gradient descent

Alyssa Vanderbeek

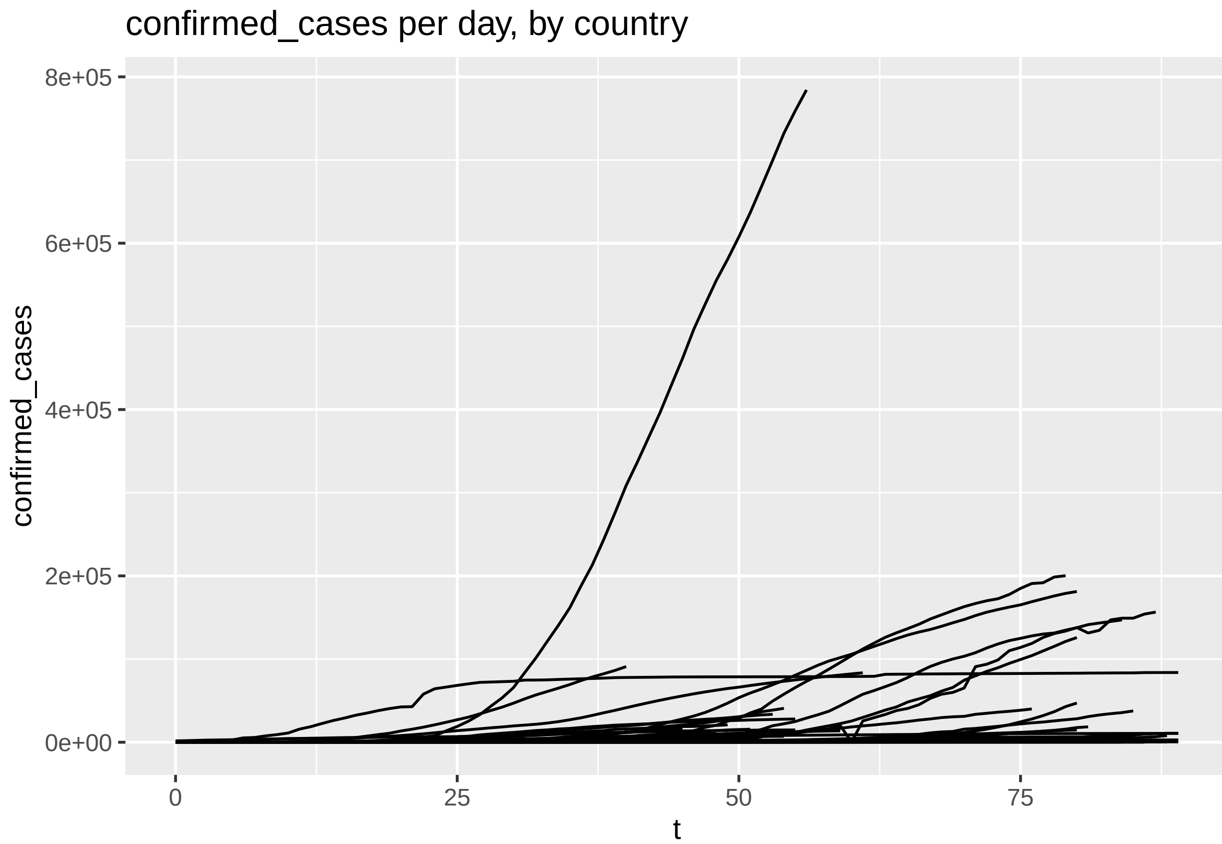
4/22/2020

```r
covid = read.csv(file.path(getwd(), "covid_final.csv")) %>%
  filter(confirmed_cases > 0) %>%
  mutate(date = lubridate::parse_date_time(x = date,
                                           orders = c("m/d/y", "m-d-Y")))


## group data by country
by_country = covid %>%
  dplyr::group_by(country_region) %>%
  dplyr::mutate(t = as.numeric(difftime(date, min(date), units = "days"))) %>%
  dplyr::group_by(country_region, t) %>%
  dplyr::summarise(confirmed_cases = sum(confirmed_cases), # group by date and take total
            fatalities = sum(fatalities))


# covid_ls = list(x = by_country[,c("country_region", "t")], y = by_country$confirmed_cases)


by_country %>%
  ggplot(., aes(x = t, y = confirmed_cases, group = country_region)) +
  geom_line() +
  labs(title = "confirmed_cases per day, by country",
       xlab = "Days since first infection",
       ylab = "Number of confirmed_cases")
```




```r
# FUNCTION TO CALCULATE THE GRADIENT
# logistic_gradient = function(data, a, b, c) {
#   # a = betavec[1] # upper bound
#   # b = betavec[2] # growth rate
#   # c = betavec[3] # midpoint of the curve
#   x = data$t  # predictor: days since first infection
#
#   # gradient
#   grad <- c(
#     sum(2 / (1 + exp(-b*(x - c)))) / length(x),
#     sum(-(2*a*(c - x)*exp(-b*(x - c))) / (1 + exp(-b*(x - c)))^2) / length(x) ,
#     sum(-(2*a*b*exp(-b*(x - c))) / (1 + exp(-b*(x - c)))^2) / length(x)
#   )
#
#   return(grad)
# }

# calculate predictions given data and parameter values
logistic_pred = function(t, a, b, c) {
  t  # predictor: days since first infection
  y_hat = a / (1 + exp(-b*(t - c)))
  return(y_hat)
}

## Our goal is to minimize the loss function (SSE)
loss = function(data, a, b, c) {
```

```r
  y = data$confirmed_cases
  y_hat = logistic_pred(data$t, a, b, c) # predicted
  error = y_hat - y

  loss = sum( error^2 ) / length(y)
  return(loss)
}


## Coordinate-wise optimization to fit logistic curve to data
fit_curve = function(data, aseq, bseq, cseq) {
  c_lim = c(min(cseq), max(cseq))

  # matrix of optimal values of c across all combos of a and b values
  cvals = sapply(aseq, function(a){
    sapply(bseq, function(b){
      optimize(loss, interval = c_lim, data = data, a = a, b = b)$minimum
    })
  })
  # matrix of the computed loss for all combos
  closs = sapply(aseq, function(a){
    sapply(bseq, function(b){
      optimize(loss, interval = c_lim, data = data, a = a, b = b)$objective
    })
  })

  loc = which(closs == min(closs), arr.ind = TRUE)
  a = aseq[loc[2]]
  b = bseq[loc[1]]
  c = cvals[loc[1], loc[2]]

  return(cbind(a = a, b = b, c = c))
}



# a.seq = seq(0, 1e6, 1e4)
# b.seq = seq(0, 1, 0.01)
# c.seq = seq(0, 90, 1)
#
#
# usa = by_country %>%
#   filter(country_region == "US")
#
# usa_curve = fit_curve(usa, aseq = a.seq, bseq = b.seq, cseq = c.seq)
# usa_fitted = logistic_pred(usa$t, usa_curve[1], usa_curve[2], usa_curve[3])
# plot(x = usa$t,
#      y = usa$confirmed_cases)
# lines(x = usa$t,
#       y = usa_fitted)
#
#
# china = by_country %>%
```

```
#   filter(country_region == "China")
# china_curve = fit_curve(china, aseq = a.seq, bseq = b.seq, cseq = c.seq)
# china_fitted = logistic_pred(seq(0, 89, 1), china_curve[1], china_curve[2], china_curve[3])
#
# plot(x = china$t,
#      y = china$confirmed_cases)
# lines(x = china$t,
#       y = china_fitted)
# data = usa
# start = c(a = 5e5, b = 0.4, c = 30)
# #tol = 1e-10
# maxiter = 200
#
# x = data$t  # predictor
# y = data$confirmed_cases
#
# i <- 0
# p <- length(start) # number of parameters being estimated
# n <- length(y) # number of obervations
# betavec <- start # initial guess of parameters
# gradient = logistic_gradient(data, betavec)
# Hess = diag(p) # for graident descent, the Hessian is the identity matrix
#
# current_loss <- loss(data, betavec) # since our goal is to
# prev_loss <- -Inf # To make sure it iterates
#
# half_initial <- 0.5 # using half steps for betavec update
# halving_iteration = 1
#
# res = c(0, current_loss, betavec)
#
# # while the current loss is greater than the last (since we want to minimize)
# while (i < maxiter && current_loss < prev_loss) {
#   i = i + 1
#   prev_loss = current_loss
#
#   gradient = logistic_gradient(data, betavec) # get new gradient
#   betavec = betavec + Hess%*%gradient
#
#   current_loss = loss(data, betavec) # updated loss with new parameter values
#
#   continue = current_loss < prev_loss
#   # while the current loss is greater than the last, keep halving the betavec
#
#   if (continue) {
#     while (continue == TRUE) {
#       halving_iteration = halving_iteration + 1
#       half = half_initial^halving_iteration
#       betavec = betavec - half*gradient
#       current_loss = loss(data, betavec)
#       res = rbind(res, c(i, current_loss, betavec))
#
#       continue = current_loss > prev_loss
```

```
#     }
#   } else {
#       res = rbind(res, c(i, current_loss, betavec))
#   }
#
# }
```

Loop over countries

```
country_reg = by_country$country_region %>% unique() %>% as.vector()
a.seq = c(0, 100, 500, 1e3, seq(1e4, 1e6, 1e4))
b.seq = seq(0, 1, 0.01)
c.seq = seq(0, 90, 1)
param_df = NULL
fitted_list <- list()

for (i in country_reg[1:186]){
 param = fit_curve(by_country %>% filter(country_region == i),
                      aseq = a.seq, bseq = b.seq, cseq = c.seq)
 fitted = logistic_pred(as.numeric(as.matrix(filter(by_country,country_region == i)[,2])),
                        a = as.numeric(param[1]),
                        b = as.numeric(param[2]),
                        c = as.numeric(param[3]))
 param_df = rbind(param_df, param)
 fitted_list[[i]] <- fitted
}

param_df1 = tibble(region = country_reg[1:186],
                   a = param_df[,1],
                   b = param_df[,2],
                   c = param_df[,3])
```