Project 3 - COVID19 Model

Margaret Gacheru, Joy Hsu, Melanie Mayer, Rachel Tsong, Adina Zhang

4/16/2020

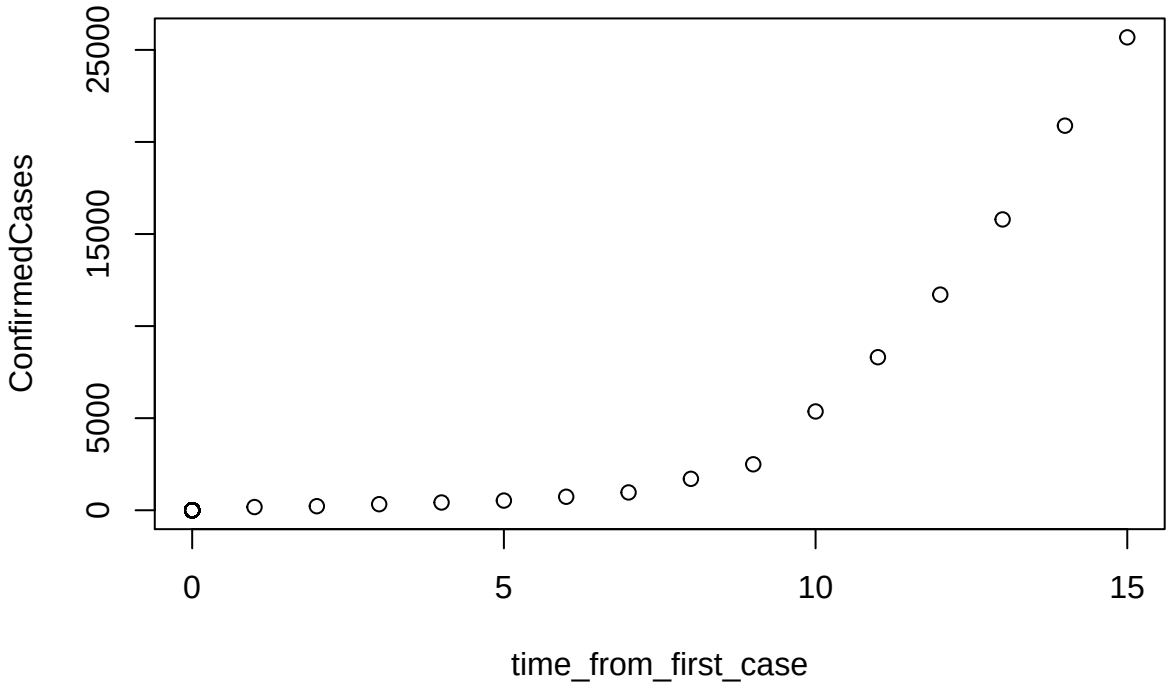
Read in data:

```
covid19 <- read_csv("covid19-1.csv")
```

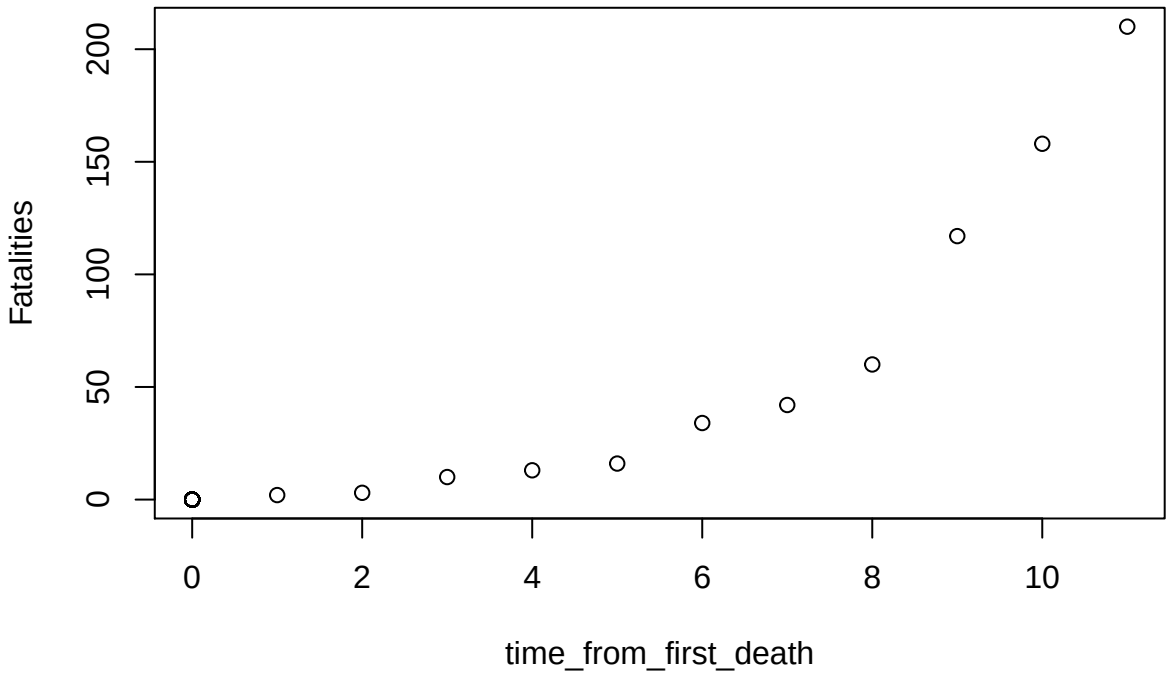
```
## Parsed with column specification:
## cols(
##   Id = col_double(),
##   `Province/State` = col_character(),
##   `Country/Region` = col_character(),
##   Lat = col_double(),
##   Long = col_double(),
##   Date = col_character(),
##   ConfirmedCases = col_double(),
##   Fatalities = col_double()
## )
```

Explore data - NY Example:

```
NY_dat = covid19 %>% filter(`Province/State` == "New York")

#Initialize column - time from 1st case
NY_dat$time_from_first_case <- rep(0, dim(NY_dat)[1])
#Find location of first confirmed case
case = sum(NY_dat$ConfirmedCases == 0)
#Fill in column
j = 1
for (i in (case+1):dim(NY_dat)[1]) {
  NY_dat$time_from_first_case[i] = j
  j = j+1
}

#Check whether logistic curve is good aproximation - cases
plot(ConfirmedCases ~ time_from_first_case, data = NY_dat)
```




```
#Initialize column - time from 1st death
NY_dat$time_from_first_death <- rep(0, dim(NY_dat)[1])
#Find location of first confirmed case
deaths = sum(NY_dat$Fatalities == 0)
#Fill in column
j = 1
for (i in (deaths+1):dim(NY_dat)[1]) {
  NY_dat$time_from_first_death[i] = j
  j = j+1
}

#Check whether logistic curve is good aproximation - fatalities
plot(Fatalities ~ time_from_first_death, data = NY_dat)
```




```
#Appears to have exponential growth, will try logistic growth model however
```

Estimation of logistic growth curve parameters using R functions:

```
model <- drc::drm(ConfirmedCases ~ time_from_first_case, fct = L.3(), data = NY_dat)
plot(model, log="", main = "Logistic function")

model <- drc::drm(Fatalities ~ time_from_first_death, fct = L.3(), data = NY_dat)
plot(model, log="", main = "Logistic function")
```

Newton Raphson

```
NewtonRaphson = function(data, main_function, start, tol = 1e-10, maxiter = 200) {
  #convert data into matrix form
  model_data = data%>%
    select(-c("diagnosis"))
  intercept = rep(1, dim(model_data)[1])
  #X = as.matrix(cbind(intercept, scale(model_data, scale = TRUE)))
  X = as.matrix(cbind(intercept, model_data))
  Y = as.matrix(as.integer(data$diagnosis == "M"))
  Beta = as.matrix(start, nrow = dim(model_data)[2] + 1)

  i = 0
  cur = Beta
  lik_grad_hess = main_function(Y, X, cur)
  res = c(0, lik_grad_hess$loglik, cur)
  step = 1

  prevloglik = -Inf # To make sure it iterates

  diff_loglik = abs(lik_grad_hess$loglik - prevloglik)
```

```
  #if (is.nan(diff_loglik)) { diff_loglik <- 1e-2 }

  while(i < maxiter && diff_loglik > tol) {
    i = i + 1

    prevlik_grad_hess = lik_grad_hess #time step i - 1
    prevloglik = prevlik_grad_hess$loglik
    prev = cur #step i - 1


    #ensure that the direction of the step is in ascent direction
    d_grad = - t(prevlik_grad_hess$grad) %*% ginv(prevlik_grad_hess$Hess, 2.9876e-18 ) %*% (prevlik_grad

    #max_eig = max(eigen(prevlik_grad_hess$Hess)$values)
    n = ncol(prevlik_grad_hess$Hess)
    gamma = 0.01

    while (d_grad <= 0){

      prevlik_grad_hess$Hess = prevlik_grad_hess$Hess - gamma*diag(n)

      d_grad = - t(prevlik_grad_hess$grad) %*% ginv(prevlik_grad_hess$Hess, 2.9876e-18 ) %*% (prevlik_g

      gamma = gamma + 0.01
    }

    cur = prev - ginv(prevlik_grad_hess$Hess, 2.9876e-18) %*% prevlik_grad_hess$grad #step find theta f
    lik_grad_hess = main_function(Y, X, cur) #update log-lik, gradient, Hessian for step i

    while (lik_grad_hess$loglik < prevloglik) {

      step = 0.5*step
      cur = prev - step * ginv(prevlik_grad_hess$Hess, 2.9876e-18) %*% prevlik_grad_hess$grad
      lik_grad_hess = main_function(Y, X, cur)

      }

    res = rbind(res, c(i, lik_grad_hess$loglik, cur))

    diff_loglik = abs(lik_grad_hess$loglik - prevloglik)
    if (is.nan(diff_loglik)) { diff_loglik <- 1e-2 }

    }

  return(res)

  }

path = NewtonRaphson(breastcancer_data, main_function, start = rep(0, dim(data)[2]))
path[dim(path)[1],]
```

Create time from first case/fatality variables for entire dataset by Country/Region:

```
#create desired variables, remove regions with less than 14 days with cases
covid19_country <- covid19 %>%
  group_by(`Country/Region`, Date) %>%
  summarise(ConfirmedCases = sum(ConfirmedCases),
            Fatalities = sum(Fatalities)) %>%
  filter(ConfirmedCases != 0) %>%
  mutate(Date = as.Date(Date, format="%m/%d/%Y")) %>%
  arrange(`Country/Region`, Date) %>%
  mutate(
    time_from_first_case = ifelse(ConfirmedCases >= 1, 1, 0),
         time_from_first_case = cumsum(time_from_first_case),
         time_from_first_death = ifelse(Fatalities >= 1, 1, 0),
         time_from_first_death = cumsum(time_from_first_death)) %>%
  filter(max(time_from_first_case) >= 14)
```

Estimate Logistic Curve by Country/Region:

```
#Find estimates per country of cases - premade R function
cases_country_curves <- covid19_country %>%
  split(.$`Country/Region`) %>%
  map(~coefficients(drm(ConfirmedCases ~ time_from_first_case, fct = L.3(), data = .)))

cases_country_curves <- data.frame(matrix(unlist(cases_country_curves), nrow=length(cases_country_curves

cases_country_curves <- cbind(cases_country_curves, Country = levels(factor(covid19_country$`Country/Reg
  rename(B = "X1", A = "X2", C = "X3")


#Find estimates per country of fatalities - premade R function
#Fails to converge, use oy countries with time since first death >= 14
covid19_country_fatal <- covid19_country %>%
  filter(max(time_from_first_death) >= 14)
fatalities_country_curves <- covid19_country_fatal %>%
  split(.$`Country/Region`) %>%
  map(~coefficients(drm(Fatalities ~ time_from_first_death, fct = L.3(), data = .)))

fatalities_country_curves <- data.frame(matrix(unlist(fatalities_country_curves), nrow=length(fatalities

fatalities_country_curves <- cbind(fatalities_country_curves, Country = levels(factor(covid19_country_fa
  rename(B = "X1", A = "X2", C = "X3")
```

Task 2. Clustering your fitted Curves

K-mean clustering:

- K clusters: $C_1, C_2, ..., C_k$
- Want to minimize within cluster correlation: minimize

$$\sum_{k=1}^{K} W(C_k)$$

- Define within cluster variance using the squared Euclidian distance:

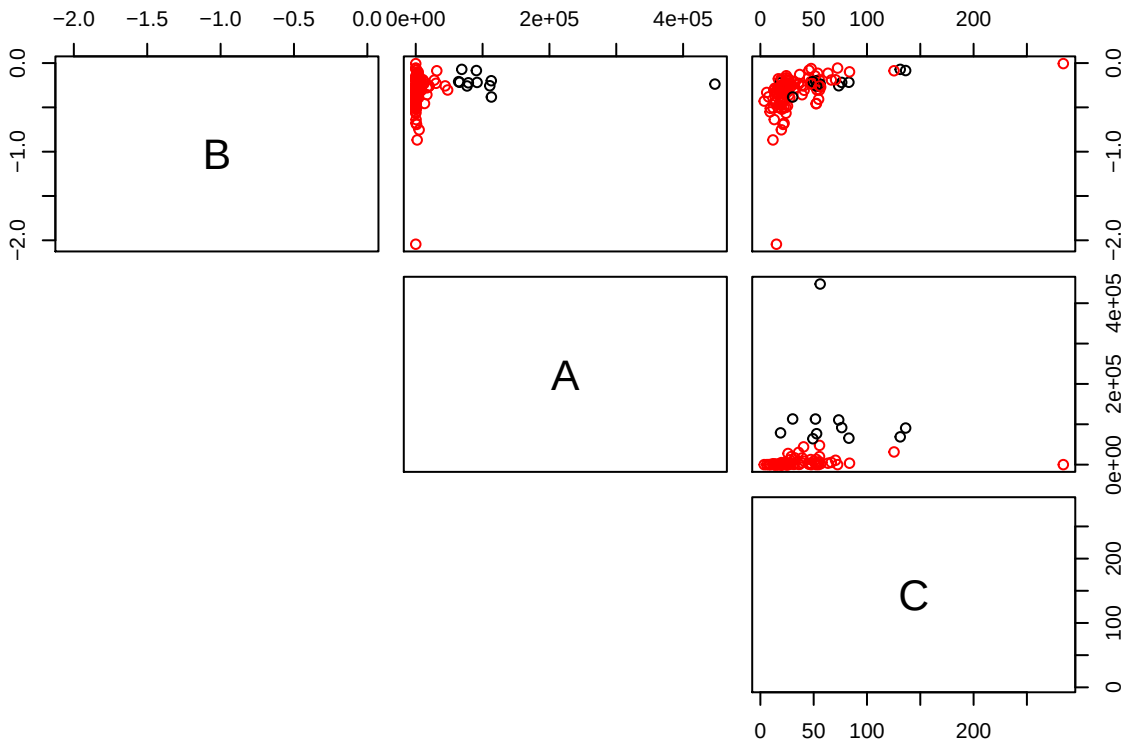
$$W(C_k) = \frac{1}{|x|} \sum_{i,i' \in C_k} \sum_{j=1}^{p} (x_{ij - x_{i'j}})^2$$

  where $|C_k|$ is the number of observations in cluster k

Algorithm:

1. Randomly choose k observations, use chosen observations' p observed values as centroid
2. Assign each observation to the cluster whose centroid is closest based on sum of p Euclidian distances
3. Compute p * k cluster means
4. Iterate step 2 & 3 until stop changing

```
kmeans_cluster <- function(X, k){
  #X: data frame
  #k: number of clusters desired
  p <- dim(X)[2]  # number of parameters
  n <- dim(X)[1]  # number of observations
  delta <- 1
  iter <- 0
  itermax <- 30
  while(delta > 1e-4 && iter <= itermax){
    if(iter == 0){
      centroid <- X[sample(n, k),] #Initiate, randomly pick three observations, use values as centroid
      centroid_mem <- centroid
    }

    d <- sapply(1:k, function(c) sapply(1:n,
      function(i) sum((centroid[c,] - X[i,])^2))) #sum of p Euclidian distances from k centroids per ob

    cluster <- apply(d, 1, which.min) #Place obs. in cluster with smallest distance

    centroid <- t(sapply(1:k, function(c)
      apply(X[cluster == c,], 2, mean))) # Compute new k*p centroids

    delta <- sum((centroid - centroid_mem)^2) #Check converegence
    iter <- iter + 1
    centroid_mem <- centroid
  }
  X = cbind(X, cluster = cluster)
  return(list(centroid = centroid, cluster = cluster, df = X))
}

# run K-means
km <- kmeans_cluster(cases_country_curves[,1:3], 2)
pairs(cases_country_curves[,1:3], lower.panel = NULL, col = km$cluster)
```




```
summary(factor(km$cluster))
```

```
##  1  2
## 11 98
```

```
#Compare to package
km_pkg <- kmeans(cases_country_curves[,1:3], 2)
summary(factor(km_pkg$cluster))
```

```
##  1  2
## 11 98
```

```
km_pkg_vec <- cbind(cases_country_curves, cluster = km_pkg$cluster)
```