```
### SIR MODEL (OC) ===== ===== ===== ===== =====

# last updated: March 31, 2023

library(cowplot)
library(deSolve)
library(ggplot2)
library(graphics)
library(reshape2)
library(tidyverse)

rm(list=ls()) # clear environment


### Initialize ===== ===== ===== ===== =====

# time
dt = 0.05
tvals = seq(0,200,by=dt)
tlen = length(tvals)

# parameters
beta = 0.00009
gamma = 0.05
C1 = 1
C2 = 0.1
epsilon = 100
vmax = 0.05

pars = c(beta=beta, gamma=gamma,
         C1=C1, C2=C2, epsilon=epsilon, vmax=vmax)

# initial conditions
N = 10000
S0 = N - 1
I0 = 1
R0 = 0
Cases0 = 0

inits_sta = c(S=S0, I=I0, R=R0, Cases=Cases0)

# transversality condition, lambda(T) = 0
inits_lam = c(lambdaS=0, lambdaI=0, lambdaR=0)
```

```
### Equations ===== ===== ===== ===== =====

# states (x)
SIR = function(t, x, pars){
  with(as.list(c(x, pars)), {
    v = v_interp(t)
    dS = -beta*S*I - v*S
    dI = beta*S*I-gamma*I
    dR = gamma*I + v*S
    dCases = beta*S*I
    return(list(c(dS, dI, dR, dCases)))
  })
}

# adjoints (lambda)
Lambda = function(t, lambda, pars){
  state = sapply(1:length(pars$x_interp),
                 function(i){pars$x_interp[[i]](t)})
  names(state) = c("S", "I", "R")
  with(as.list(c(lambda, pars, state)), {
    v = v_interp(t)
    dlambdaS = -(C1*beta*I + C2*v - lambdaS*beta*I -
                   lambdaS*v + lambdaI*beta*I + lambdaR*v)
    dlambdaI = -(C1*beta*S - lambdaS*beta*S + lambdaI*beta*S -
                   lambdaI*gamma + lambdaR*gamma)
    dlambdaR = -(0)
    return(list(c(dlambdaS, dlambdaI, dlambdaR)))
  })
}


### Forward-Backwards Sweep ===== ===== ===== ===== =====

delta = 0.01
test = -1
count = 0

# initial guesses
v = rep(0, tlen)
v_interp = approxfun(tvals, v, rule=2)
pars = c(pars, v_interp)
solx = ode(y=inits_sta, times=tvals, func=SIR, parms=pars)
x_interp <- lapply(2:ncol(solx),
```

```
                    function(x){approxfun(solx[,c(1,x)],
                                          rule = 2)})
pars = c(pars, x_interp)
lambda = matrix(0, tlen, 3)

# while loop
while(test < 0 & count < 100){

  print(count)

  # set previous control, state, adjoint
  v_old = v
  x_old = solx
  lambda_old = lambda

  # interpolate control (v)
  pars$v_interp = approxfun(tvals, v, rule=2)

  # solve and interpolate states (x)
  solx = ode(y=inits_sta, times=tvals, func=SIR, parms=pars)
  pars$x_interp = lapply(2:ncol(solx),
                         function(y){approxfun(solx[,c(1,y)],
                                               rule=2)})

  S = solx[,"S"]
  I = solx[,"I"]
  R = solx[,"R"]
  Cases = solx[,"Cases"]

  # solve and interpolate adjoints (lambda)
  sollambda = ode(y=inits_lam, times=rev(tvals),
                  func=Lambda, parms=pars)
  lambda = sollambda[nrow(lambda):1,]

  lambdaS = lambda[,"lambdaS"]
  lambdaI = lambda[,"lambdaI"]
  lambdaR = lambda[,"lambdaR"]

  # calculate control characteristic and update control
  v_char = with(pars, (-C2*S + lambdaS*S -
                         lambdaR*S)/(2*epsilon))
  v_star = pmin(vmax, pmax(0, v_char))
  v = (v_old + v_star)*0.5
```

```
  # test convergence
  test = delta*sum(abs(v)) - sum(abs(v_old - v))
  print(test)

  # update count
  count = count + 1
}


### Total Cases ===== ===== ===== ===== =====

TotC = round(Cases[length(Cases)], digits=0)
print(TotC)


### Total Cost (J) ===== ===== ===== ===== =====

Jcases = sum(C1*beta*I*S)*dt
Jvax = sum(C2*v*S + epsilon*v^2)*dt
Jtot = Jcases + Jvax

print(Jcases)
print(Jvax)
print(Jtot)


### Plots ===== ===== ===== ===== =====

OC_states = solx %>% data.frame(S=S, I=I, R=R)
SIR_plot = OC_states %>% gather(key, individuals, S, I, R) %>%
  ggplot(aes(x=time, y=individuals, color=key)) +
  geom_line() +
  ggtitle("(a) SIR model with vax") +
  xlab("days") + ylab("individuals")

OC_vax = data.frame(tvals, v)
v_plot = OC_vax %>% gather(key, vax, v) %>%
  ggplot(aes(x=tvals, y=v, color=key)) +
  geom_line() +
  ggtitle("(b) Optimal vax rate") +
  xlab("days") + ylab("prop. of ind. per day")

plot_grid(SIR_plot, v_plot, ncol = 1, nrow = 2)
```