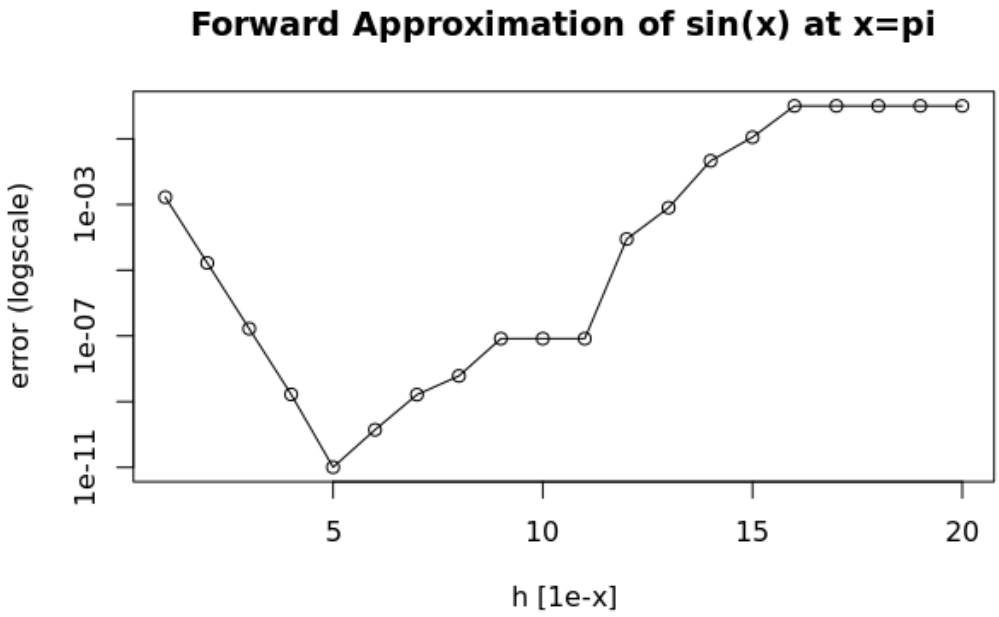
CS-7863: Scientific and Statistical Computing
Homework #2
Noah L. Schrick – 1492657

1.)

```
## 1. Approximate deriv. of sin(x) at x=pi from h=10^-1 -> 10^-20
forward.approx <- function(func, x, h){
  approx <- (func(x+h)-func(x))/h
}

forward.approx.table <- matrix(nrow = 0, ncol = 3)
colnames(forward.approx.table) <- c("h", "approximation", "error")
for(h in 10^(seq(-1,-20,-1))){
  approx <- forward.approx(sin, pi, h)
  error <- abs(cos(pi)-approx)
  forward.approx.table <- rbind(forward.approx.table, c(h, approx, error))
}
```

	h	approximation	error
[1,]	1e-01	-0.9983342	1.665834e-03
[2,]	1e-02	-0.9999833	1.666658e-05
[3,]	1e-03	-0.9999998	1.666668e-07
[4,]	1e-04	-1.0000000	1.664556e-09
[5,]	1e-05	-1.0000000	1.011558e-11
[6,]	1e-06	-1.0000000	1.396114e-10
[7,]	1e-07	-1.0000000	1.636580e-09
[8,]	1e-08	-1.0000000	6.077471e-09
[9,]	1e-09	-1.0000001	8.274037e-08
[10,]	1e-10	-1.0000001	8.274037e-08
[11,]	1e-11	-1.0000001	8.274037e-08
[12,]	1e-12	-1.0000889	8.890058e-05
[13,]	1e-13	-0.9992007	7.992778e-04
[14,]	1e-14	-1.0214052	2.140518e-02
[15,]	1e-15	-0.8881784	1.118216e-01
[16,]	1e-16	0.0000000	1.000000e+00
[17,]	1e-17	0.0000000	1.000000e+00
[18,]	1e-18	0.0000000	1.000000e+00
[19,]	1e-19	0.0000000	1.000000e+00
[20,]	1e-20	0.0000000	1.000000e+00

```
plot(abs(log(forward.approx.table[,"h"],10)), forward.approx.table[,"error"],
     xlab="h [1e-x]", ylab="error (logscale)", type="o", log="y",
     main = "Forward Approximation of sin(x) at x=pi")
```

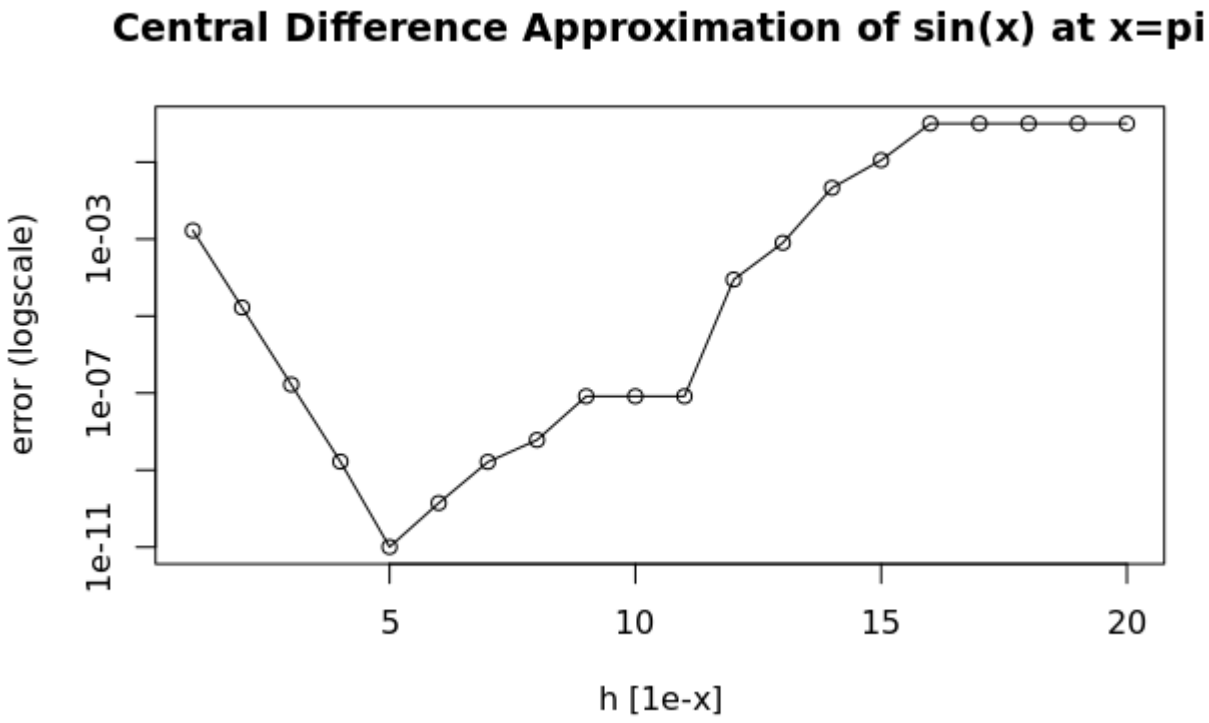



```
# Repeat with central difference approx
central.diff.approx <- function(func, x, h){
  approx <- (func(x+h)-func(x-h))/(2*h)
}
central.diff.table <- matrix(nrow = 0, ncol = 3)
colnames(central.diff.table) <- c("h", "approximation", "error")
for(h in 10^(seq(-1,-20,-1))){
  approx <- central.diff.approx(sin, pi, h)
  error <- abs(cos(pi)-approx)
  central.diff.table <- rbind(central.diff.table, c(h, approx, error))
}
```

	h	approximation	error
[1,]	1e-01	-0.9983342	1.665834e-03
[2,]	1e-02	-0.9999833	1.666658e-05
[3,]	1e-03	-0.9999998	1.666668e-07
[4,]	1e-04	-1.0000000	1.664556e-09
[5,]	1e-05	-1.0000000	1.011546e-11
[6,]	1e-06	-1.0000000	1.396114e-10
[7,]	1e-07	-1.0000000	1.636580e-09
[8,]	1e-08	-1.0000000	6.077471e-09
[9,]	1e-09	-1.0000001	8.274037e-08
[10,]	1e-10	-1.0000001	8.274037e-08
[11,]	1e-11	-1.0000001	8.274037e-08
[12,]	1e-12	-1.0000889	8.890058e-05
[13,]	1e-13	-0.9992007	7.992778e-04
[14,]	1e-14	-1.0214052	2.140518e-02
[15,]	1e-15	-0.8881784	1.118216e-01
[16,]	1e-16	0.0000000	1.000000e+00

```
[17,] 1e-17    0.0000000     1.000000e+00
[18,] 1e-18    0.0000000     1.000000e+00
[19,] 1e-19    0.0000000     1.000000e+00
[20,] 1e-20    0.0000000     1.000000e+00
```

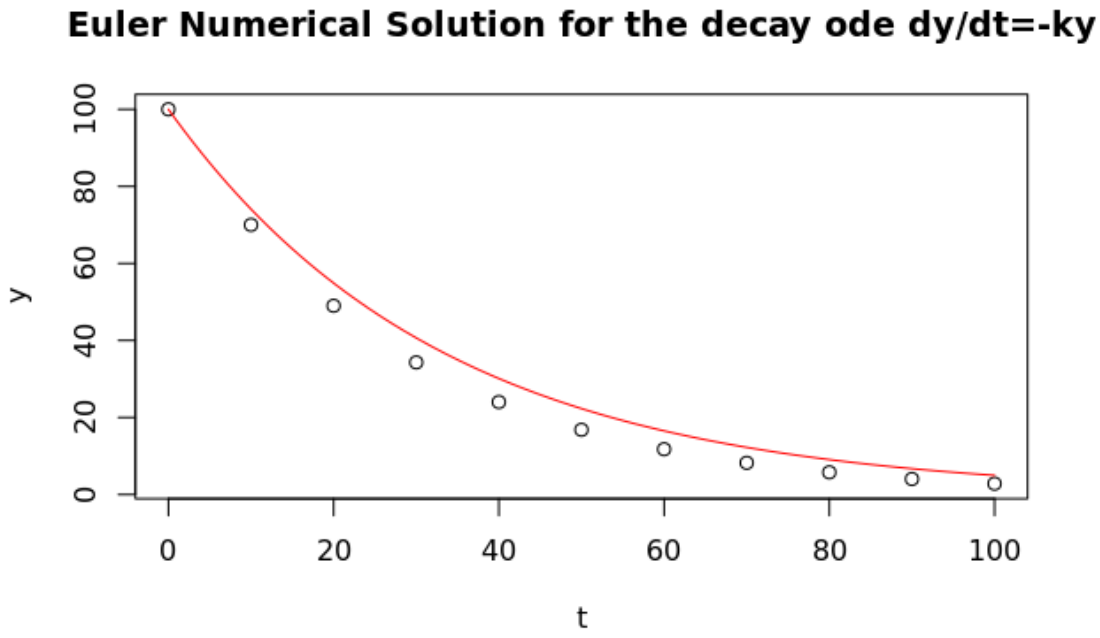
```
plot(abs(log(central.diff.table[,"h"],10)), central.diff.table[,"error"],
     xlab="h [1e-x]", ylab="error (logscale)", type="o", log="y",
     main = "Central Difference Approximation of sin(x) at x=pi")
```



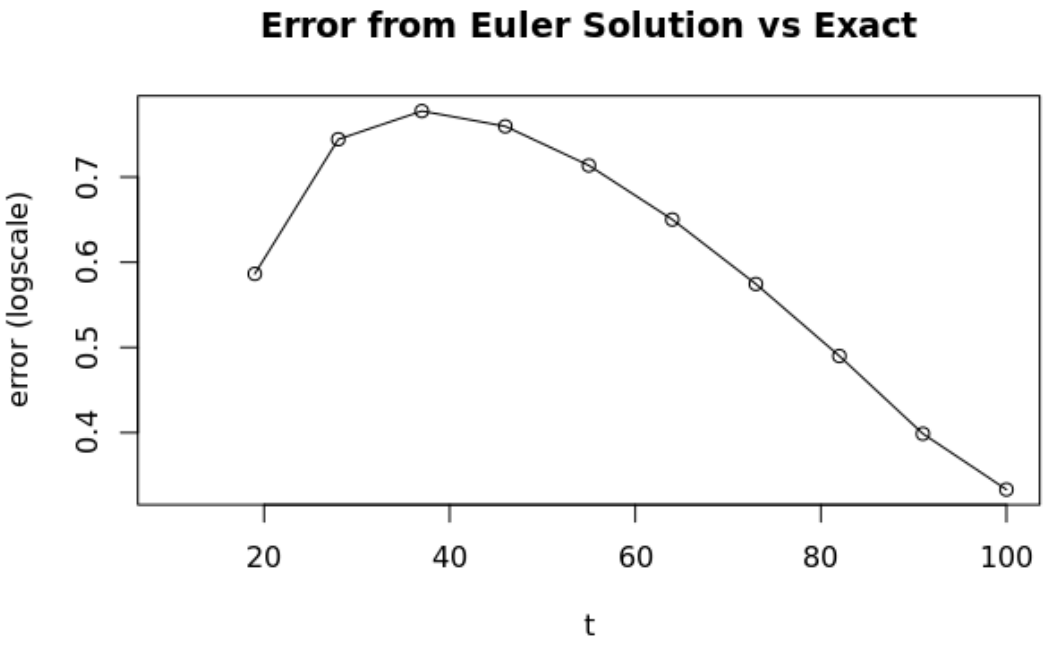
2.)

```
# a) 4th-order Runge-Kutta
my_rk4 <- function(f, y0, t0, tfinal, h){
  tspan <- seq(t0,tfinal,h)
  npts <- length(tspan)
  y<-rep(y0,npts) # initialize, this will be a matrix for systems

  for (i in seq(2,npts)){
    k1 <- f(t[i-1], y[i-1])
    k2 <- f(t[i-1] + 0.5 * h, y[i-1] + 0.5 * k1 * h)
    k3 <- f(t[i-1] + 0.5 * h, y[i-1] + 0.5 * k2 * h)
    k4 <- f(t[i-1] + h, y[i-1] + k3*h)
    # Update y
    y[i] = y[i-1] + (h/6)*(k1 + 2*k2 + 2*k3 + k4)
    # Update t
    #t0 = t0 + h
  }
  return(list(t=tspan,y=y))
}

# b) Numerically solve the decay ode and compare to Euler and RK error
decay.f <- function(t,y,k){
  # define the ode model
  # k given value when solver called
  dydt <- -k*y
  as.matrix(dydt)
}

# From class docs:
my_euler <- function(f, y0, t0, tfinal, dt){
  # follow ode45 syntax, dt is extra
  # y0 is initial condition
  tspan <- seq(t0,tfinal,dt)
  npts <- length(tspan)
  y<-rep(y0,npts) # initialize, this will be a matrix for systems
  for (i in 2:npts){
    dy <- f(t[i-1],y[i-1])*dt # t is not used in this case
    y[i] <- y[i-1] + dy
  }
  return(list(t=tspan,y=y))
}

# Solve
t0 <- 0
tfinal <- 100
h <- 10
k <- 0.03
y0 <- 100

decay.euler.sol <- my_euler(f=function(t,y){decay.f(t,y,k=k)},
                            y0, t0, tfinal, dt=h)
# Exact sol
exact.y <- y0*exp(-k*seq(t0,tfinal,len=100))
exact.y.slice <- exact.y[seq(1,length(exact.y),h)]
# Add in extra point to make lengths even to euler sol
exact.y.slice <- append(exact.y.slice,exact.y[length(exact.y)])
```

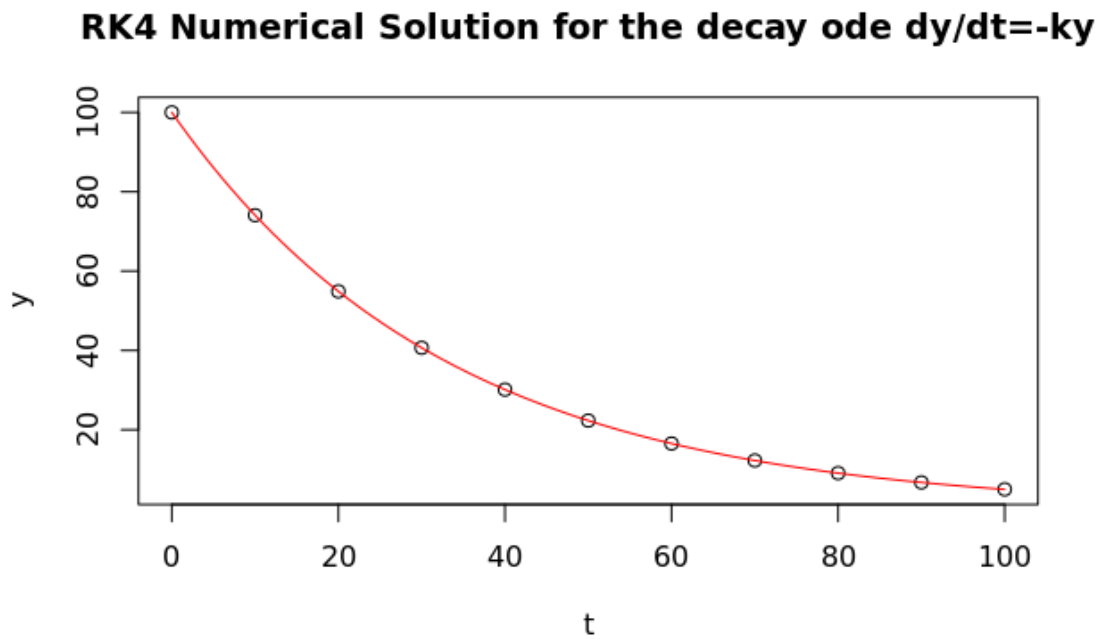
```
plot(decay.euler.sol$t,decay.euler.sol$y, xlab="t", ylab="y",
     main="Euler Numerical Solution for the decay ode dy/dt=-ky")
par(new=T)
lines(seq(t0,tfinal,len=100),
      exact.y, col="red")
```



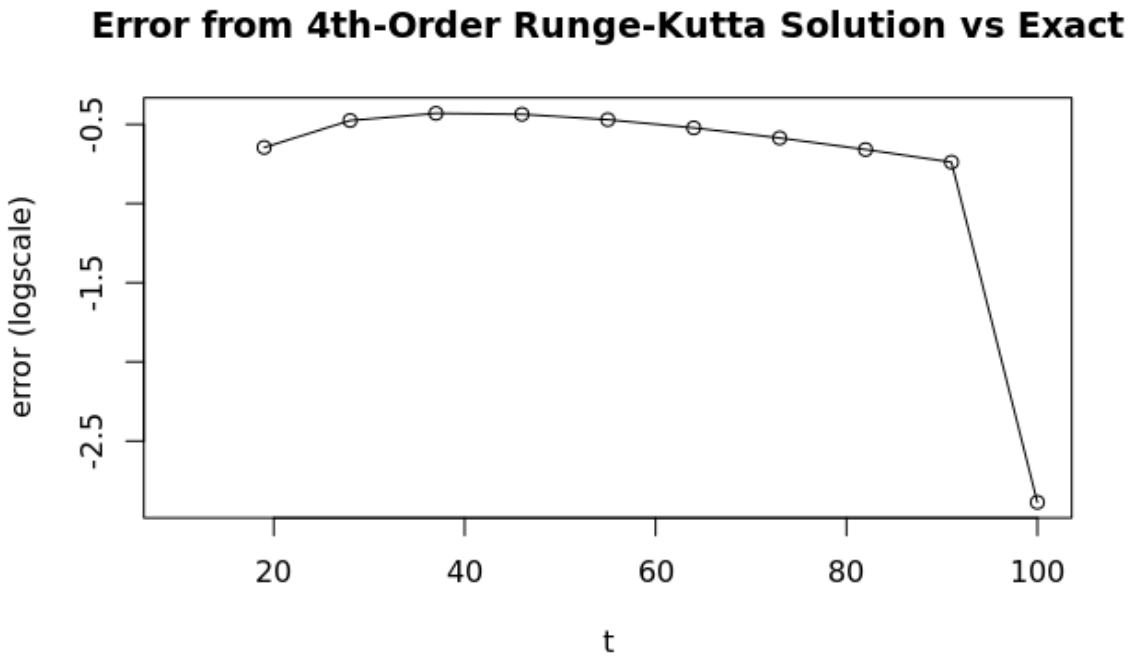

```
# Error
euler.err <- abs(decay.euler.sol$y - exact.y.slice)
euler.log.err <- log(euler.err, 10)
# Unsophisticated way to account for the -Inf error in the logscale
euler.log.err[which(!is.finite(euler.log.err))] <- NA

plot(seq(t0+h,tfinal,len=11), euler.log.err,
     xlab="t", ylab="error (logscale)", type="o",
     main = "Error from Euler Solution vs Exact")
```




```
# Repeat with RK4
decay.rk4.sol <- my_rk4(f=function(t,y){decay.f(t,y,k=k)},
                        y0, t0, tfinal, h)
plot(decay.rk4.sol$t,decay.rk4.sol$y, xlab="t", ylab="y",
     main="RK4 Numerical Solution for the decay ode dy/dt=-ky")
par(new=T)
lines(seq(t0,tfinal,len=100),
      exact.y, col="red")
```



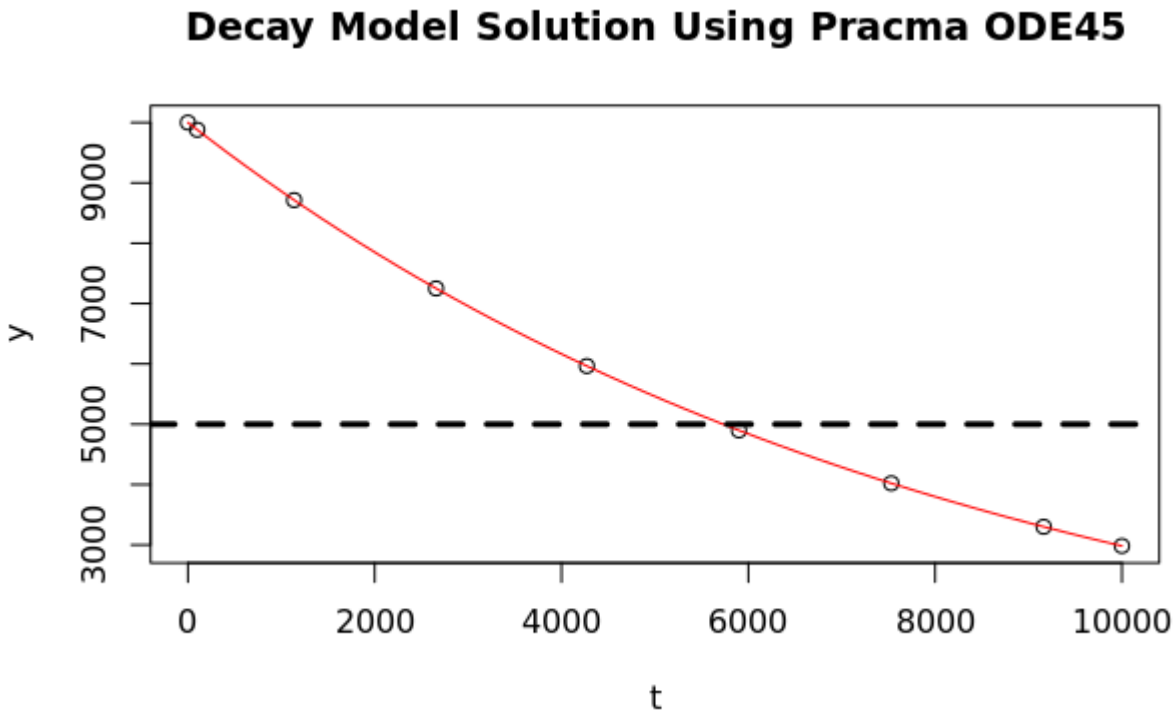

```
# Error
rk4.err <- abs(decay.rk4.sol$y - exact.y.slice)
rk4.log.err <- log(rk4.err, 10)
# Unsophisticated way to account for the -Inf error in the logscale
rk4.log.err[which(!is.finite(rk4.log.err))] <- NA

plot(seq(t0+h,tfinal,len=11), rk4.log.err,
     xlab="t", ylab="error (logscale)", type="o",
     main = "Error from 4th-Order Runge-Kutta Solution vs Exact")
```



Comparing the exact solution (in red) to the Euler solution and the Runge-Kutta solution shows a noticeable difference in error. When plotting the error of each, it is observable that the Runge-Kutta is substantially more accurate.

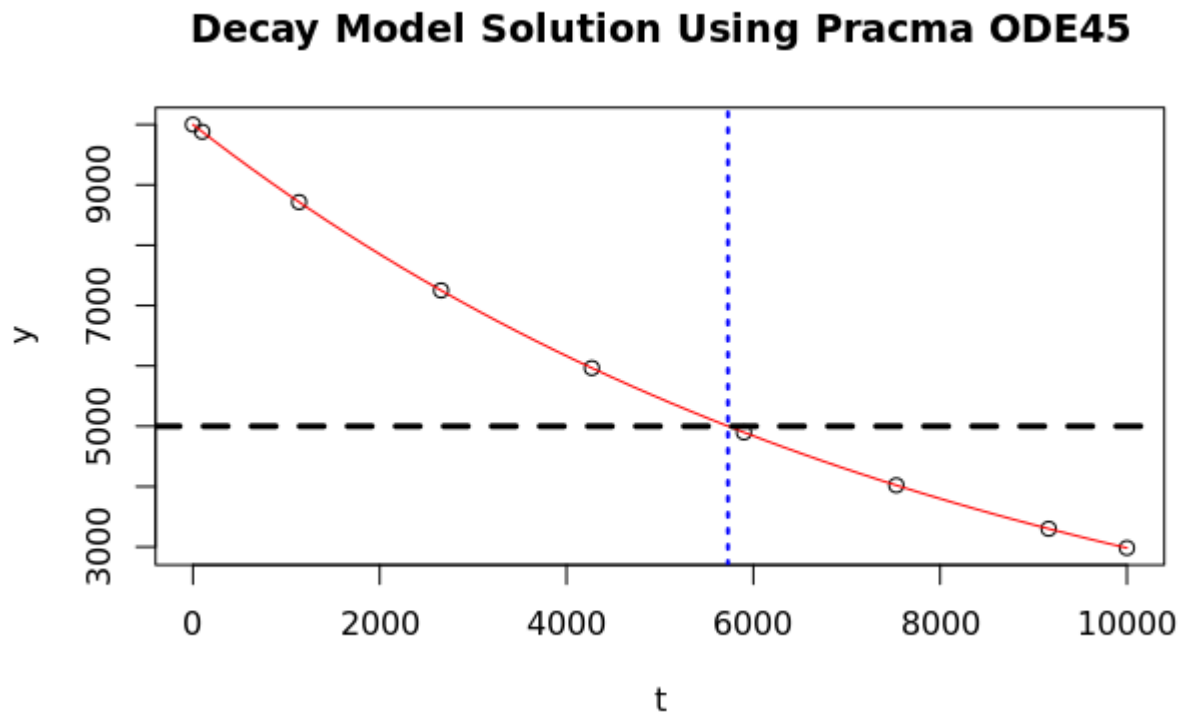
3.)

```
## 3. Use library function ode45 to solve the decay numerically
if (!require("pracma")) install.packages("pracma")
library(pracma)
# specify intial conds and params
k <- 1.21e-4
tmin <- 0
tmax <- 10000
y.init <- 10000

decay.ode45.sol <- ode45(f=function(t,y){decay.f(t,y,k=k)},
                         y=y.init, t0=tmin, tfinal=tmax)
plot(decay.ode45.sol$t,decay.ode45.sol$y, xlab="t", ylab="y",
     main="Decay Model Solution Using Pracma ODE45")
par(new=T)
lines(seq(tmin,tmax,len=50),
      y.init*exp(-k*seq(tmin,tmax,len=50)), col="red")
abline(h=y.init/2, lty="dashed", lwd=3)
```




```
#Solve for half-life with bisection
findZeroBisect <- function(func, xl, xr, tol, maxsteps=1e6){
  steps <- 0
  if (func(xl) * func(xr) >= 0){
    stop("Bad interval: try again with different endpoints")
  }

  repeat{
    xm <- (xl + xr)/2
    if(abs(func(xm)) < tol || steps >= maxsteps){
      break
```

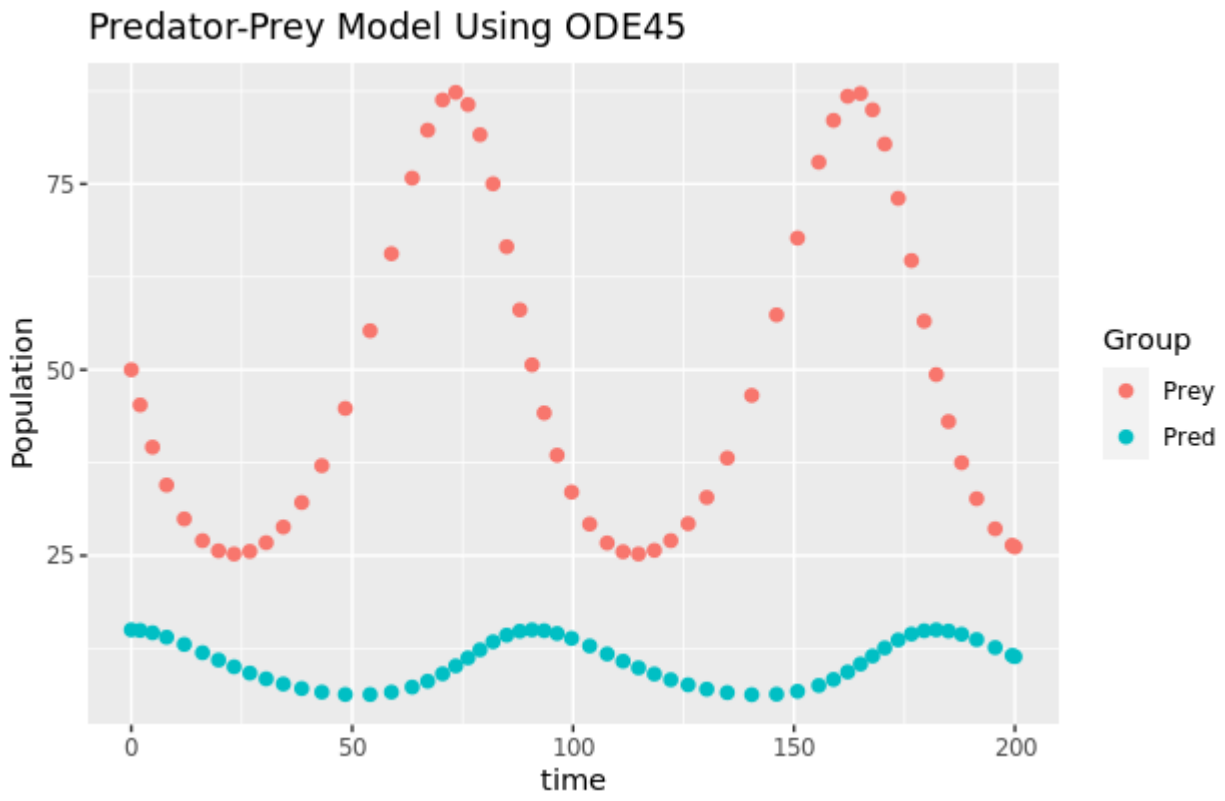
```
    }

    ifelse(func(xl) * func(xm) > 0, xl<-xm, xr<-xm)
    steps <- steps + 1
  }
  return(c(xm, func(xm), steps))
}
fun.hl <- function(x) {-5000+(y.init)*exp(-k*x)}

halflife <- findZeroBisect(fun.hl, 5000, 6500, 1e-8)[1]
abline(v=halflife, lty="dotted", lwd=2, col="blue")
```



> halflife
[1] 5728.489

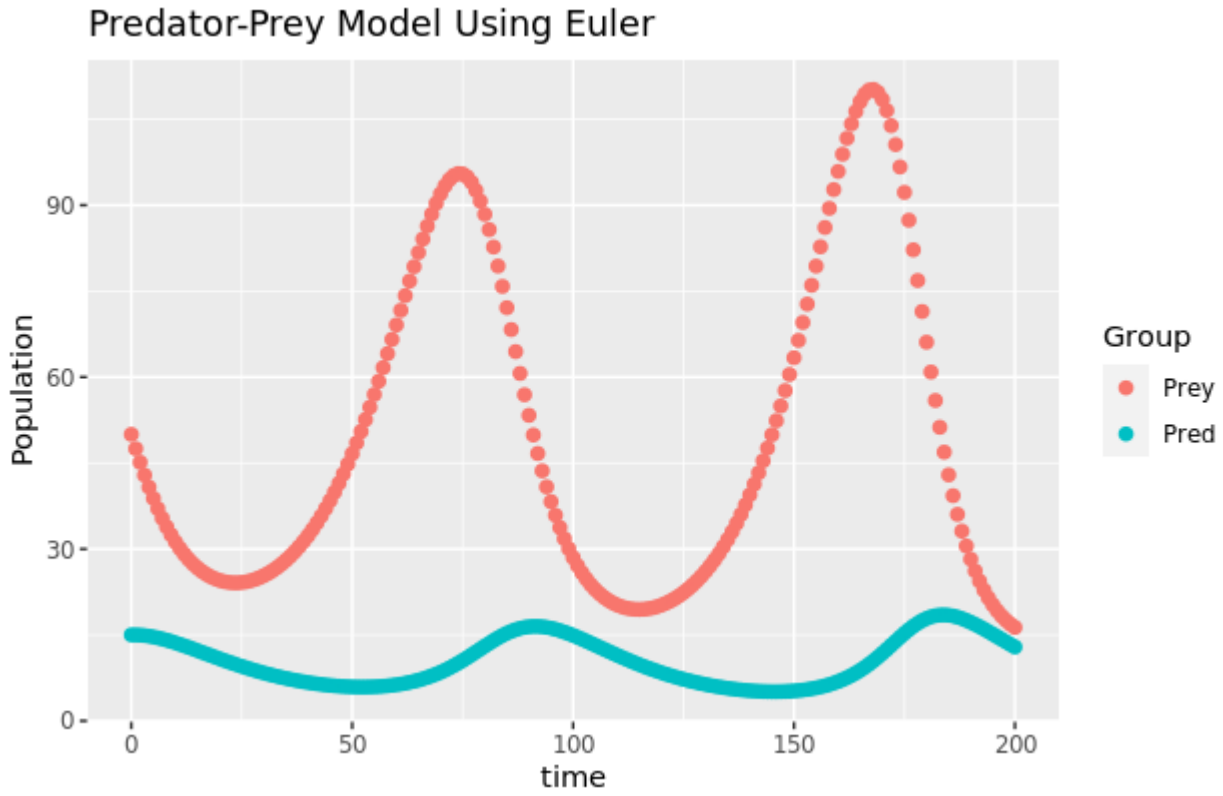
4.)

```
## 4. Solve the predator-prey model numerically
# pred-prey ode system with ode45
pp.f <- function(t,y,k){
  # k is a vector of model params
  # y is a vector of length equal to the number of odes
  prey <- y[1]
  pred <- y[2]
  dPrey <- -k[1]*prey*pred + k[2]*prey
  dPred <- k[3]*prey*pred - k[4]*pred
  return(as.matrix(c(dPrey, dPred)))
}

# a) k1=0.01, k2=0.1, k3=0.001, k4=0.05, prey(0)=50, pred(0)=15. t=0 -> 200
tmin <- 0; tmax<-200;
pp.params <- c(.01, .1, .001, .05)
prey0 <- 50
pred0 <- 15

pp.sol <- ode45(f = function(t,y){pp.f(t,y,k=pp.params)},
                y = c(prey0, pred0),
                t0 = tmin, tfinal = tmax)

# plot
if (!require("reshape")) install.packages("reshape")
library(reshape)
if (!require("ggplot2")) install.packages("ggplot2")
library(ggplot2)

plot.pred.prey <- function(sol, method){
  # - pred/prey columns melted to 1 column called "value"
  sol.melted <- melt(data.frame(time=sol$t,
                                Prey=sol$y[,1],
                                Pred=sol$y[,2]),
                     id = "time") # melt based on time
  # New column created with variable names, called “variable”
  colnames(sol.melted)[2] <- "Group" # used in legend

  g <- ggplot(data = sol.melted,
              aes(x = time, y = value, color = Group))
  g <- g + geom_point(size=2)
  g <- g + xlab("time") +  ylab("Population")
  g <- g + ggtitle(paste("Predator-Prey Model Using", method))
  show(g)
}

plot.pred.prey(pp.sol, "ODE45")
```

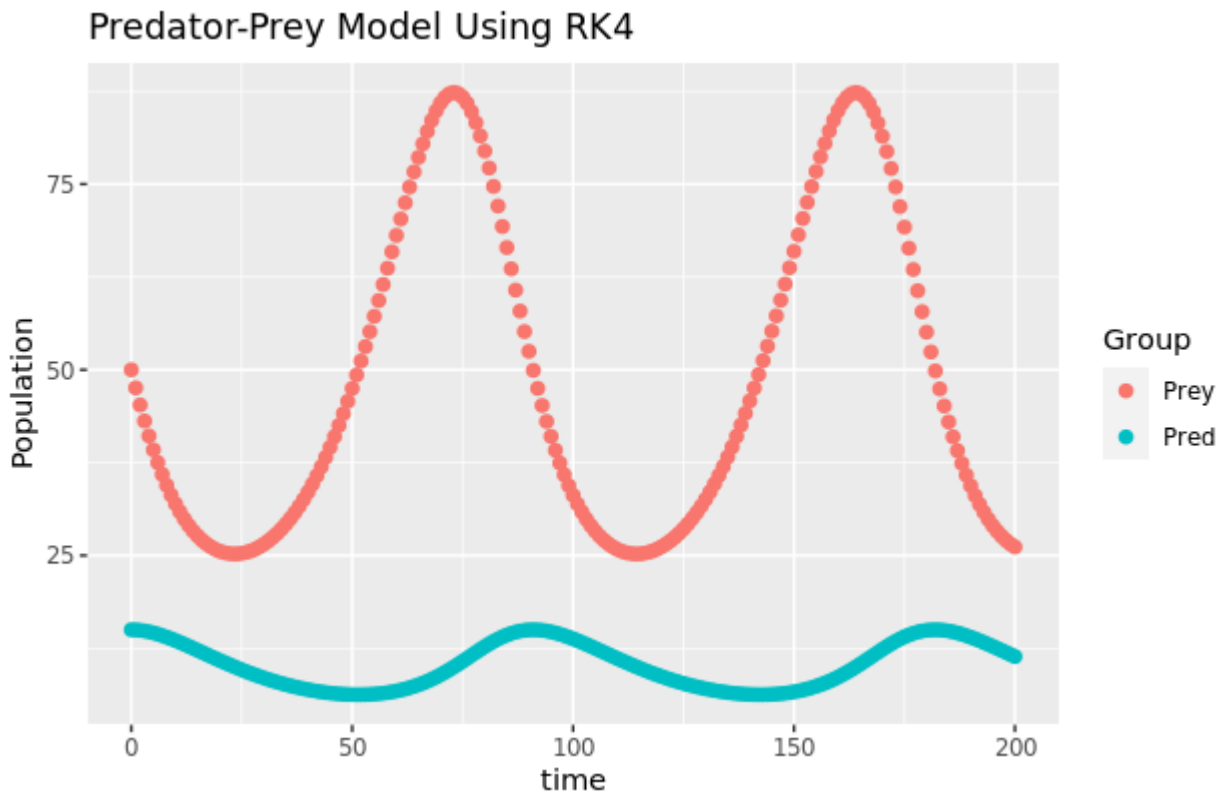



```
# b) Use Euler and RK to solve with h=10
h <- 10

my_euler2 <- function(f, y0, t0, tfinal, dt){
  # follow ode45 syntax, dt is extra
  # y0 is a vector of initial conditions
  # this y determines the number of dependent variables to solve for
  tspan <- seq(t0,tfinal,dt)
  npts <- length(tspan)
  nvars <- length(y0)
  # intialize, this will be a matrix for systems
  y <- matrix(0,nrow=npts, ncol=nvars)
  y[1,] <- y0
  for (i in seq(2,npts)){
    y_prev <- y[i-1,]    # vector at previous time
    dy <- f(t[i-1],y_prev)*dt    # also a vector/matrix
    y[i,] <- y_prev + dy         # Euler update
  }
  return(list(t=tspan, y=y))
}

my_rk4_2 <- function(f, y0, t0, tfinal, h){
  tspan <- seq(t0,tfinal,h)
  npts <- length(tspan)
  nvars <- length(y0)
  # intialize, this will be a matrix for systems
  y <- matrix(0,nrow=npts, ncol=nvars)
  y[1,] <- y0
  for (i in seq(2,npts)){
```

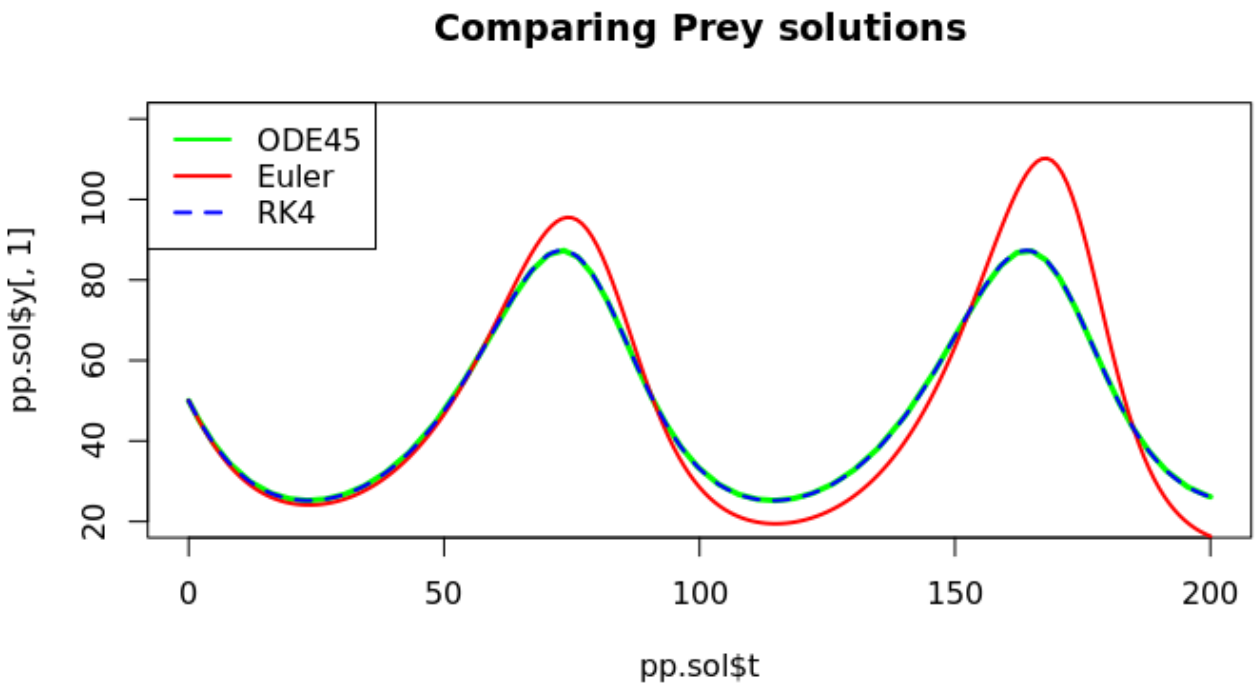
```
    y_vec_prev <- y[i-1,] # both y values at previous time point

    k1 <- f(t[i-1], y_vec_prev)
    k2 <- f(t[i-1] + 0.5 * h, y_vec_prev + 0.5 * k1 * h)
    k3 <- f(t[i-1] + 0.5 * h, y_vec_prev + 0.5 * k2 * h)
    k4 <- f(t[i-1] + h, y_vec_prev + k3*h)

    y[i,] <- y_vec_prev + (h/6)*(k1 + 2*k2 + 2*k3 + k4)
  }
  return(list(t=tspan,y=y))
}

pp.euler.sol <- my_euler2(f = function(t,y){pp.f(t,y,k=pp.params)},
                          y = c(prey0, pred0),
                          t0 = tmin, tfinal = tmax, dt=h)
plot.pred.prey(pp.euler.sol, "Euler")
```



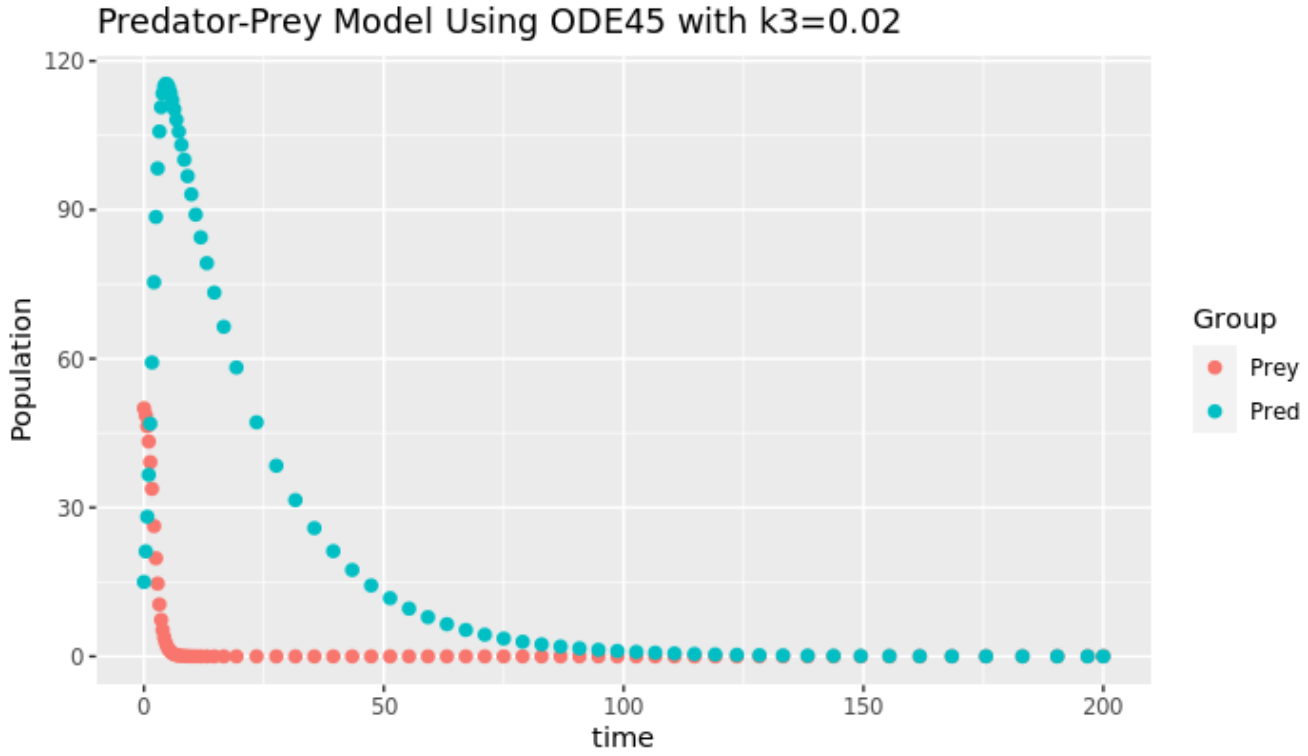

```
pp.rk4.sol <- my_rk4_2(f = function(t,y){pp.f(t,y,k=pp.params)},
                       y = c(prey0, pred0),
                       t0 = tmin, tfinal = tmax, h)
plot.pred.prey(pp.rk4.sol, "RK4")
```



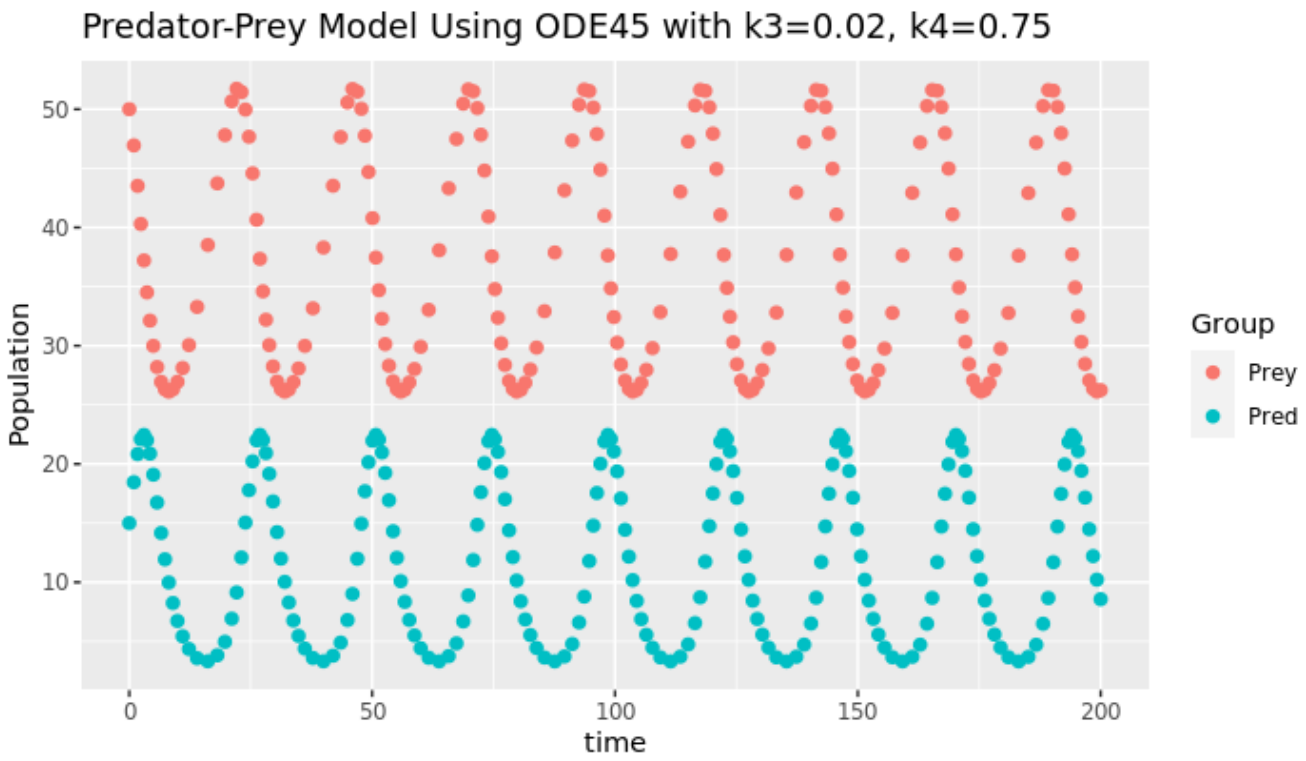

```
# plot comparing Prey solutions to ode45
plot(pp.sol$t, pp.sol$y[,1], type="l", col="green", lwd=3, ylim=c(20,120),
     xlim=c(0,200), main="Comparing Prey solutions")
par(new=T)
lines(pp.euler.sol$t,pp.euler.sol$y[,1], col="red", lwd=2)
par(new=T)
lines(pp.rk4.sol$t,pp.rk4.sol$y[,1], col="blue", lty="dashed", lwd=2)
legend(x = "topleft",
       legend = c("ODE45", "Euler", "RK4"),
       lty = c(1, 1, 2),
       col = c("green", "red", "blue"),
       lwd = 2)
```




```
# c) Use k3=0.02
pp.params.c <- c(.01, .1, .02, .05)
pp.sol.c <- ode45(f = function(t,y){pp.f(t,y,k=pp.params.c)},
                  y = c(prey0, pred0),
                  t0 = tmin, tfinal = tmax)
plot.pred.prey(pp.sol.c, "ODE45 with k3=0.02")
```



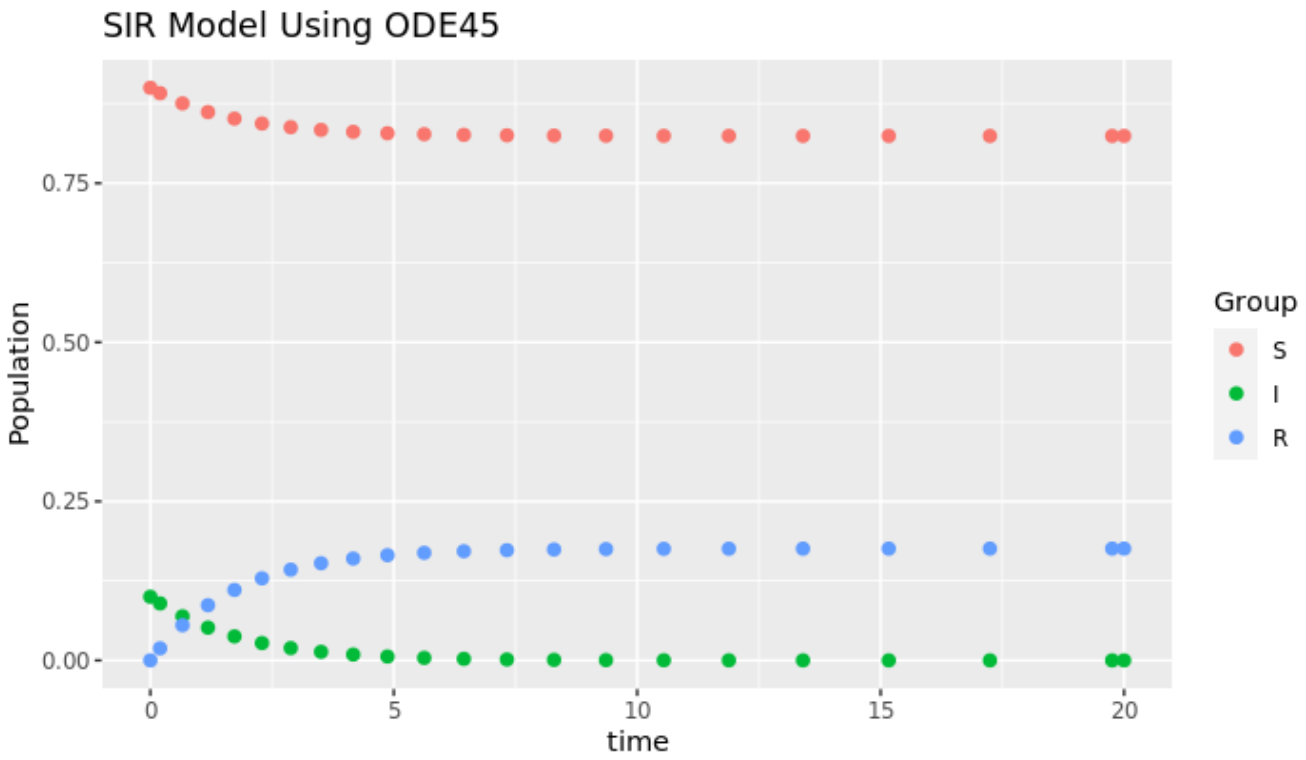
```
pp.params.c2 <- c(.01, .1, .02, .75)
pp.sol.c2 <- ode45(f = function(t,y){pp.f(t,y,k=pp.params.c2)},
                   y = c(prey0, pred0),
                   t0 = tmin, tfinal = tmax)
plot.pred.prey(pp.sol.c2, "ODE45 with k3=0.02, k4=0.75")
```



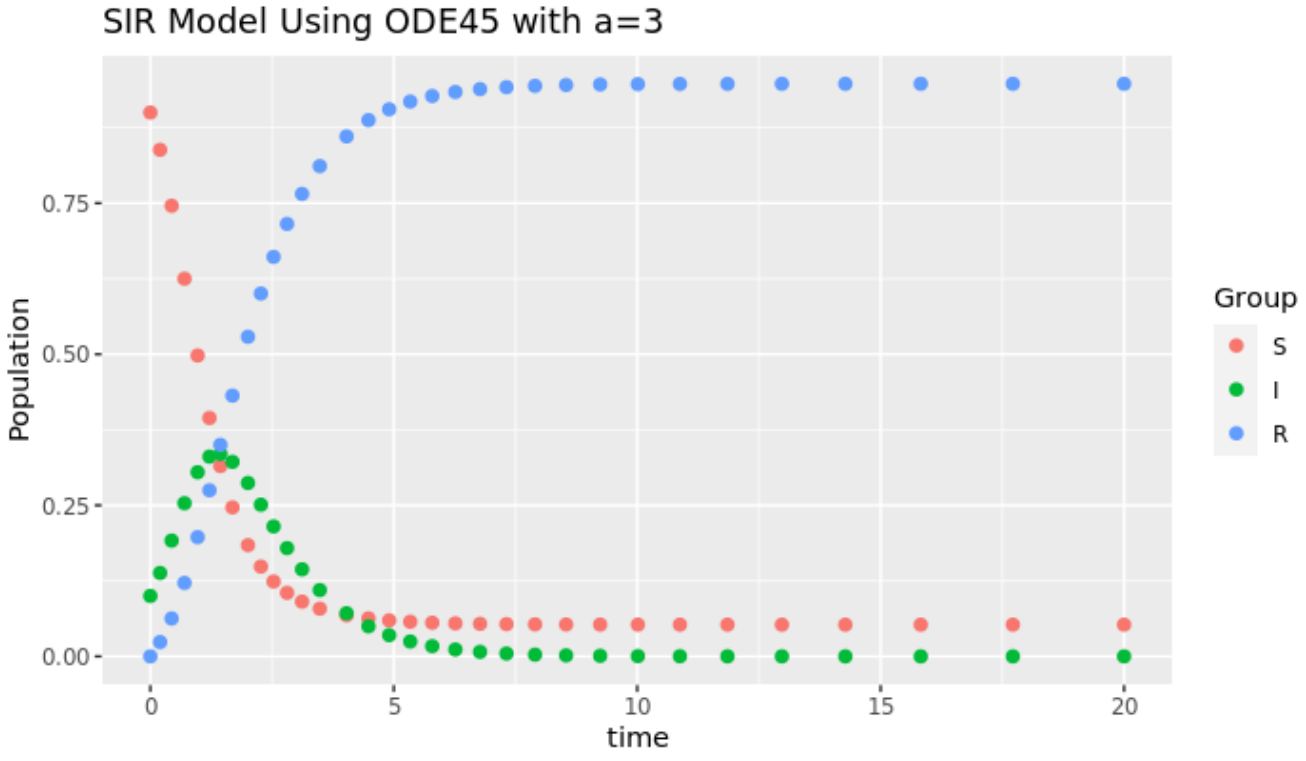
Not balanced, but better.

5.)

```
## 5. Solve SIR model numerically from t=0 -> 20
sir.f <- function(t, y, k) {
  dS <- -k[1] * y[1] * y[2]
  dI <-  k[1] * y[1] * y[2] - k[2] * y[2]
  dR <-  k[2] * y[2]
  return(as.matrix(c(dS, dI, dR)))
}

# a) a=0.5, b=1, S(0)=0.9, I(0)=0.1, R(0)=0
sir.params <- c(0.5, 1)
S0 <- 0.9
I0 <- 0.1
R0 <- 0.0
tmin <- 0
tmax <- 20

sir.ode.sol <- ode45(f = function(t,y){sir.f(t,y,k=sir.params)},
                y = c(S0, I0, R0),
                t0 = tmin, tfinal = tmax)

# plot
plot.sir <- function(sol, method){
  # - pred/prey columns melted to 1 column called "value"
  sol.melted <- melt(data.frame(time=sol$t,
                                S=sol$y[,1],
                                I=sol$y[,2],
                                R=sol$y[,3]),
                        id = "time") # melt based on time
  # New column created with variable names, called “variable”
  colnames(sol.melted)[2] <- "Group" # used in legend

  g <- ggplot(data = sol.melted,
              aes(x = time, y = value, color = Group))
  g <- g + geom_point(size=2)
  g <- g + xlab("time") +  ylab("Population")
  g <- g + ggtitle(paste("SIR Model Using", method))
  show(g)
}

plot.sir(sir.ode.sol, "ODE45")
```



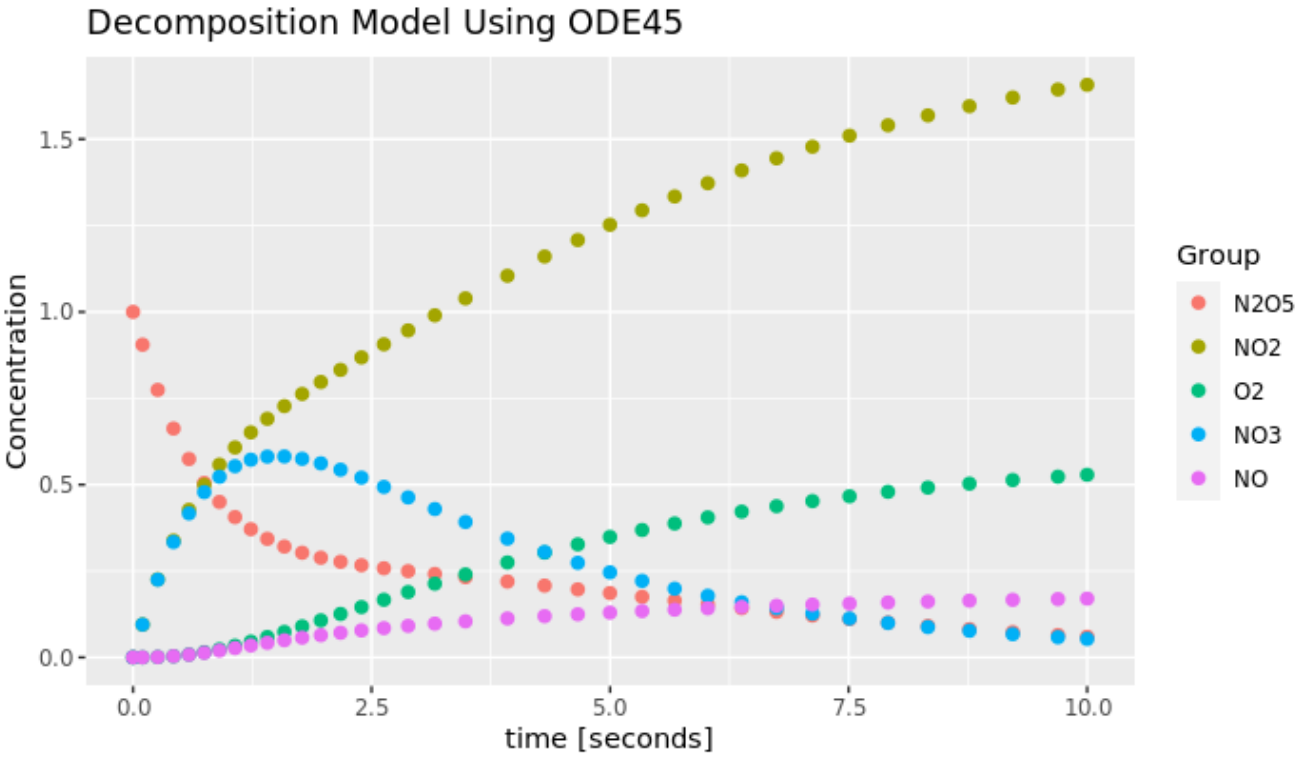

```
# b) a=3
sir.params.b <- c(3, 1)
sir.ode.sol.b <- ode45(f = function(t,y){sir.f(t,y,k=sir.params.b)},
                       y = c(S0, I0, R0),
                       t0 = tmin, tfinal = tmax)
plot.sir(sir.ode.sol.b, "ODE45 with a=3")
```



The infected group quickly rises to 0.35 at t=1.43, and decreases back to 0 where it remains for the duration of the time modeled. The susceptible group decreases at this time, and the recovered group rises.

6.)

```
## 6. Decomp of dinitrogen pentoxygen into nitrogen dioxide and molecular oxygen
decomp.f <- function(t, y, k) {
  dA <- -k[1]*y[1] + k[2]* y[2]*y[4]
  dB <- k[1]*y[1] - k[2]*y[2]*y[4] + 2*k[4]*y[4]*y[5]
  dC <- k[3]*y[2]*y[4]
  dD <- k[1]*y[1] - k[2]*y[2]*y[4] - k[3]*y[2]*y[4] - k[4]*y[4]*y[5]
  dE <- k[3]*y[2]*y[4] - k[4]*y[4]*y[5]
  return(as.matrix(c(dA, dB, dC, dD, dE)))
}

# a) k1=1.0, k2=0.5, k3=0.2,k4=1.5, [N2O5]o=1, all other IC's=0, t=0 -> 10
decomp.params <- c(1.0, 0.5, 0.2, 1.5)
A0 <- 1
B0 <- 0; C0 <- 0; D0 <- 0; E0 <- 0;
tmin <- 0
tmax <- 10

decomp.ode.sol <- ode45(f = function(t,y){decomp.f(t,y,k=decomp.params)},
                        y = c(A0, B0, C0, D0, E0),
                        t0 = tmin, tfinal = tmax)

# plot
plot.decomp <- function(sol, method){
  # - pred/prey columns melted to 1 column called "value"
  sol.melted <- melt(data.frame(time=sol$t,
                                N2O5=sol$y[,1],
                                NO2=sol$y[,2],
                                O2=sol$y[,3],
                                NO3=sol$y[,4],
                                NO=sol$y[,5]),
                        id = "time") # melt based on time
  # New column created with variable names, called “variable”
  colnames(sol.melted)[2] <- "Group" # used in legend

  g <- ggplot(data = sol.melted,
              aes(x = time, y = value, color = Group))
  g <- g + geom_point(size=2)
  g <- g + xlab("time [seconds]") +  ylab("Concentration")
  g <- g + ggtitle(paste("Decomposition Model Using", method))
  show(g)
}

plot.decomp(decomp.ode.sol, "ODE45")
```




```
# b) Increase k4 to make the intermediate species -> 0
k4.mono.table <- matrix(nrow = 0, ncol = 2)
for(k in 10^(seq(-1, 4, 1))){
  k4 <- k
  decomp.params.b <- c(1.0, 0.5, 0.2, k4)
  decomp.ode.sol.b <- ode45(f = function(t,y){decomp.f(t,y,k=decomp.params.b)},
                        y = c(A0, B0, C0, D0, E0),
                        t0 = tmin, tfinal = tmax)
  #plot.decomp(decomp.ode.sol.b, paste("ODE45 Using k4 =", k4))
  # Is NO monotonically increasing?
  mono.check <- all(decomp.ode.sol.b$y[,5] == cummax(decomp.ode.sol.b$y[,5]))
  k4.mono.table <- rbind(k4.mono.table, c(k4, mono.check))
}

k4.mono.table
```

```
> k4.mono.table
      [,1] [,2]
[1,] 1e-01    1
[2,] 1e+00    1
[3,] 1e+01    1
[4,] 1e+02    1
[5,] 1e+03    1
[6,] 1e+04    0
```

NO stops monottonically increasing at k=1e04

Setting dE/dt=0 in the model is not a good idea since it is not accurate to the decomposition. NO is the hypothesized intermediate, and its change is dependent on the other steps of the reaction. Fully removing dE/dt ignores the reaction process and sets it to the ideal, which may not be the case.