Code to accompany paper/pre-print: Detecting changes in population trends in infection surveillance using community SARS-CoV-2 as an examplar

This document contains key pieces of code used to implement methods described in the paper/pre-print titled “Detecting changes in population trends in infection surveillance using community SARS-CoV-2 as an exemplar”. We used R version 4.0.2 for all analysis. The code is broken down into the following stages:

* Summary of dataset
* Fitting Generalised Additive Models (GAMs) and calculating second derivatives
* Fitting and predicting from Iterative Sequential Regression (ISR)

# Summary of dataset

The data used in this example was from the UK’s COVID-19 Infection Survey. This dataset includes swab results from randomly sampled participants across the UK. Data for this paper was restricted to between 1st August 2020 and 30th June 2022.

# Fitting GAMs & calculating second derivatives

## Fitting GAMs

Before running a GAM, we first had to choose an appropriate number of basis functions, *k*. We selected from 25, 50, 75, 100 as the lowest value with predicted positivity within ±0.25% compared with *k* = 100, optimising computational time, without large increases in the effective degrees of freedom (code not shown).

We chose *k* = 50 and penalised the splines based on the third derivative (*m* = 3) as the second derivative was the measure of interest. We fitted the GAM as follows:

# read in clean dataset   
load(file = paste0(Current\_run\_directory,"/growth\_rates\_fullpop.Rdata"))  
# subset data to region of interest   
dataset <- subset(sample, region == "LD")  
  
# run model   
k <- 50   
m <- 3   
gam <- gam(result ~ s(study\_day, k = k, m = m, bs = "tp"),   
 data = dataset,   
 family = nb(link = "log"),   
 method = "REML")

We next predicted the percentage testing positive per day from the model. We wrote a short function, *gam\_predictions*, as to avoid repetitive code when predicting from multiple models:

# create values to predict over   
pred\_data <- data.frame(study\_day = (seq(min(dataset$study\_day), max(dataset$study\_day), by = 1)))  
  
# function to predict data (to avoid code repetition)   
gam\_predictions <- function(object, newdata) {   
 prediction <- data.frame(predict.gam(gam, newdata = pred\_data, type = "response", se = T))   
 prediction <- transform(prediction,  
 pred\_lower = (fit - (2\*se.fit))\*100,   
 pred\_upper = (fit + (2\*se.fit))\*100,   
 pred = fit\*100,   
 se = se.fit)   
 prediction <- dplyr::select(prediction, pred, pred\_lower, pred\_upper, se)   
 prediction <- cbind(newdata, prediction)   
}   
  
# predict from gam   
predictions <- gam\_predictions(gam, newdata = pred\_data)   
predictions$k <- k   
predictions$m <- m

## Calculating derivatives

We next calculated second derivatives on the response scale with simultaneous confidence intervals and a Metropolis Hasting sampler. Details of why we chose this method are documented in the **Supplementary Methods** of the paper. The following code was adapted from the *gratia* package1 (<https://gavinsimpson.github.io/gratia>).

# function to calculate derivatives (adapted from gratia)   
  
derivatives\_mh <- function(object, newdata, n = 200, h1 = 1e-7,   
 h2 = 1e-7, startpoint = 0,   
 type = c("forward", "backward", "central")) {   
   
 # check correct type provided to function - forward, backward, or central   
 if (!type %in% c("forward", "backward", "central")) {   
 stop("Type must be either forward, backward, or central")  
 }  
   
 # posterior simulation with gam fit   
 linv <- object$family$linkinv   
 set.seed(2929)  
 br <- gam.mh(object, thin = 2, ns = 2000, rw.scale = 0.4)$bs   
   
 ## calculate first derivative ##   
   
 # create datasets for 1st derivatives   
 newdata\_subh <- newdata - h1   
 newdata\_addh <- newdata + h1   
   
 # linear predictors for model   
 X0 <- predict(object, type = "lpmatrix", newdata = newdata)   
 X\_subh <- predict(object, type = "lpmatrix", newdata = newdata\_subh)   
 X\_addh <- predict(object, type = "lpmatrix", newdata = newdata\_addh)   
   
 # simulate deviations from expected   
 ff0 <- linv(X0 %\*% t(br))  
 ff\_subh <- linv(X\_subh %\*% t(br))  
 ff\_addh <- linv(X\_addh %\*% t(br))  
   
 # calculate derivative  
 firstdiv <- switch(type,   
 forward = (ff\_addh - ff0) / h1,   
 backward = (ff0 - ff\_subh) / h1,   
 central = (ff\_addh - ff\_subh) / h1)   
   
 ffquant <- apply(firstdiv, 1, quantile, probs = c(0.025, 0.5, 0.975))   
   
 plot\_data <- newdata   
 plot\_data$first\_derivative <- ffquant[2,]  
 plot\_data$first\_lower <- ffquant[1,]  
 plot\_data$first\_upper <- ffquant[3,]  
  
 ## calculate second derivative ##   
   
 # create dataset for second derivative   
 newdata\_subh <- newdata - h2   
 newdata\_addh <- newdata + h2   
   
 # linear predictors for model   
 X0 <- predict(object, type = "lpmatrix", newdata = newdata)   
 X\_subh <- predict(object, type = "lpmatrix", newdata = newdata\_subh)   
 X\_addh <- predict(object, type = "lpmatrix", newdata = newdata\_addh)   
   
 # simulate deviations from expected   
 ff0 <- linv(X0 %\*% t(br))  
 ff\_subh <- linv(X\_subh %\*% t(br))  
 ff\_addh <- linv(X\_addh %\*% t(br))  
   
 newdata\_add2h <- newdata + (2 \* h2)   
 X\_add2h <- predict(object, type = "lpmatrix", newdata = newdata\_add2h)   
 ff\_add2h <- linv(X\_add2h %\*% t(br))   
   
 newdata\_sub2h <- newdata - (2 \* h2)  
 X\_sub2h <- predict(object, type = "lpmatrix", newdata = newdata\_sub2h)   
 ff\_sub2h <- linv(X\_sub2h %\*% t(br))   
   
 # calculate derivativr  
 seconddiv <- switch(type,   
 forward = (ff\_add2h - (2 \* ff\_addh) + ff0) / (h2^2),   
 backward = (ff0 - (2 \* ff\_subh) + ff\_sub2h) / (h2^2),   
 central = (ff\_addh - (2 \* ff0) + ff\_subh) / (h2^2))   
   
 sdquant <- apply(seconddiv, 1, quantile, probs = c(0.025, 0.5, 0.975))   
   
 plot\_data$second\_derivative <- sdquant[2,]   
 plot\_data$second\_lower <- sdquant[1,]   
 plot\_data$second\_upper <- sdquant[3,]  
  
 ## calculate change-points based on second derivatives ##   
 plot\_data$second\_derivative\_signif <- 1   
 plot\_data$second\_derivative\_signif[plot\_data$second\_lower <= 0 & plot\_data$second\_upper >= 0] <- 0  
 plot\_data$derivative\_breakpoint <- 0   
 plot\_data$derivative\_breakpoint[plot\_data$second\_derivative\_signif == 1 &   
 lag(plot\_data$second\_derivative\_signif == 0, 1)] <- 1  
 plot\_data$derivative\_breakpoint[1:nrow(plot\_data) < startpoint] <- 0   
   
 # calculate direction of change-point   
 plot\_data <- plot\_data %>%   
 dplyr::mutate(lag\_upper = dplyr::lag(second\_upper, n = 1, default = NA),   
 lag\_lower = dplyr::lag(second\_lower, n = 1, default = NA))  
 plot\_data$direction <- NA   
   
 plot\_data$direction[plot\_data$derivative\_breakpoint == 1 &   
 plot\_data$second\_lower > 0 &   
 plot\_data$lag\_lower < 0] <- "increasing"  
   
 plot\_data$direction[plot\_data$derivative\_breakpoint == 1 &   
 plot\_data$second\_upper < 0 &   
 plot\_data$lag\_upper > 0] <- "decreasing"  
   
 plot\_data <- plot\_data %>% dplyr::select(!c(lag\_upper, lag\_lower))  
   
 # return output   
 return(plot\_data)  
   
}  
  
# calculate derivative using new function   
derivatives <- derivatives\_mh(gam, type = "backward", newdata = pred\_data,   
 h1 = 0.001, h2 = 0.1, startpoint = 31)  
  
# merge predicted values with derivatives   
combined <- merge(predictions, derivatives, by = "study\_day")

# Fitting and predicting from ISR

The function to run ISR was first developed in the manuscript by Schlackow et al.2. To access the function, first download the code from <https://github.com/ischlackow/ISR-iterative-sequential-regression-> and install the package contained inside .tar.gz using the following command:

install.packages("/.../isr\_1.0.tar.gz", repos = NULL, type = "source")

To update the functions inside this package as used in the paper, download the R script titled *librariesupdatingpackagefunction\_2020.R* on the following link: <https://github.com/karinadorisvihta/ISR-iterative-sequential-regression->. Run this R script prior to running functions within the Iterative Sequential Regression package.

Before running ISR, the dataset was formatted to be compatible with the ISR function. Dates were required to be in the *as.POSIXct* format.

# set-up dataset   
dataset$study\_day <- dataset$study\_day - min(dataset$study\_day)  
dataset$date <- dataset$study\_day + min(dataset$visit\_date)  
dataset$date <- as.POSIXct(as.character(dataset$date))  
dataset <- dplyr::select(dataset, participant\_id, result, study\_day, date)

We then set up empty data frames to capture the model output:

pred\_data\_range <- data.frame(

study\_day = (seq(min(sample$study\_day), max(sample$study\_day), by = 1)),

date = (seq(mindate, maxdate, by = 1)))   
  
# set up data frames to catch output   
isr\_output <- data.frame(study\_day = double(),   
 date = character(),   
 pred\_lower\_isr = double(),   
 pred\_upper\_isr = double(),   
 pred\_isr = double(),   
 se\_isr = double(),   
 min\_distance = double(),   
 interval\_length = double(),  
 region = character,   
 runtime\_isr = double())   
  
isr\_output\_summary <- data.frame(breakpoint = character(),  
 left.CI = character(),   
 right.CI = character(),   
 when.noticed = character(),   
 angle = double(),   
 se = double(),   
 angle.left.se = double(),   
 angle.right.se = double(),  
 angle.left.CI = double(),   
 angle.right.CI = double(),   
 pvalue = double(),  
 model = character(),   
 region = character)

We then defined all the parameters required for the model, and lastly ran the model using the *master.isr* function from the ISR package downloaded earlier.

resp.variable <- "result"  
resp.variable.label <- "Positivity"  
step <- FALSE  
t.boxcox <- FALSE  
factor.expl.variables <- ""  
cts.expl.variables <- ""  
offset.variable <- ""  
segm.variable <- "date"  
segm.variable.label <- "Visit date"  
model.type<-"glm.nb"  
model.parameters<-""  
angle.param <- 1  
start.point <- min(dataset[[segm.variable]]) + 31\*24\*60\*60  
end.point <- max(dataset[[segm.variable]])  
time.unit <- "days"  
criterion <- "aic"  
criterion.difference <- 6.635   
min.distance <- 7  
interval.length <- 3   
  
# run model   
start\_time <- Sys.time()  
isr <- master.isr(dataset=dataset,   
 resp.variable = resp.variable,   
 resp.variable.label = resp.variable.label,   
 step = step,   
 t.boxcox = t.boxcox,   
 factor.expl.variables = factor.expl.variables,   
 cts.expl.variables = cts.expl.variables,   
 offset.variable = offset.variable,  
 segm.variable = segm.variable,   
 segm.variable.label = segm.variable.label,   
 model.type = model.type,   
 model.parameters = model.parameters,   
 angle.param = angle.param,  
 start.point = start.point,   
 end.point = end.point,   
 time.unit = time.unit,   
 interval.length = interval.length,   
 min.distance = min.distance,  
 criterion = criterion,   
 criterion.difference = criterion.difference)  
end\_time <- Sys.time()  
saveRDS(isr, paste0(Current\_run\_directory,"/ISR\_md",min.distance,"\_il",interval.length,"\_",regions[r],".RDS"))

We next predicted the proportion testing positive on each visit date from the fitted ISR model. We wanted to predict for each visit date, even if raw data was unavailable on a particular day, for example Christmas day. This allowed us to get a smooth, connected estimate over the full time-series. This required creating a prediction dataset as follows:

# create prediction dataset   
  
# breakpoints   
summary\_isr <- isr$output$summary.dataset  
summary\_isr <- summary\_isr[2:nrow(summary\_isr),]   
summary\_isr$min\_distance <- min.distance   
summary\_isr$interval\_length <- interval.length  
   
#################################################################################  
   
# create list of breakpoints   
breakpoints <- as.Date(summary\_isr$breakpoint)  
  
# range of dates to predict over   
pred\_data <- data.frame(study\_day = (seq(min(sample$study\_day), max(sample$study\_day), by = 1)),   
 date = (seq(mindate, maxdate, by = 1)))   
  
# first column - date format   
pred\_data$char.date.0 <- as.POSIXct.Date(ifelse(pred\_data$date <= breakpoints[1],   
 seq(mindate, breakpoints[1], 1),   
 breakpoints[1]))  
# loop of breakpoints 2:n-1  
for (i in 1:(length(breakpoints)-1)) {   
 num <- as.numeric(which(pred\_data$date == breakpoints[i]))  
 num2 <- as.numeric(which(pred\_data$date == breakpoints[i+1]))  
 count <- num2 - num   
   
 assign(paste0("char.date.",i),   
 as.difftime(ifelse(pred\_data$date <= breakpoints[i], 0,   
 ifelse(pred\_data$date > breakpoints[i] & pred\_data$date < breakpoints[i+1],   
 (seq\_along(1:nrow(pred\_data)) - num) \* 86400,   
 rep(86400\*count))), units = "secs"))  
   
 pred\_data <- cbind(pred\_data, get(paste0("char.date.",i)))   
 colnames(pred\_data)[i+3] <- paste("char.date.",i, sep = "")  
   
}  
   
# for the final breakpoint, count until the end of the dataframe   
num <- as.numeric(which(pred\_data$date == breakpoints[length(breakpoints)]))  
assign(paste0("char.date.",length(breakpoints)),   
 as.difftime(ifelse(pred\_data$date <= breakpoints[length(breakpoints)], 0,   
 (seq\_along(1:nrow(pred\_data)) - num) \* 86400), units = "secs"))  
pred\_data <- cbind(pred\_data, get(paste0("char.date.",length(breakpoints))))   
colnames(pred\_data)[length(breakpoints)+3] <- paste("char.date.",length(breakpoints), sep = "")  
  
#################################################################################  
  
# predict   
predictions <- data.frame(predict(isr$output$best.model, pred\_data, type = "response", se = T))  
predictions <- cbind(pred\_data[c(1,2)], predictions)  
predictions <- transform(predictions,  
 pred\_lower\_isr = (fit - (2\*se.fit))\*100,   
 pred\_upper\_isr = (fit + (2\*se.fit))\*100,   
 pred\_isr = fit\*100,   
 se\_isr = se.fit)   
predictions <- dplyr::select(predictions, "study\_day", "date", "pred\_lower\_isr", "pred\_upper\_isr", "pred\_isr", "se\_isr")  
predictions$min\_distance <- min.distance   
predictions$interval\_length <- interval.length  
predictions$region <- regions[r]   
predictions$runtime\_isr <- round(as.numeric(difftime(end\_time, start\_time, units = "mins")), digits = 1)   
write.csv(predictions, paste0(Current\_run\_directory,"GrowthRates\_ISRPredictions\_md",min.distance,"\_il",interval.length,"\_",regions[[r]],".csv"), row.names = FALSE)   
  
# Extract breakpoints   
summary\_isr$model <- paste0("ISR (MD = ",min.distance,", IL = ",interval.length,")")   
summary\_isr$region <- regions[r]   
isr\_output\_summary <- rbind(isr\_output\_summary, summary\_isr)   
write.csv(isr\_output\_summary, paste0(Current\_run\_directory,"GrowthRates\_ISRBreakpoints\_md",min.distance,"\_il",interval.length,"\_",regions[[r]],".csv"), row.names = FALSE)

# Plotting and summarising findings

We combined estimates for both the GAMs and ISR models so we could compare estimates and change-points.

k <- 50   
m <- 3   
min.distance <- 7  
interval.length <- 3   
  
gam\_output <- read.csv(paste0(Current\_run\_directory,"GrowthRates\_GAMPredictions\_k",k,"\_m",m,"\_LD.csv"))  
gam\_output <- gam\_output[,-1]  
gam\_output$date <- gam\_output$study\_day + mindate   
isr\_output <- read.csv(paste0(Current\_run\_directory,"GrowthRates\_ISRPredictions\_md",min.distance,"\_il",interval.length,"\_LD.csv"))   
  
# combine estimates   
output <- merge(gam\_output, isr\_output, by = c("study\_day", "region", "date"), all = T)  
  
# Add in breakpoints   
isr\_output\_summary <- read.csv(paste0(Current\_run\_directory,"GrowthRates\_ISRBreakpoints\_md",min.distance,"\_il",interval.length,"\_LD.csv"))   
isr\_output\_summary$isr\_breakpoint <- as.Date(isr\_output\_summary$breakpoint)   
isr\_output\_summary$isr\_breakpoint\_lower <- as.Date(isr\_output\_summary$left.CI)   
isr\_output\_summary$isr\_breakpoint\_upper <- as.Date(isr\_output\_summary$right.CI)   
isr\_output\_summary$isr\_breakpoint\_whennoticed <- as.Date(isr\_output\_summary$when.noticed)   
isr\_output\_summary <- dplyr::rename(isr\_output\_summary, angle\_se = se)   
isr\_output\_summary <- dplyr::select(isr\_output\_summary, -c(breakpoint, left.CI, right.CI, when.noticed))  
isr\_output\_summary$study\_day <- as.numeric(as.Date(isr\_output\_summary$isr\_breakpoint) - mindate)  
  
output <- merge(output, isr\_output\_summary, by = c("study\_day", "region"), all = T)

We could then plot and compare GAMs with ISR models:

plotdata <- subset(output, region == "LD")  
  
p\_gamisr <- ggplot(plotdata, aes(x = date, y = pred)) +   
 geom\_rect(aes(xmin = as.Date("2020-11-05") , xmax = as.Date("2020-12-20"),   
 ymin = -Inf, ymax = Inf),   
 color = NA, fill = "gray85", alpha = 0.05) +  
 geom\_rect(aes(xmin = as.Date("2020-12-26") , xmax = as.Date("2021-03-29"),   
 ymin = -Inf, ymax = Inf),   
 color = NA, fill = "gray85", alpha = 0.05) +  
 geom\_rect(aes(xmin = as.Date("2021-12-08") , xmax = as.Date("2022-01-27"),   
 ymin = -Inf, ymax = Inf),   
 color = NA, fill = "gray85", alpha = 0.05) +  
 geom\_ribbon(aes(ymin = pred\_lower, ymax = pred\_upper), alpha = 0.2, color = "orange", fill = "orange") +   
 geom\_line(color = "orange") +   
 geom\_line(aes(y = pred\_isr), color = "blue") +   
 geom\_ribbon(aes(ymin = pred\_lower\_isr, ymax = pred\_upper\_isr), alpha = 0.2, color = "blue", fill = "blue") +   
 theme\_classic() +   
 labs(x = "Date", y = "Predicted percent testing positive") +   
 scale\_x\_date(date\_breaks = "2 months",   
 date\_labels = format("%d%b%y"),   
 limits = c(mindate, maxdate), expand = c(0.025, 0.025)) +   
 scale\_y\_continuous(breaks = seq(0, 10, 2),   
 labels = seq(0, 10, 2),   
 limits = c(-0.1, 10)) +   
 ggtitle("London") +   
theme(plot.title = element\_text(size = 13, face = "bold")) +  
 geom\_rect(aes(xmin = mindate, xmax = variant\_dates[1], ymin = 9.6, ymax = 10),   
 color = "black", fill = "slategray1") +   
 geom\_text(aes(x = mindate+65, y = 9.8), label = "Pre-Alpha", size = 3.5) +   
 geom\_rect(aes(xmin = variant\_dates[1], xmax = variant\_dates[2], ymin = 9.6, ymax = 10),   
 color = "black", fill = "lightgoldenrod1") +   
 geom\_text(aes(x = variant\_dates[1]+75, y = 9.8), label = "Alpha dominant", size = 3.5) +   
 geom\_rect(aes(xmin = variant\_dates[2], xmax = variant\_dates[3], ymin = 9.6, ymax = 10),   
 color = "black", fill = "slategray1") +   
 geom\_text(aes(x = variant\_dates[2]+90, y = 9.8), label = "Delta dominant", size = 3.5) +   
 geom\_rect(aes(xmin = variant\_dates[3], xmax = variant\_dates[4], ymin = 9.6, ymax = 10),   
 color = "black", fill = "lightgoldenrod1") +   
 geom\_text(aes(x = variant\_dates[3]+38, y = 9.8), label = "BA.1 dominant", size = 3.5) +   
 geom\_rect(aes(xmin = variant\_dates[4], xmax = variant\_dates[5], ymin = 9.6, ymax = 10),   
 color = "black", fill = "slategray1") +   
 geom\_text(aes(x = variant\_dates[4]+50, y = 9.8), label = "BA.2 dominant", size = 3.5) +   
 geom\_rect(aes(xmin = variant\_dates[5], xmax = maxdate, ymin = 9.6, ymax = 10),   
 color = "black", fill = "lightgoldenrod1") +   
 geom\_text(aes(x = variant\_dates[5]+12, y = 9.8), label = "BA.4/\nBA.5", size = 2.5) +   
 geom\_vline(xintercept = as.numeric(variant\_dates), col = "black", linetype = "dashed", size = 1) +  
 theme(panel.grid.major = element\_line())  
p\_gamisr

The plot produced by this code can be found in the pre-print/paper as **Figure 1**. This process was repeated for the other 11 geographical regions included in our dataset.

REFERENCE LIST

1. Simpson GL. *gratia*: Graceful *ggplot*-Based Graphics and Other Functions for *GAMs* Fitted using *mgcv*. . 2022. <https://gavinsimpson.github.io/gratia/>.

2. Schlackow I, Walker AS, Dingle K, et al. Surveillance of infection severity: a registry study of laboratory diagnosed Clostridium difficile. *PLoS Med* 2012; **9**(7): e1001279.