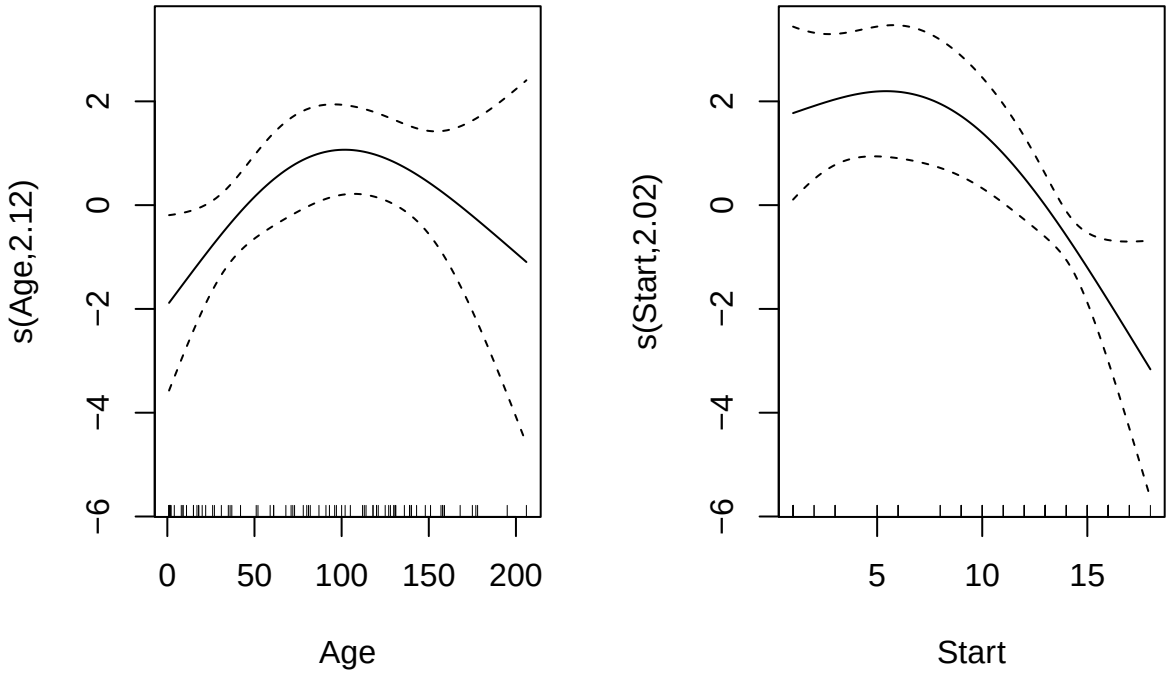
`binomial()` argument indicates that we want to model the response variable using logit link function with binomial random errors. This sets up a logistic regression model.

In table 10.1 of Hastie and Tibshirani [1990] they establish which effects should be included by utilizing a backward/forward step wise selection strategy. The end result is the model that happens to have the lowest deviance, model (x) in Table 10.1. This model includes only the smooth terms (cubic splines with df =3) for `age` and `start`. More explicitly the model is $Kyphosis \sim s(age, df = 3) + s(start, df = 3)$.

```
# Fit the GAM model with two smooth terms and 3 df
fit_gam <- gam(Kyphosis ~ s(Age, k=4) + s(Start, k=4),
               family = binomial, data = kyphosis)
summary(fit_gam)
```

```
##
## Family: binomial
## Link function: logit
##
## Formula:
## Kyphosis ~ s(Age, k = 4) + s(Start, k = 4)
##
## Parametric coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  -2.1993     0.4913  -4.477 7.58e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Approximate significance of smooth terms:
##            edf Ref.df Chi.sq p-value
## s(Age)   2.118  2.539  6.529 0.06074 .
## s(Start) 2.018  2.404 12.673 0.00321 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## R-sq.(adj) =  0.329   Deviance explained =   37%
## UBRE = -0.2253  Scale est. = 1         n = 81
```

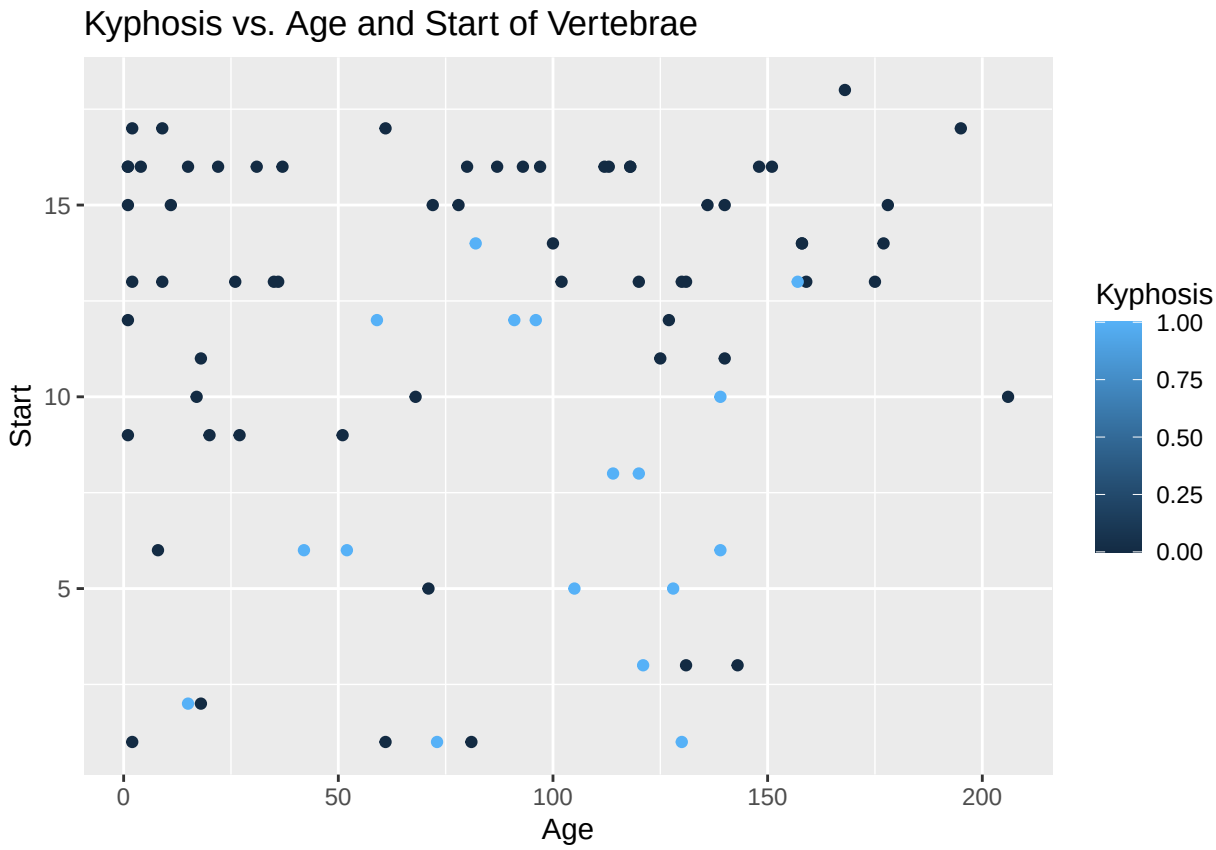
```r
# Plot the smooth terms
plot(fit_gam, pages = 1)
```



b

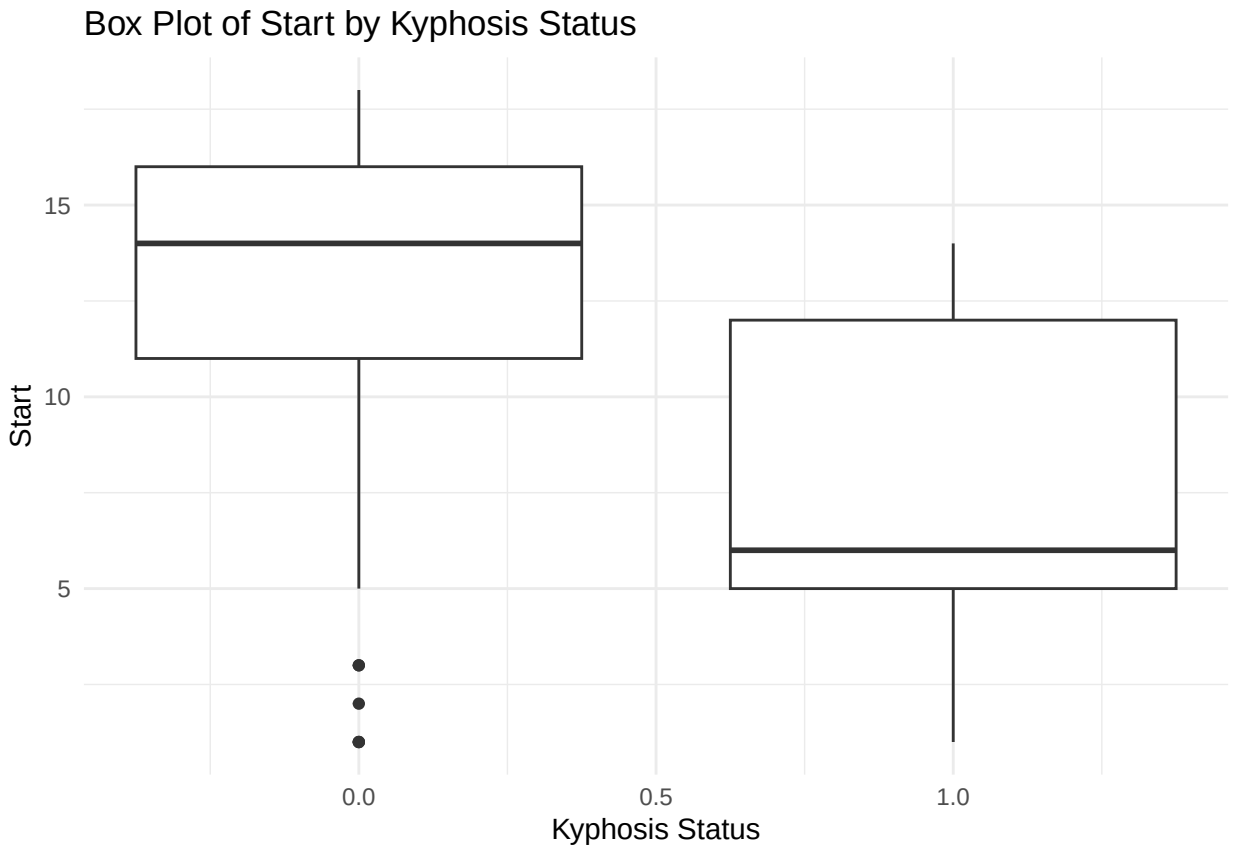
We can plot the relationship between the predictors and the response.

```r
# Scatterplot
ggplot(kyphosis, aes(x = Age, y = Start, color = Kyphosis)) +
  geom_point() +
  labs(title = "Kyphosis vs. Age and Start of Vertebrae")
```




```
# Box plot for Age
ggplot(kyphosis, aes(x = Kyphosis, y = Age, group = Kyphosis)) +
  geom_boxplot() +
  labs(title = "Box Plot of Age by Kyphosis Status",
       y = "Age", x = "Kyphosis Status") +
  theme_minimal()
```


```
# Boxplot for Start
ggplot(kyphosis, aes(x = Kyphosis, y = Start, group = Kyphosis)) +
  geom_boxplot() +
  labs(title = "Box Plot of Start by Kyphosis Status",
       y = "Start", x = "Kyphosis Status") +
  theme_minimal()
```



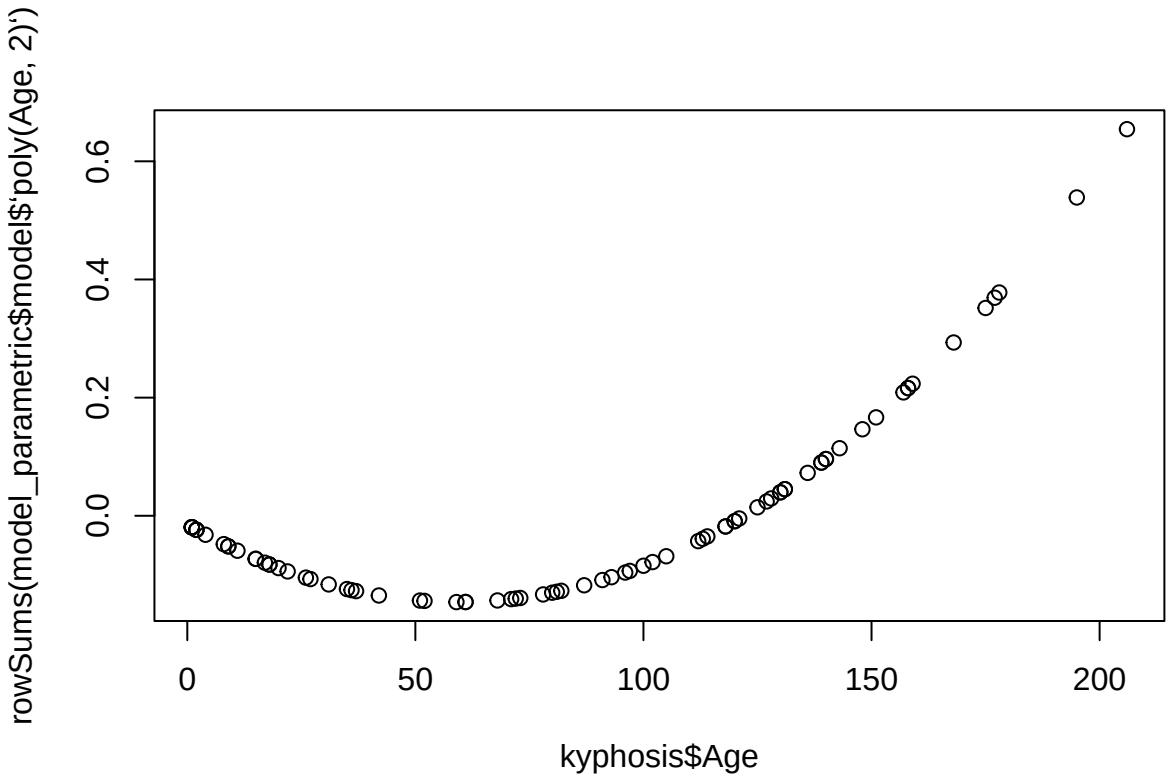
c

The model can be simplified ever further. As in in Hastie and Tibshirani [1990], we can see above that when we plot the smooth terms against the effect on the log odds of `Kyphosis` that `Age` is approximately quadratic. `Start` can be divided into two groups, the thoracic vertebrae $ start 12$ and the lumbar vertebrae $ start $ > 12$. This results in constant fit for thoracic group joined continuously fit to a linear fit for the lumbar group They define a parametric approximation approximation of the two term smooth model as

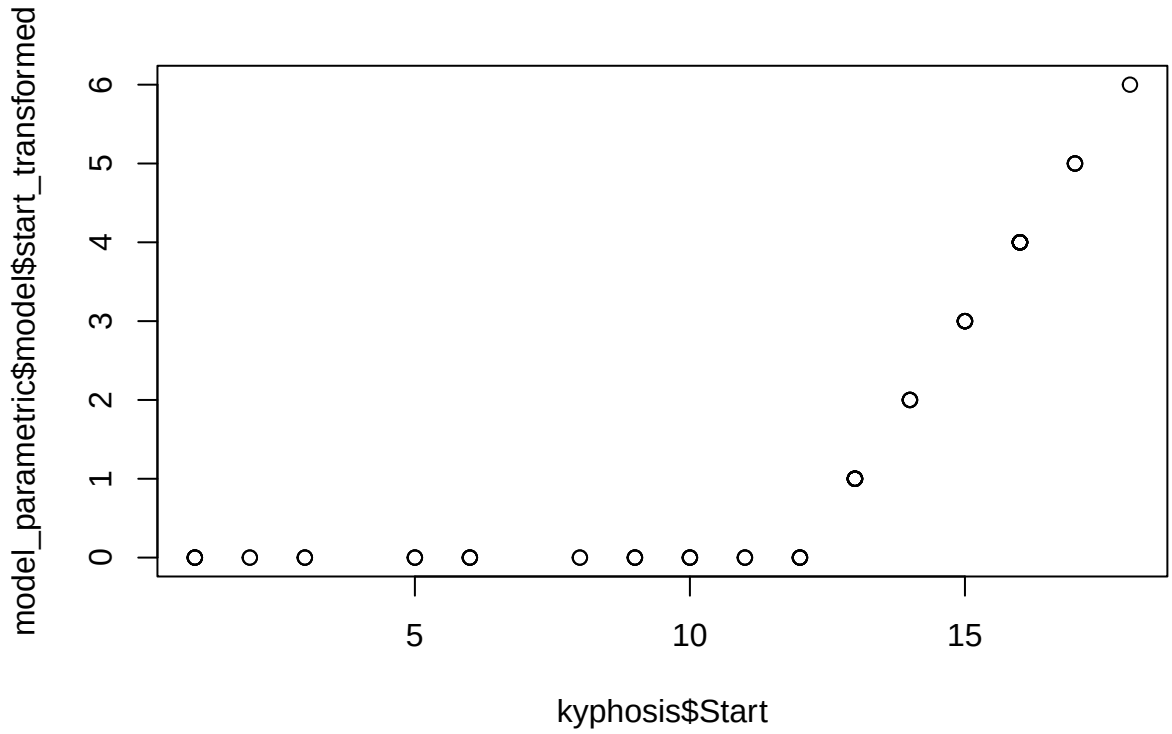
$$Kyphosis \sim poly(Age, 2) + (Start - 12) * I(Start > 12).$$

This model has a quadratic term for age and an interaction term that uses a dummy variable to capture the change in slope for start before and after the value of 12. The $poly(age, 2)$ function creates orthogonal polynomial contrasts for age. This means it fits both linear and quadratic terms for age in the model. The term $(start - 12) * I(start > 12)$ creates an interaction where values of start below or equal to 12 will have a coefficient of 0 for the interaction term, while values above 12 will have a coefficient reflecting $(start - 12)$. T Hastie and Tibsharani, in Table 10.2, state that this model performs the best given that it is parsimonious and captures the functional form suggested by the non parametric model above.

```r
# Create the interaction term
kyphosis$start_transformed <- (kyphosis$Start - 12) *
  as.integer(kyphosis$Start > 12)

# Fit parametric model
model_parametric <- gam(Kyphosis ~ poly(Age, 2) + start_transformed,
                        family = binomial, data = kyphosis)



plot(kyphosis$Age, rowSums(model_parametric$model$`poly(Age, 2)`))
```




```r
plot(kyphosis$Start, model_parametric$model$start_transformed)
```

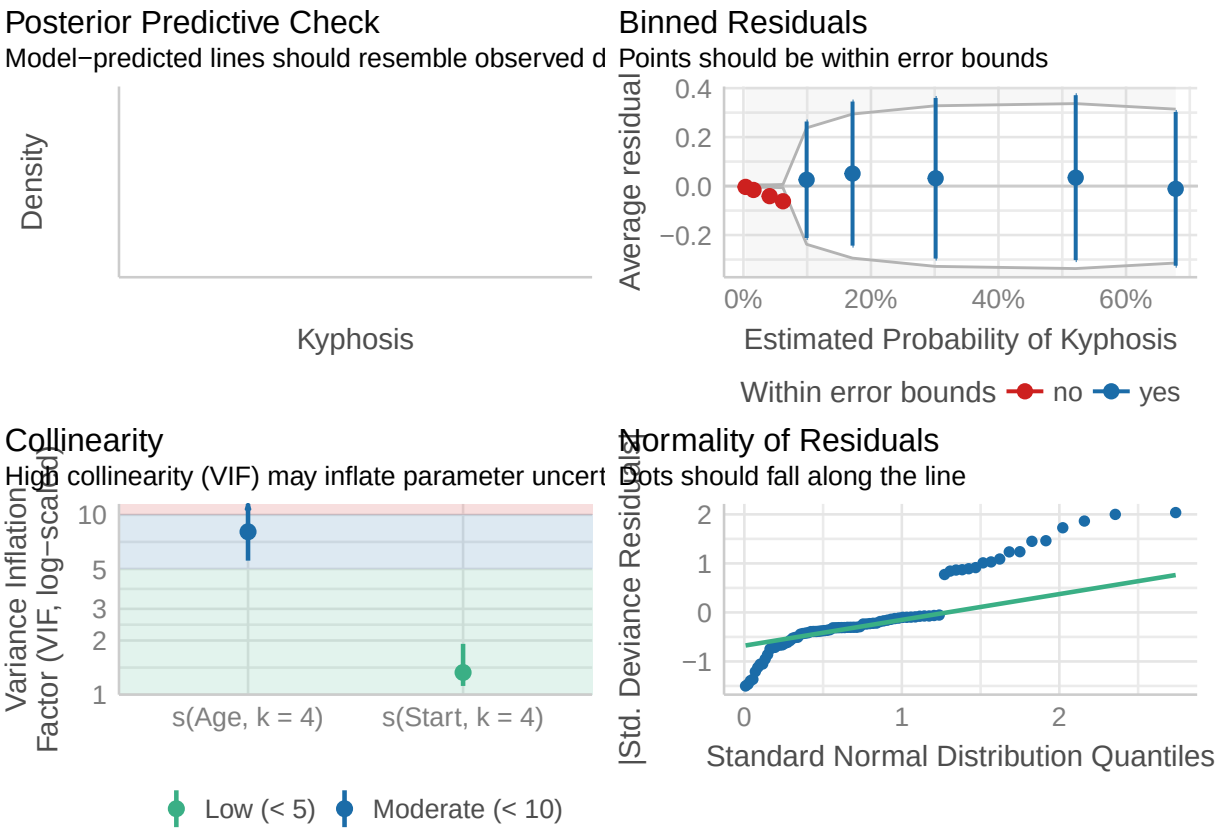


These pictures are not correct. They are in some sense inverted in the x axis and vertically translated compared to Hastie and Tibshirani [1990]. Cant figure this out.

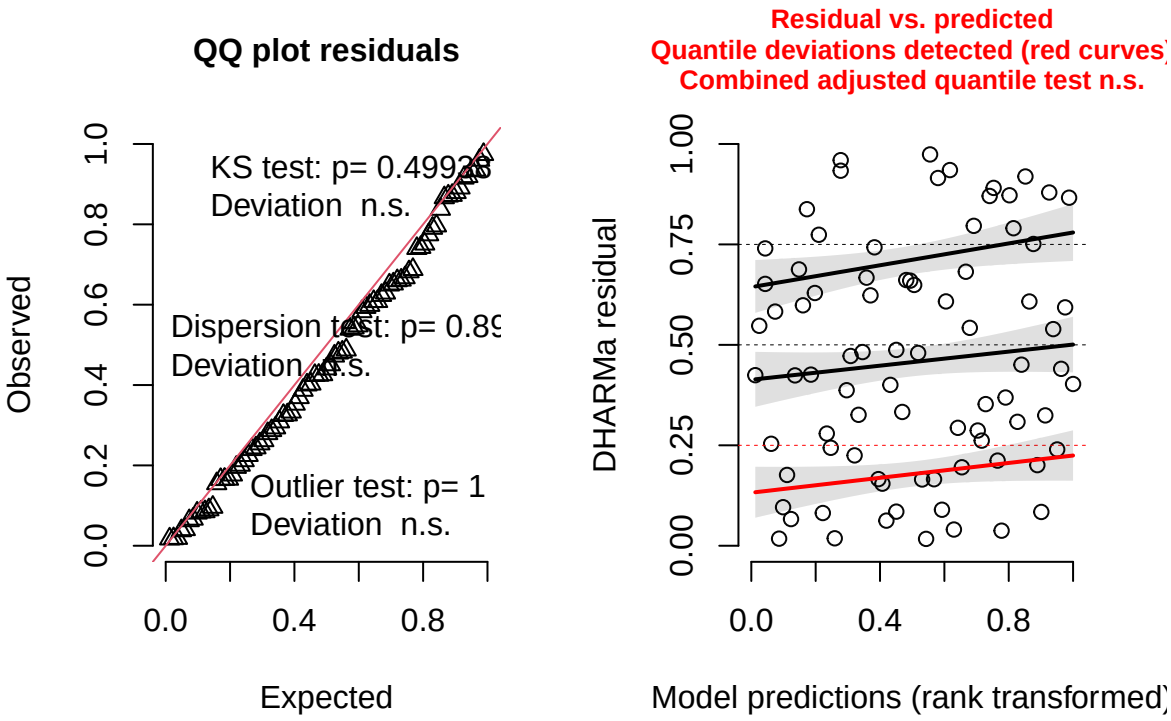
d

Since I couldn't get the parametric approximation model correct, we will work with the two term smooth model `fit_gam`.

```
# Diagnostic plots from performance
 check_model(fit_gam)
```




```
# Diagnostic plots from DHARMa
residuals <- simulateResiduals(fittedModel = fit_gam)
plot(residuals)
```



We can see the the residuals in the QQplot from `DHARMa` fit much better than the residuals in `performance::model_check`. Lets examine why that might be. First lets figure out what each package does differently with regards to residual plots.

Note: Skip ahead for solution, below is for my own pedagogical purposes.

For GLM(M)s the predictive distribution of the data changes with the fitted values. This means that that for the standard residual plot. the residuals may not appear homogeneously spread around zero across the entire fitted range and therefore lead to the misinterpretation of a mis-specified model. dHARMA addresses the issues concerning the correct interpretation of residuals in GL(M)Ms. The basic approach is to simulate new data from the fitted model for each observation. Them using the ECDF we see where the observed values fall relative to the simulated values. Recall the formula for the ECDF

$$F(x) = \frac{\text{number of simulated values} \leq x}{\text{total number of simulations}},$$

whose range is $(0, 1)$. The idea is that if the model is correct, the ECDF values for all your data points should be $\sim U(0, 1)$ because under the true model each observed point has an equal chance of being anywhere in the range of the simulated values. Lastly dHARMA uses a transformation step to ensure these ECDF values are uniformly distributed. Therefore the residuals always have the same distribution independent of the model that is fit.

Analysis starts here Examining the above figure on the right, which contains a pot of the residuals against the predicted variable, we see that the bottom red line indicates a significant deviation of the residuals (in the y - direction) from the expectation in the first quantile. The deviation is non significant for the 2nd and 3rd quantile. Furthermore the KS test, which tests if the simulated residuals are uniformly distributed, we fail to rejest the null hypothesis that the they are. The dispersion test and outlier test also do no detect significant deviation.

Now let us look at `performance::check_model()`. Looking at the documentation EasyStats Community [2023] for the performance package we see that `check_model()` use standardized Pearson's residuals for GLMs. it is picking up a possible false "misspecification" in the model.

What I believe might be happening here is the following. The natural division of `Start` into thoracic and lumbar vertebrae described above, as a piece wise curve consisting of a constant and a negative linear component divide at $Start = 12$. In section f below we see that the fitted prevalence of Age conditioned on a lumbar start value vs a thoracic star value classifies the data into two groups in the first figure. Perhaps this is why the `performance::check_model()` residuals plot appears divided into two groups. `dHARMA`, through its uniform residual transformation is able to correct this.

e

Above we have already visualized the effects of the smooth predictors. We saw the quadratic effect of `Age` and the approximately piece-wise linear effect of `Start` divided at $start = 12$.

Since we have smooth terms we cant use traditional coefficient plots. But we could give the estimated degrees of freedom. This gives us an idea of how flexible each term is. Recall the EDF reflects the effective number of parameters used to describe the smooth function.

```
data_to_plot <- data.frame(Term = c("s(Age)", "s(Start)"),
                           EDF = c(summary(fit_gam)$s.table[, "edf"]))
print(data_to_plot)
```

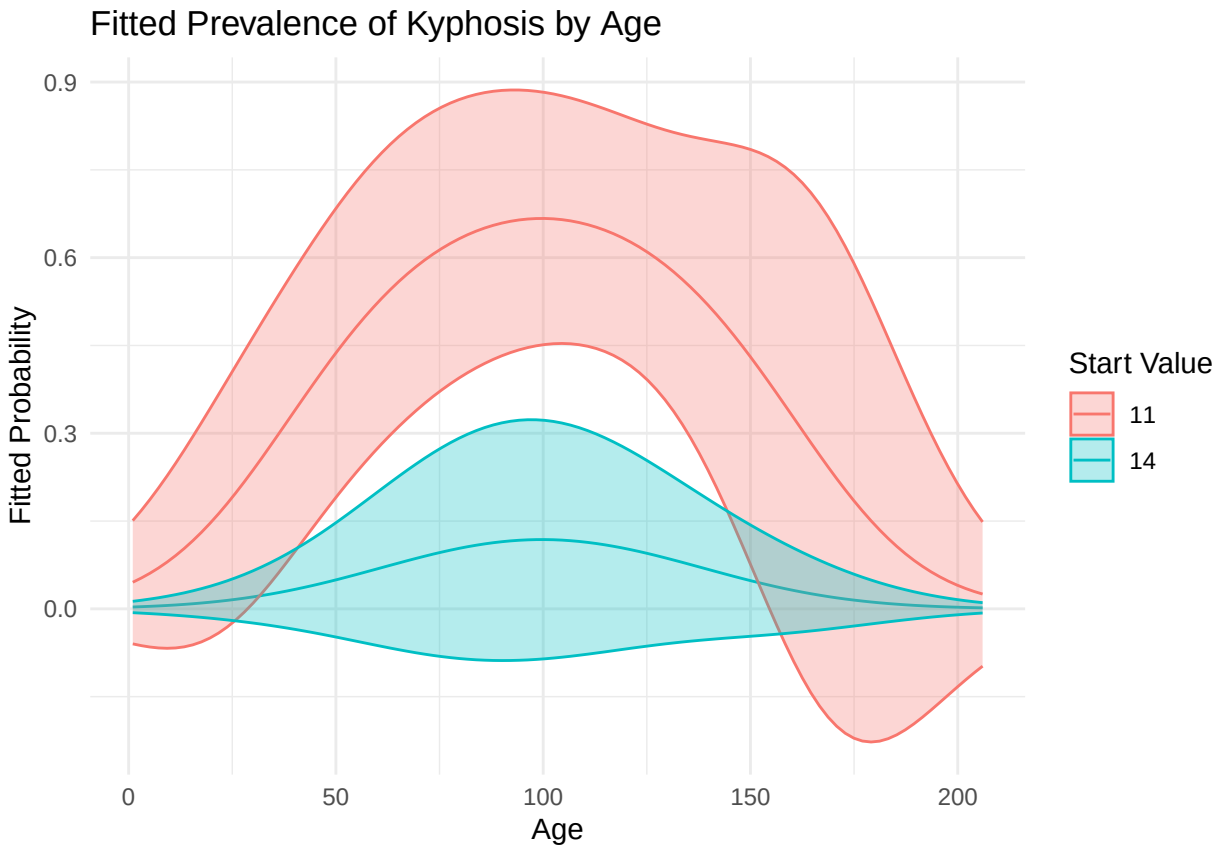
```
##               Term      EDF
## s(Age)     s(Age) 2.117754
## s(Start) s(Start) 2.018250
```

`s(Age)` has an EDF of 2.12 and `s(Start)` has an EDF of 2.01 which both suggests a not overly wiggly non-lineaer relationship.

f bonus fun

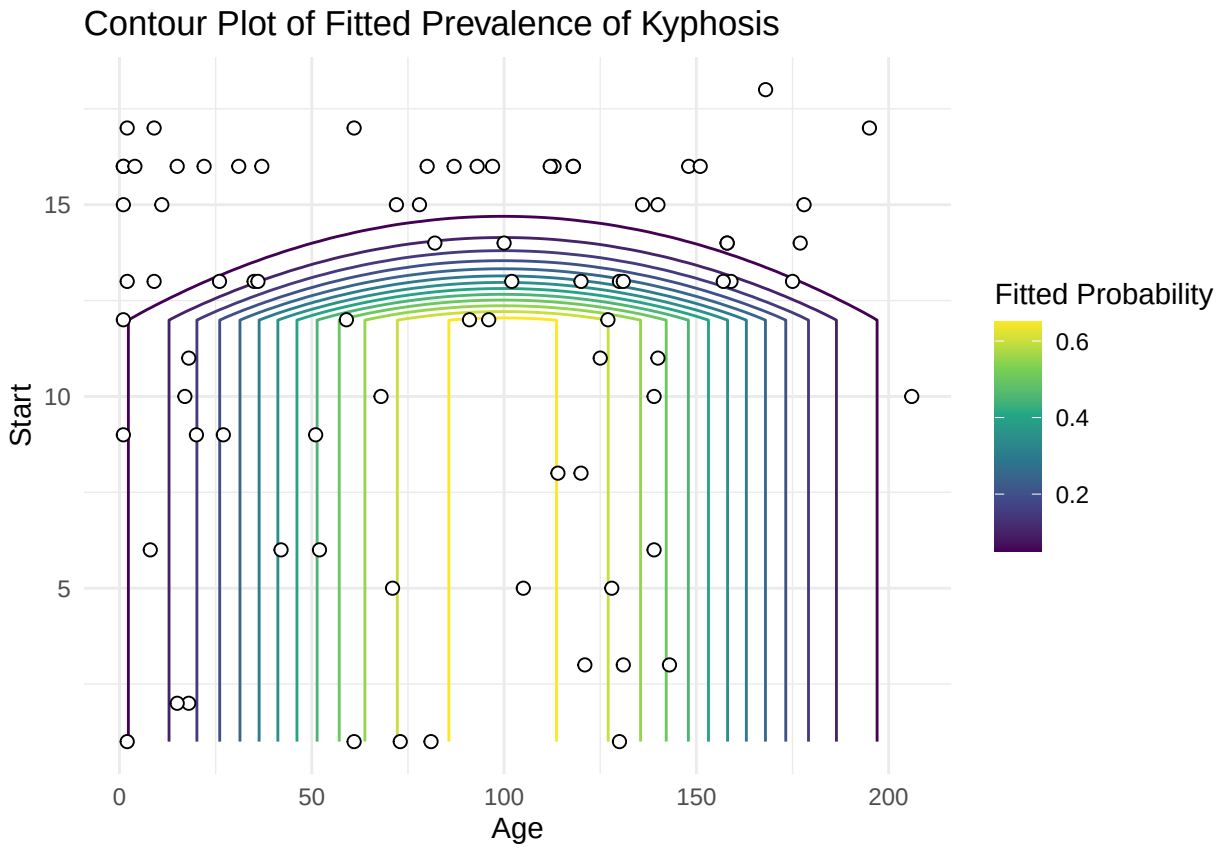
This is figure 10.7 from Hastie and Tibshirani [1990].

```
# Create new data for prediction
new_data <- data.frame(
  Age = rep(seq(min(kyphosis$Age), max(kyphosis$Age), length.out = 100), 2),
  Start = c(rep(11, 100), rep(14, 100))
)

# Add the start_transformed variable
new_data$start_transformed <- (new_data$Start - 12) *
  as.integer(new_data$Start > 12)



# Predict using the new data
new_data$fit <- predict(model_parametric, newdata = new_data, type = "response")
new_data$se <- predict(model_parametric, newdata = new_data, type = "response",
                       se.fit = TRUE)$se.fit

# Calculate the upper and lower bounds for the confidence interval
new_data$fit_upper <- with(new_data, fit + 1.96*se)
new_data$fit_lower <- with(new_data, fit - 1.96*se)

ggplot(new_data, aes(x = Age, y = fit, color = as.factor(Start))) +
  geom_line() +
  geom_ribbon(aes(ymin = fit_lower, ymax = fit_upper, fill = as.factor(Start)),
              alpha = 0.3) +
  labs(title = "Fitted Prevalence of Kyphosis by Age",
       y = "Fitted Probability", x = "Age",
       color = "Start Value", fill = "Start Value") +
  theme_minimal()
```




```r
# Generate a grid for prediction
age_range <- seq(min(kyphosis$Age), max(kyphosis$Age), length.out = 100)
start_range <- seq(min(kyphosis$Start), max(kyphosis$Start), length.out = 100)

grid_data <- expand.grid(Age = age_range, Start = start_range)
grid_data$start_transformed <- (grid_data$Start - 12) *
  as.integer(grid_data$Start > 12)
grid_data$fit <- predict(model_parametric, newdata =
                           grid_data, type = "response")

# Create contour plot
ggplot(grid_data, aes(x = Age, y = Start)) +
  geom_contour(aes(z = fit, color = after_stat(level))) +
  geom_point(data = kyphosis, aes(x = Age, y = Start), size = 2, shape = 21,
             fill = "white") +
  scale_color_viridis_c() +
  labs(title = "Contour Plot of Fitted Prevalence of Kyphosis",
       x = "Age", y = "Start",
```

```
       color = "Fitted Probability") +
  theme_minimal()
```



I tried to compute point wise standard error bands using the delta method on the fitted logits as they do in the text. I couldn't get it correct so that's why the above confidence bands are slightly off. Here is what I tried

```
# Create new data for prediction
new_data <- data.frame(
  Age = rep(seq(min(kyphosis$Age), max(kyphosis$Age), length.out = 100), 2),
  Start = c(rep(11, 100), rep(14, 100))
)

#Compute the fitted logits and their standard errors:
logit_predictions <- predict(fit_gam, newdata = new_data,
                             type = "link", se.fit = TRUE)
new_data$logit_fit <- logit_predictions$fit
new_data$logit_se <- logit_predictions$se.fit
```

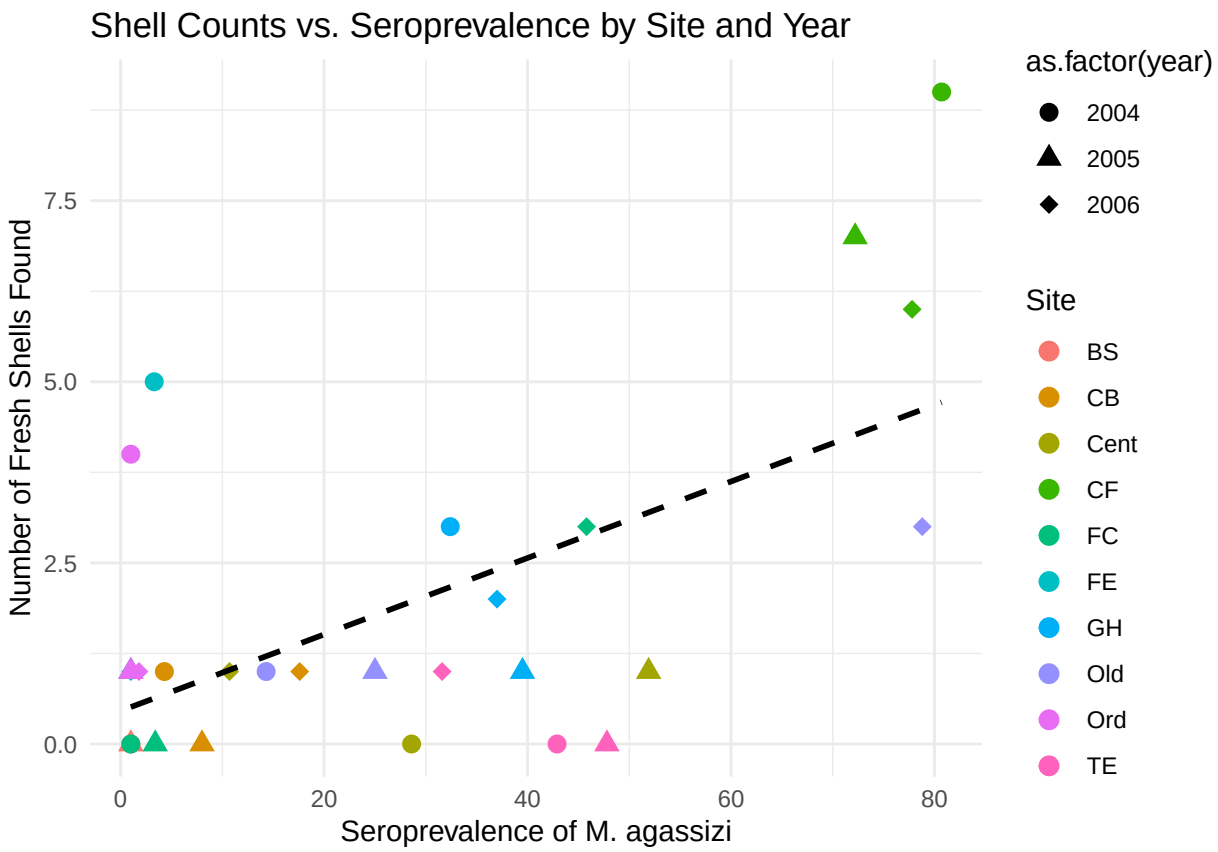
```r
# Compute  approximation of variance of logit using delta method and
# take square root.
p <- exp(new_data$logit_fit) / (1 + exp(new_data$logit_fit))
new_data$fit <- p
new_data$se <- sqrt((1 / (p * (1-p)))^2 * new_data$logit_se^2)
```

I couldn't get the plots to work and the variance ballooned at the right and left hand sides.

2

a

```r
#Visualize the relationship between the number of shells found and the
# seroprevalence for each site and year and overlay a regression line.

p <- ggplot(g_data, aes(x = prev, y = shells, color = Site)) +
  geom_point(aes(shape = as.factor(year)), size = 3) +
  geom_smooth(method = "lm", se = FALSE, color = "black", linetype = "dashed") +
  labs(title = "Shell Counts vs. Seroprevalence by Site and Year",
       x = "Seroprevalence of M. agassizi",
       y = "Number of Fresh Shells Found") +
  theme_minimal() +
  scale_shape_manual(values = c(16, 17, 18)) # Manual shape for different years

print(p)
```



b

Given that the response is count data lets first consider a poisson regression model. If over-dispersion is present we can then fit a negative binomial model.

```
# Fit poisson  regression model
fit1 <- glm(shells ~ year + prev + offset(log(Area)),
                     data = g_data, family = "poisson")
summary(fit1)
```

```
##
## Call:
## glm(formula = shells ~ year + prev + offset(log(Area)), family = "poisson",
##     data = g_data)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.19732    0.27850 -11.481  < 2e-16 ***
```

```
## year2005    -0.64486    0.35639  -1.809   0.0704 .
## year2006    -0.42876    0.31301  -1.370   0.1708
## prev         0.02113    0.00431   4.903 9.44e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for poisson family taken to be 1)
##
##     Null deviance: 49.800  on 29  degrees of freedom
## Residual deviance: 22.483  on 26  degrees of freedom
## AIC: 84.337
##
## Number of Fisher Scoring iterations: 5
```

```
# Calculate dispersion


dispersion <- fit1$deviance / fit1$df.residual
dispersion
```

```
## [1] 0.8647354
```

Since the dispersion is close to 1 lets fit the negative binomial model and compare.

```
# Fit Negative Binomial regression
nb_model <- glm.nb(shells ~ year + prev + offset(log(Area)), data = g_data,
                   control = glm.control(maxit = 100))
summary(nb_model)
```

```
##
## Call:
## glm.nb(formula = shells ~ year + prev + offset(log(Area)), data = g_data,
##     control = glm.control(maxit = 100), init.theta = 71709880450000,
##     link = log)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -3.19732    0.27850 -11.481  < 2e-16 ***
## year2005    -0.64486    0.35639  -1.809   0.0704 .
## year2006    -0.42876    0.31301  -1.370   0.1708
```

```
## prev         0.02113    0.00431   4.903 9.44e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for Negative Binomial(7.170988e+13) family taken to be 1)
##
##     Null deviance: 49.705  on 29  degrees of freedom
## Residual deviance: 22.515  on 26  degrees of freedom
## AIC: 85
##
## Number of Fisher Scoring iterations: 1
##
##
##               Theta:  7.170988e+13
##           Std. Err.:  6.436341e+13
## Warning while fitting theta: iteration limit reached
##
##  2 x log-likelihood:  -75
```

The coefficients, deviance and AIC between the two models are very similar. However the dispersion parameter θ is exceptionally large. This means that the model is approaching a Poisson distribution. Therefore the suspicion of over dispersion because is probably not warranted because of the close to 1 dispersion statistic from the poisson model and the results of the NB model.

c

```
fit2 <- mle2(shells ~ dpois(lambda = exp(logmu) * Area),
                 parameters = list(logmu ~ year + prev),
                 data = g_data,
                 start = list(logmu = 0))

summary(fit2)
```

```
## Maximum likelihood estimation
##
## Call:
## mle2(minuslogl = shells ~ dpois(lambda = exp(logmu) * Area),
```

```
##     start = list(logmu = 0), data = g_data, parameters = list(logmu ~
##         year + prev))
##
## Coefficients:
##                    Estimate Std. Error  z value     Pr(z)
## logmu.(Intercept) -3.195742   0.278362 -11.4805 < 2.2e-16 ***
## logmu.year2005    -0.644976   0.356389  -1.8098   0.07033 .
## logmu.year2006    -0.428402   0.313021  -1.3686   0.17112
## logmu.prev         0.021097   0.004310   4.8949 9.837e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## -2 log L: 76.33738
```

d

```r
X <- model.matrix(~ year + prev + offset(log(Area)), data = g_data)
nll_fit3 <- function(params, data) {
  # Given the structure of X, params should be unpacked into 4 beta coefficients
  b0 <- params[1]  # Intercept
  b1 <- params[2]  # Coefficient for year2005
  b2 <- params[3]  # Coefficient for year2006
  b3 <- params[4]  # Coefficient for prev

  beta <- c(b0, b1, b2, b3)

  mu <- exp(X %*% beta)

  # negative log-likelihood for Poisson
  neg_log_lik <- -sum(dpois(data$shells, lambda = mu, log = TRUE))
  return(neg_log_lik)
}

# Initial parameter values
init_params <- as.vector(fit1$coefficients)
```

```r
# Optimization
fit3 <- optim(par = init_params, fn = nll_fit3, data = g_data,
              hessian = TRUE)
names(fit3$par)<- colnames(X)
```

e

Compute Wald and profile CIs.

```r
# CIs for fit1
coefs <- coef(summary(fit1))

wald_CI_lower <- coefs[, 1] - 1.96 * coefs[, 2]
wald_CI_upper <- coefs[, 1] + 1.96 * coefs[, 2]

wald_fit1 <- data.frame(
  Estimate = coefs[, 1],
  `Wald CI Lower` = wald_CI_lower,
  `Wald CI Upper` = wald_CI_upper
)
profile_fit1 <- confint(fit1, level = 0.95)
```

```r
# CIs for fit2
coefs2 <- coef(summary(fit2))

wald_CI_lower2 <- coefs2[, 1] - 1.96 * coefs2[, 2]
wald_CI_upper2 <- coefs2[, 1] + 1.96 * coefs2[, 2]

wald_fit2 <- data.frame(
  Estimate = coefs2[, 1],
  `Wald CI Lower` = wald_CI_lower,
  `Wald CI Upper` = wald_CI_upper
)

profile_fit2 <- confint(fit2, level = 0.95)
```

Following Bolker [2008] for computing profile CI's from `optim()` fit.

```r
# CIs for fit3
v_cov <- solve(fit3$hessian)
std_errors <- sqrt(diag(v_cov))

conf_level <- 0.95
z_value <- qnorm((1 + conf_level)/2)
lower_bounds <- fit3$par - z_value * std_errors
upper_bounds <- fit3$par + z_value * std_errors



wald_fit3 <- data.frame(Estimate = fit3$par,
                        Lower = lower_bounds,
                        Upper = upper_bounds)

# profile ll for optim foll
intvec <- seq(-0.6,1, by = 0.014)
yearvec <- seq(-1, 1, by = 0.05)
prevvec <- seq(-2, 4, by = 0.05)




# initialize result matrices
int.profile = matrix(ncol = 2, nrow = length(intvec))
year2005.profile = matrix(ncol = 2, nrow = length(yearvec))
year2006.profile = matrix(ncol = 2, nrow = length(yearvec))
prev.profile = matrix(ncol = 2, nrow = length(prevvec))

# write ll as function of single parameters

nll_profile_intercept <- function(fixed_intercept, other_params, data) {
    b0 <- intvec[1]
    b1 <- fit3$par[2]
    b2 <- fit3$par[3]
    b3 <- fit3$par[4]
    beta <- c(b0, b1, b2, b3)
    mu <- exp(X %*% beta)
```

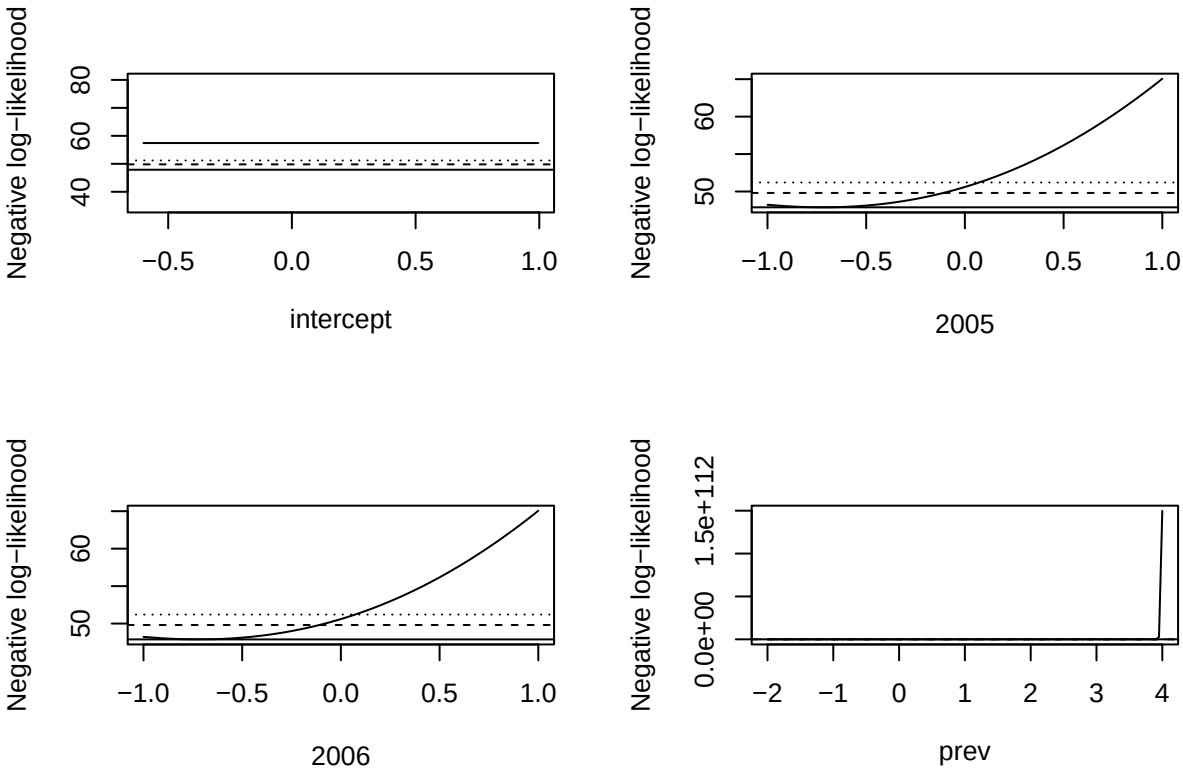
```r
  # negative log-likelihood for Poisson
  neg_log_lik <- -sum(dpois(data$shells, lambda = mu, log = TRUE))
  return(neg_log_lik)
}


nll_profile_year <- function(fixed_year, other_params, data) {
    b0 <- other_params[1]
    b2 <- fit3$par[3]
    b3 <- other_params[2]

    beta <- c(b0, fixed_year, b2, b3)
    mu <- exp(X %*% beta)

    # negative log-likelihood for Poisson
    neg_log_lik <- -sum(dpois(data$shells, lambda = mu, log = TRUE))
    return(neg_log_lik)
}


nll_profile_prev <- function(fixed_prev, other_params, data) {
    b0 <- other_params[1]
    b1 <- other_params[2]
    b2 <- fit3$par[3]
    b3 <- fixed_prev

    beta <- c(b0, b1, b2, b3)
    mu <- exp(X %*% beta)

    # negative log-likelihood for Poisson
    neg_log_lik <- -sum(dpois(data$shells, lambda = mu, log = TRUE))
    return(neg_log_lik)
}
```

```r
# compute  likelihood for fixed values of parameter varying over the other two.

for (i in 1:length(intvec)) {
  temp_fn <- function(x) nll_profile_intercept(intvec[i], x, g_data)

  Oval = optim(fn = temp_fn, par = fit3$par[2:4])

  int.profile[i, ] = c(intvec[i], Oval$value)
}
colnames(int.profile) = c("Intercept", "NLL")



for (i in 1:length(yearvec)) {
    temp_fn_year <- function(x) nll_profile_year(yearvec[i], x, g_data)

    Oval = optim(fn = temp_fn_year, par = c(fit3$par[1], fit3$par[3], fit3$par[4]))

    year2005.profile[i, ] = c(yearvec[i], Oval$value)
}
colnames(year2005.profile) = c("Year2005", "NLL")

for (i in 1:length(yearvec)) {

    temp_fn_year <- function(x) nll_profile_year(yearvec[i], x, g_data)

    Oval = optim(fn = temp_fn_year, par = c(fit3$par[1], fit3$par[2], fit3$par[4]))

    year2006.profile[i, ] = c(yearvec[i], Oval$value)
}
colnames(year2006.profile) = c("Year2006", "NLL")



for (i in 1:length(prevvec)) {
```

```
    temp_fn_prev <- function(x) nll_profile_prev(prevvec[i], x, g_data)

    Oval = optim(fn = temp_fn_prev, par = fit3$par[1:3])

    prev.profile[i, ] = c(prevvec[i], Oval$value)
}
colnames(prev.profile) = c("Prevalence", "NLL")
```

Sorry I spent too much time on this and need to move on.

```
par(mfrow = c(2,2))

plot(intvec, int.profile[, "NLL"], type = "l", xlab = expression(intercept),
 ylab = "Negative log-likelihood")
cutoffs = c(0, qchisq(c(0.95, 0.99), 1)/2)
 nll.levels = fit3$value + cutoffs
 abline(h = nll.levels, lty = 1:3)

 plot(yearvec, year2005.profile[, "NLL"], type = "l",
      xlab = expression(year=2005),
 ylab = "Negative log-likelihood")
 cutoffs = c(0, qchisq(c(0.95, 0.99), 1)/2)
 nll.levels = fit3$value + cutoffs
 abline(h = nll.levels, lty = 1:3)

plot(yearvec, year2006.profile[, "NLL"], type = "l",
     xlab = expression(year=2006),
 ylab = "Negative log-likelihood")
cutoffs = c(0, qchisq(c(0.95, 0.99), 1)/2)
 nll.levels = fit3$value + cutoffs
 abline(h = nll.levels, lty = 1:3)

plot(prevvec, prev.profile[, "NLL"], type = "l", xlab = expression(prev),
 ylab = "Negative log-likelihood")
cutoffs = c(0, qchisq(c(0.95, 0.99), 1)/2)
 nll.levels = fit3$value + cutoffs
 abline(h = nll.levels, lty = 1:3)
```



These seem funny looking especially `prevalence` and the `intercept`.

Now we compute the profile confidence intervals.

```
# Name and extract hessian and information matrix
colnames(fit3$hessian)<- colnames(X)
rownames(fit3$hessian) <- colnames(X)
s1 = solve(fit3$hessian)
a = fit3$value

# CI for intercept
b = fit3$par[4]
c= fit3$hessian["(Intercept)", "(Intercept)"]/2
se.int =  sqrt(s1["(Intercept)", "(Intercept)"])
ci.info.int = fit3$par["(Intercept)"]+ c(-1,1)*qnorm(0.975) *se.int

# CI for year2005
b = fit3$par[2]
c= fit3$hessian["year2005", "year2005"]/2
se.year2005 =  sqrt(s1["year2005", "year2005"])
ci.info.2005 = fit3$par["year2005"]+ c(-1,1)*qnorm(0.975) *se.year2005
```

```r
# CI for year 2006
b = fit3$par[3]
c= fit3$hessian["year2006", "year2006"]/2
se.year2006 =  sqrt(s1["year2006", "year2006"])
ci.info.2006 = fit3$par["year2006"]+ c(-1,1)*qnorm(0.975) *se.year2006

#CI for prev
b = fit3$par[4]
c= fit3$hessian["prev", "prev"]/2
se.prev =  sqrt(s1["prev", "prev"])
ci.info.prev = fit3$par["prev"]+ c(-1,1)*qnorm(0.975) *se.prev

# Create a data frame to compile all the data
profile_fit3 <- data.frame(
  `2.5 %` = c(ci.info.int[1], ci.info.2005[1], ci.info.2006[1],
              ci.info.prev[1]),
  `97.5 %` = c(ci.info.int[2], ci.info.2005[2], ci.info.2006[2],
               ci.info.prev[2])
)
```

Lets compare the 3 methods and 2 CI's

```r
print(wald_fit1)
```

```
##                Estimate Wald.CI.Lower Wald.CI.Upper
## (Intercept) -3.19731743   -3.74317472   -2.65146014
## year2005    -0.64485763   -1.34338123    0.05366597
## year2006    -0.42875881   -1.04226312    0.18474550
## prev         0.02113395    0.01268543    0.02958247
```

```r
print(wald_fit2)
```

```
##                      Estimate Wald.CI.Lower Wald.CI.Upper
## logmu.(Intercept) -3.19574221   -3.74317472   -2.65146014
## logmu.year2005    -0.64497549   -1.34338123    0.05366597
## logmu.year2006    -0.42840235   -1.04226312    0.18474550
## logmu.prev         0.02109688    0.01268543    0.02958247
```

```r
print(wald_fit3)
```

```
##                Estimate       Lower       Upper
## (Intercept)  0.05926294 -0.50024058  0.61876646
## year2005    -0.72024080 -1.41827756 -0.02220404
## year2006    -0.46035782 -1.07533405  0.15461842
## prev         0.02526048  0.01617485  0.03434610
```

```r
print(profile_fit1)
```

```
##                   2.5 %      97.5 %
## (Intercept) -3.78047101 -2.68513102
## year2005    -1.37646427  0.03521203
## year2006    -1.05249570  0.18368304
## prev         0.01272002  0.02969948
```

```r
print(profile_fit2)
```

```
##                          2.5 %      97.5 %
## logmu.(Intercept) -3.78051882 -2.68517873
## logmu.year2005    -1.37661027  0.03521198
## logmu.year2006    -1.05250341  0.18368802
## logmu.prev         0.01271875  0.02969852
```

```r
print(profile_fit3)
```

```
##        X2.5..     X97.5..
## 1 -0.50024058  0.61876646
## 2 -1.41827756 -0.02220404
## 3 -1.07533405  0.15461842
## 4  0.01617485  0.03434610
```

We can see that everything other than the `intercept` for `optim()` is nearly identical.

3

```r
# Fit model using glm()
fit_glm <- glm(HG ~ PI + EH + NV, family = binomial, data = endometrial)
```

```r
# Fit model using arm::bayesglm()
fit_bayesglm <- arm::bayesglm(HG ~ PI + EH + NV, family = binomial,
                              data = endometrial)

# Fit model using brglmFit
fit_brglm <- glm(HG ~ PI + EH + NV, family = binomial, data = endometrial,
                 method = "brglmFit")



glm_coefs <- coef(summary(fit_glm))
bayesglm_coefs <- coef(summary(fit_bayesglm))
brglm_coefs <- coef(summary(fit_brglm))

comparison_df <- data.frame(
  Term = rownames(glm_coefs),
  `GLM Estimate` = glm_coefs[, "Estimate"],
  `GLM Std. Error` = glm_coefs[, "Std. Error"],
  `GLM z value` = glm_coefs[, "z value"],
  `GLM Pr(>|z|)` = glm_coefs[, "Pr(>|z|)"],

  `BayesGLM Estimate` = bayesglm_coefs[, "Estimate"],
  `BayesGLM Std. Error` = bayesglm_coefs[, "Std. Error"],
  `BayesGLM z value` = bayesglm_coefs[, "z value"],
  `BayesGLM Pr(>|z|)` = bayesglm_coefs[, "Pr(>|z|)"],

  `BRGLM Estimate` = brglm_coefs[, "Estimate"],
  `BRGLM Std. Error` = brglm_coefs[, "Std. Error"],
  `BRGLM z value` = brglm_coefs[, "z value"],
  `BRGLM Pr(>|z|)` = brglm_coefs[, "Pr(>|z|)"]
)
```

	(Intercept)	PI	EH	NV
GLM.Estimate	4.3045178	-0.0421834	-2.9026056	18.1855558
GLM.Std..Error	1.637299e+00	4.433196e-02	8.455516e-01	1.715751e+03
GLM.z.value	2.62903649	-0.95153474	-3.43279555	0.01059918
GLM.Pr…z..	0.0085627172	0.3413329906	0.0005973924	0.9915432347
BayesGLM.Estimate	3.71159448	-0.02903616	-2.62864678	3.29815553
BayesGLM.Std..Error	1.38953838	0.03657463	0.73865115	1.58067797
BayesGLM.z.value	2.6710989	-0.7938881	-3.5587121	2.0865449
BayesGLM.Pr…z..	0.0075603362	0.4272605796	0.0003726778	0.0369292945
BRGLM.Estimate	3.77455909	-0.03475174	-2.60416372	2.92927316
BRGLM.Std..Error	1.48869154	0.03957814	0.77601760	1.55076365
BRGLM.z.value	2.5354877	-0.8780538	-3.3558050	1.8889230
BRGLM.Pr…z..	0.0112290873	0.3799145349	0.0007913435	0.0589021443

In Heinze and Schemper [2002], they state that the in endometrial data set, there is no observation for which $NV = 1$ and $HG = 0$. This means that one combination of predictor variables (with $NV = 1$) corresponds to mostly one outcome ($HG \neq 1$). This is called quasi-complete separation. This leads to a problem: the maximum likelihood estimate (MLE) will keep increasing without bound to try to make the predicted probability of the outcome (`HG` being 1 for $NV = 1$) as close to 1 as possible. This is why the estimate of the effect of NV becomes infinite or very large for `glm()` with `method = "IWLS"`. Instead we can modify the score equation $ log L / {r} U({r}) = 0$. Firth suggested basing estimation on a modified score equation

$$U(\theta_r)^* \equiv U(\theta_{r+1}) = \frac{1}{2}\text{trace}\left[I(\theta)^{-1}\{\partial I(\theta) = \partial\theta_r\}\right] = 0 \quad (\text{for } r = 1, \ldots, k),$$

This modified score function is connected to the penalized likelihood. The penalized likelihood is essentially the original $L(\beta)$ plus a factor $\mid I(\beta) \mid^{\frac{1}{2}}$ based on the information matrix $I(\beta)$ refereed to as Jeffery's invariant prior. Jeffreys prior is influential for smaller sample sized and cases with separation but its influence diminishes as the sample size grows StackExchange. Firth demonstrated that the bias, which is proportional to $\mathcal{O}(n^{-1})$, is removed from the ML estimate.

If instead we instead set `method = brglmFit"` for `stats::glm()`, then this utilizes Firths penalty of half the trace of a particular matrix product, to adjust the parameters estimates in the case of our quasi-complete seperation. Similarly `arm::bayesglm()` uses a non-informative prior assumption to regularize the coefficients and pull them slightly towards zero. Gelman \cite{{gelman2008weakly} recommends to put a cauchy prior with median 0 and scale 2.5 on each coefficient. This is how `arm::bayesglm()` solves the seperation problem.

Therefore in the above data we see that `glm` with the default method estimates `NV` 18.18 with standard error $1.71e^3$ is extremely large compared to the other two methods. The other parameters are very similar.

No we computeLRT's for each parameter for the two methods of glm().

```
# For fit_glm:

# LRT for PI:
fit_glm_reduced_PI <- glm(HG ~ EH + NV, family = binomial, data = endometrial)
anova_glm_PI <- anova(fit_glm_reduced_PI, fit_glm, test = "LRT")

# LRT for EH:
fit_glm_reduced_EH <- glm(HG ~ PI + NV, family = binomial, data = endometrial)
anova_glm_EH <- anova(fit_glm_reduced_EH, fit_glm, test = "LRT")

# LRT for NV:
fit_glm_reduced_NV <- glm(HG ~ PI + EH, family = binomial, data = endometrial)
anova_glm_NV <- anova(fit_glm_reduced_NV, fit_glm, test = "LRT")



# For fit_brglm:

# LRT for PI:
fit_brglm_reduced_PI <- glm(HG ~ EH + NV, family = binomial, data = endometrial,
                            method = "brglmFit")
anova_brglm_PI <- anova(fit_brglm_reduced_PI, fit_brglm, test = "LRT")

# LRT for EH:
fit_brglm_reduced_EH <- glm(HG ~ PI + NV, family = binomial, data = endometrial,
                            method = "brglmFit")
anova_brglm_EH <- anova(fit_brglm_reduced_EH, fit_brglm, test = "LRT")

# LRT for NV:
fit_brglm_reduced_NV <- glm(HG ~ PI + EH, family = binomial, data = endometrial,
                            method = "brglmFit")
anova_brglm_NV <- anova(fit_brglm_reduced_NV, fit_brglm, test = "LRT")
```

```r
# You can then check the results using:
print(anova_glm_PI)
```

```
## Analysis of Deviance Table
##
## Model 1: HG ~ EH + NV
## Model 2: HG ~ PI + EH + NV
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1        76     56.378
## 2        75     55.393  1  0.98513   0.3209
```

```r
print(anova_glm_EH)
```

```
## Analysis of Deviance Table
##
## Model 1: HG ~ PI + NV
## Model 2: HG ~ PI + EH + NV
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1        76     75.154
## 2        75     55.393  1   19.761 8.777e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
print(anova_glm_NV)
```

```
## Analysis of Deviance Table
##
## Model 1: HG ~ PI + EH
## Model 2: HG ~ PI + EH + NV
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1        76     64.751
## 2        75     55.393  1   9.3576 0.002221 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
print(anova_brglm_PI)
```

```
## Analysis of Deviance Table
##
## Model 1: HG ~ EH + NV
```

```
## Model 2: HG ~ PI + EH + NV
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1        76     57.394
## 2        75     56.575  1  0.81898   0.3655
```

```
print(anova_brglm_EH)
```

```
## Analysis of Deviance Table
##
## Model 1: HG ~ PI + NV
## Model 2: HG ~ PI + EH + NV
##   Resid. Df Resid. Dev Df Deviance  Pr(>Chi)
## 1        76     76.206
## 2        75     56.575  1   19.631 9.394e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
print(anova_brglm_NV)
```

```
## Analysis of Deviance Table
##
## Model 1: HG ~ PI + EH
## Model 2: HG ~ PI + EH + NV
##   Resid. Df Resid. Dev Df Deviance Pr(>Chi)
## 1        76     64.888
## 2        75     56.575  1   8.3126 0.003937 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

For `NV` in the default glm fit, the significant p-value suggests that adding `NV` to the model significantly improves the fit. Given the quasi-complete separation concerning `NV`, this suggests that the fit_glm model might be attempting to fit an extreme coefficient for NV. This extreme coefficient would yield a significant improvement in the model fit when added, hence the significant p-value.

For `NV` in the bias-reduced glm fit, the significance of `NV` remains in the bias-reduced GLM, but the difference in deviance is slightly less than in the standard GLM. This is likely because the bias reduction methods attempt to mitigate the extreme estimates caused by separation, leading to a more moderated coefficient estimate.

The LRT indicates that adding `NV` significantly improves the model, which might be partly attributable to

the separation problem.

References

Benjamin M. Bolker. *Ecological Models and Data in R*. Princeton University Press, Princeton, NJ, 2008. URL https://doi.org/10.2307/j.ctvcm4g37. Accessed 16 Oct. 2023.

EasyStats Community. Check model - performance, 2023. URL https://easystats.github.io/performance/reference/check_model.html.

Trevor Hastie and Robert Tibshirani. *Generalized additive models*. Wiley Online Library, 1990.

Georg Heinze and Michael Schemper. A solution to the problem of separation in logistic regression. *Statistics in Medicine*, 21, 2002. URL https://api.semanticscholar.org/CorpusID:9412385.

StackExchange. How to deal with perfect separation in logistic regression, 2011. URL https://stats.stackexchange.com/questions/11109/how-to-deal-with-perfect-separation-in-logistic-regression.