```
# Make.R --> Wrapper script to run simulations
rm(list=ls())
require(plyr)
require(lattice)

# load parameters, functions
    source("Parameters.R")
    source("Functions.R")

# set up different parameter combinations for all simulations

sims <- data.frame(model =
 c("noThresh","noThresh","noThresh","noThresh","noThresh","Thresh"),
                    MPA = c("cons","cons","fish","fish","null","null"),
                    effort_allocate = c(NA, "yes", NA, "yes", NA, NA),
 stringsAsFactors=FALSE)

for(run in 1:nrow(sims)){
# run analysis
    # choose threshold or no threshold
    model = sims$model[run] # "noThresh"; "Thresh"

    # if no threshold, choose MPA: "null", "cons", "fish"
    MPA = sims$MPA[run]
        # choose how effort should be allocated if MPAs present.
        effort_allocate = sims$effort_allocate[run]
        effort_allocate = ifelse(MPA!="null",effort_allocate,NA)

# analysis
timed <- system.time(
    if(model=="noThresh")
 {sapply(c("Parameters_nothresh.R","Sim_noThresh.R"),source,.GlobalEnv)} else {
        if(model=="Thresh" & MPA == "null")
 {sapply(c("Parameters_thresh.R","Sim_thresh.R"),source,.GlobalEnv)} else {
            warning("model needs to be 'noThresh' or 'Thresh', MPA needs to be 'null'")
            }
        }
    )

cat(paste("Time elapsed: ",round(timed[1]/3600,3)," hours\n","Finished running a ", model, "
 simulation with ", MPA, " MPAs", " and effort re_allocate set to ", effort_allocate,".
 \n",nrow(sims)-run, " simulations left to go...",sep=""))
}
#-----------------------------------------------------------------------#
# Functions.R ---> defines reproduction, dispersal, and climate velocity functions used in
 simulations

#Laplace dispersal kernel
k<-function(x,y,b) return(1/2*b*exp(-b*abs(x-y)))

# dispersal matrix
d<-array(,c(w,w))
#dispersal probabilities to point i from every point j
for(i in 1:w) d[i,]=k(world[i],world,b)
```

```
#Beverton-Holt recruitment
f<-function(n,R0,K) return(R0*n/(1+(R0-1)/K*n))

#Standard error removing NAs
stderr <- function(x) sqrt(var(x,na.rm=TRUE)/length(na.omit(x)))

moveMPA <- function(MPA.current = MPA.current, displaced = displaced, mpa.yes=mpa.yes,
 mpa.no=mpa.no, world){

  ##### Move MPAS
  # move MPA forward by <displaced> amount
  next_MPA = MPA.current[displaced:length(MPA.current)]
  lost <- MPA.current[1:(displaced-1)]

  # are there any MPAs in <next_MPA>?
  #any(test[[i]]$next_MPA==1)
  # IF FALSE, then need to figure out how many 0s were lost in
  # move, and make sure to preserve interval of zeros == mpa.no,
  # then fill in mpa.yes,mpa.no to length of world. But also this
  # is only for when mpa.no exists on both sides of MPA_next. If
  # exactly to the edge of one reserve is lost, should shift down
  # to next if statement
  if(any(next_MPA==1)==FALSE & any(lost==1)==FALSE){
    # these are the intervals that are left behind as world moves
    # forward
    lost <- MPA.current[1:(displaced-1)]
    # this is the last continuous chunk of numbers at the end of
    # <lost>
    length_last <- rep(tail(rle(lost)$value,1),
        tail(rle(lost)$lengths,1))
    # want to know how long <length_last> is so can make sure to
    # get correct interval
    L_int <- length(length_last)
    # how many more zeros do we need before we start with mpa.yes
    # again?
    zero_add <- sum(mpa.no==0) - L_int - sum(next_MPA==0)
    # this is what needs to be appended to <next_MPA>
    new_MPA <- c(rep(0,zero_add),
        rep(c(mpa.yes,mpa.no),
        length.out=length(world)))

  }else{

    # IF FALSE (there are some 1s) then we only care about the
    # last interval of the <next_MPA>, is that protected or not?
    last_step = tail(next_MPA,1)
    # IF <last_step>==1
    if(last_step ==1){
      #then how many 1s are at the end of the <next_MPA> section?
      end_step = rep(tail(rle(next_MPA)$value,1),
         tail(rle(next_MPA)$lengths,1))
         # number of 1s at very end of lost interval
      # length of <end_step>
```

```r
        end_int = length(end_step)
        # need to prepend sum(mpa.yes==1) - <end_step> to beginning
        prepend = rep(1, (sum(mpa.yes==1) - end_int))
        # and then fill out with mpa.no,mpa.yes for length of world
        fillOut <- rep(c(mpa.no, mpa.yes),
          length = (length(world) - length(prepend)))
      }else{
        # IF <last_step>==0
        end_step = rep(tail(rle(next_MPA)$value,1),
          tail(rle(next_MPA)$lengths,1))
        # number of 0s at very end of lost interval
        # length of <end_step>
        end_int = length(end_step)
        # need to prepend sum(mpa.no==0) - <end_step> to beginning
        prepend = rep(0, (sum(mpa.no==0) - end_int))
        # and then fill out with mpa.no,mpa.yes for length of world
        fillOut <- rep(c(mpa.yes, mpa.no),
          length = (length(world) ))
      }
      new_MPA = c(prepend,fillOut)
    }

    MPA_finish = c(next_MPA,new_MPA)
    MPA_finish = MPA_finish[1:length(world)]
    # reduce to just the size of the world

    return(MPA_finish)
  }

  m <- function(n, s, Fthresh = NA, Fharv = NA, mpa.yes = NA,
          mpa.no = NA, MPA.current=NA,effort_re_allocate=NA){
      # steps
      # 1. Harvest (check for thresholds, harvesting, MPA coverage)
      # 2. Patch moves (and MPAs are adjusted)
      # 3. Individuals outside patch die
      # 4. Individuals still alive (ie inside the patch) reproduce

      # harvesting occurs first - check to see how should
      # re-allocate effort

      if(!is.na(effort_re_allocate) & !is.na(Fharv)){ # harvesting non-zero and effort
   reallocate
          #total_catch = sum(n)*Fharv
          total_catch = sum(n)*Fharv * 1.5 # increasing effort by 50% to account for effort-
   reallocation
          available_total_pop = sum(n[which(MPA.current==0)])
          # pop with no MPA coverage
          available_fish = rep(0,length(n))
          available_fish[MPA.current==0] <-
              n[MPA.current==0]/available_total_pop
              # available_total_pop
          # proportion at each point
          catch_in_space <- total_catch*available_fish
          # allocate catch
          next_gen = n-catch_in_space
          next_gen[next_gen<0] = 0
      }else{
          if(!is.na(Fthresh)) { # if thresholds
              next_gen = ifelse(n < Fthresh,
                  n, n - (n - Fthresh) * Fharv)
              }
          if(!is.na(Fharv) & is.na(Fthresh)) {
              # if harvesting, no thresholds
              next_gen = n*(1-Fharv)
              }
          if(is.na(Fharv) & is.na(Fthresh)) {next_gen = n}
          # if no harvesting of any kind

          # but put fish back if places that were harvested were in
          # the MPA
           next_gen[MPA.current == 1] <- n[MPA.current == 1]
       }

      # move the patch
      # calculate how far the patch will move through the population
      # (if speed !=0)
      displaced = ifelse(s>0,s/step_size,1)

      # assign population that will still be inside the patch to
      #  moved patch
      next_n = next_gen[displaced:length(next_gen)]

      # fill in newly existing patch with 0s
      next_n = c(next_n,rep(0,length.out=(displaced-1)))

  # move MPAs?
    if(s > 0){MPA_finish = moveMPA(MPA.current, displaced,
           mpa.yes,mpa.no,world)}else{MPA_finish= MPA.current}

      # let patch reproduce
      next_patch = vector(mode="numeric",length(world))

      # keep individuals still in patch + those now in it due to
      # move
      next_patch[1:length(patch)] = next_n[1:length(patch)]

      babies = next_patch*f_ind
      n2 = babies %*% d *step_size
      n2 = sapply(n2,f,R0,K)

        MPA = MPA_finish
      return(list(n2,MPA)) # removed a mysterious 'harv' from here
  }

  # wrapper function to run simulation for 6000 generations and save outcome from 2000
   additional generations to take average
  longRun <- function(s, mpa.yes, mpa.no, Fthresh, Fharv, init,
       MPA.start, generations_total, generations_av,
       effort_re_allocate=effort_allocate){
      MPA.current <- MPA.start
      burn_in <- generations_total - generations_av
      for(t in 1:(burn_in)){
          output = m(n=init, s = s, Fthresh=Fthresh,Fharv=Fharv,
           mpa.yes = mpa.yes, mpa.no = mpa.no,
           MPA.current = MPA.current, effort_re_allocate=effort_allocate)
          init= output[[1]]
          MPA.current = output[[2]]
      }

      # make dataframe for simulation average
      pop <- rep(0,generations_av)
      for(keep in 1:generations_av){
          output = m(n=init, s = s, Fthresh=Fthresh,Fharv=Fharv,
           mpa.yes = mpa.yes, mpa.no = mpa.no,
           MPA.current = MPA.current, effort_re_allocate = effort_allocate)
          init = output[[1]]
          MPA.current = output[[2]]
          pop[keep] = sum(output[[1]])

      }

      # take mean for equil_abundance
      equil.pop = mean(pop)
      equil.sd = sd(pop)

      return(list(equil.pop,equil.sd))
  }

  # to introduce population to empty landscape, harvesting before
  #  adding speed treatment

  # wrapper function to initialize the population, only returns results from final generation
  startUp <- function(s, mpa.yes, mpa.no, Fthresh, Fharv, init,
       MPA.start, burn_in,effort_re_allocate=effort_allocate){
      MPA.current <- MPA.start
      for(t in 1:(burn_in)){
          output = m(n=init, s = s, Fthresh=Fthresh,Fharv=Fharv,
           mpa.yes = mpa.yes, mpa.no = mpa.no,
           MPA.current = MPA.current,
           effort_re_allocate=effort_allocate)
          init= output[[1]]
          MPA.current = output[[2]]
      }
      return(list(init,MPA.current))
  }
  #-------------------------------------------------------------------------#
  # Parameters.R
  #######################################
  ## Parameters & Building Structures ##
  ######################################
```

```
step_size=0.01 #distance between points in space
b=.5 #parameter for Laplace dispersal kernel
R0=5 #growth parameter for recruitment
K=100 #carrying capacity parameter for juvenile density dependence
threshold = 0.001 #difference between generation populations.
burn_in = 2000 # number of generations to run simulations before checking for equilibrium
 conditions
speeds = seq(0,.5,by=0.02)
harvests = seq(0,.2,by=0.01)
f_ind = 1 #per capita reproductive rate
generations_total = 8000
generations_av = 2000

patch = seq(0,1,by=step_size)
world = seq(-.51,4.5, by = step_size)   # to run the MPA versions, world has to be at least
 400 steps (max distance between MPAs in "cons" run)
w = length(world)

cons.yes = rep(1,4*b/step_size)
cons.no = rep(0,8*b/step_size)
fish.yes = rep(1,floor((1/3*b)/step_size))  # had to round because not complete step size.
 Rounded down.
fish.no = rep(0,floor((2/3*b)/step_size))

null.yes = rep(0,length(world))
null.no = rep(0, length(world))

move_window = 100
#-------------------------------------------------------------------------#
# Parameters_nothresh.R
# parameters for no-threshold simulations (just proportional harvesting)

# build dataframes
    summaries <- data.frame(
        Equil.pop = rep(NA,length=length(speeds)*length(harvests)),
        Equil.sd = rep(NA,length=length(speeds)*length(harvests)),
        speed = rep(NA,length=length(speeds)*length(harvests)),
        harvest = rep(NA,length=length(speeds)*length(harvests)),
        thresh = rep(NA,length=length(speeds)*length(harvests))
        )


# index for row number
    rownumber <- matrix(seq(1:(length(harvests)*length(speeds))),ncol=length(speeds))
#-------------------------------------------------------------------------#
# Parameters_thresh.R
# parameters for threshold simulations (no proportional harvesting)
    harvests = 1
    thresholds = seq(0,1,by=0.1)

# build dataframes
    summaries <- data.frame(
        Equil.pop = rep(NA,length=length(speeds)*length(thresholds)),
        Equil.sd = rep(NA, length=length(speeds)*length(thresholds)),
        speed = rep(NA,length=length(speeds)*length(thresholds)),
        harvest = rep(NA,length=length(speeds)*length(thresholds)),
        thresh=rep(NA,length=length(speeds)*length(thresholds)))


# index for row number
    rownumber <- matrix(seq(1:(length(thresholds)*length(speeds))),ncol=length(speeds))
#------------------------------------------------------------------------#
# Sim_noThresh.R
# runs simulations in which there is no threshold management, MPAs are possible

# set MPAs
if(MPA=="cons") {mpa.yes=cons.yes; mpa.no=cons.no} else {
    if(MPA=="fish") {mpa.yes=fish.yes; mpa.no=fish.no} else {
        if(MPA=="null") {mpa.yes=null.yes; mpa.no=null.no} else{
            if(exists("MPA")) warning(paste("MPA needs to be 'cons', 'fish', or
 'null'.",sep=""))
        }
    }
}

# initializing the population with no pressure (no harvesting, no climate)
    init<-rep(0,w) # rows are world, columns are time
    init[which(patch==0.55)]=50
    MPA.start = rep(c(mpa.yes,mpa.no),length.out=length(world))

    output <- startUp(s=0,mpa.yes=mpa.yes,mpa.no=mpa.no,burn_in=burn_in, Fharv=NA,
 Fthresh=NA, init=init, MPA.start = MPA.start, effort_re_allocate=NA)
    init.s <- output[[1]]
    MPA.start <- output[[2]]

for(q in 1:length(speeds)){
    for(j in 1:length(harvests)){
        # adding harvesting
            output <-
 startUp(s=0,mpa.yes=mpa.yes,mpa.no=mpa.no,burn_in=burn_in,Fharv=harvests[j],Fthresh=NA,
 init=init.s, MPA.start = MPA.start, effort_re_allocate=effort_allocate)
            init.h <- output[[1]]
            MPA.start <- output[[2]]

        # adding speed
            output <- longRun(s=speeds[q], mpa.yes=mpa.yes, mpa.no=mpa.no, Fthresh=NA,
 Fharv=harvests[j], init = init.h, MPA.start = MPA.start,
 generations_total=generations_total, generations_av=generations_av,
 effort_re_allocate=effort_allocate)

        # save output
            pop = output[[1]]
            pop.sd = output[[2]]
            summaries[rownumber[j,q],] <- c(pop, pop.sd, speeds[q], harvests[j],
 ifelse(exists("Fthresh"), Fthresh,NA))
    }
}
```

```
write.csv(summaries,file = paste("Data/
 MPA",MPA,"_",effort_allocate,"_",Sys.Date(),".csv",sep=""))
#------------------------------------------------------------------------#
# Sim_thresh.R
# runs simulations in which there is threshold management, MPAs are not possible

# initializing the population with no pressure (no harvesting, no climate)
    init<-rep(0,w) # rows are world, columns are time
    init[which(patch==0.55)]=50
    MPA.start = rep(c(mpa.yes,mpa.no),length.out=length(world))

    output <- startUp(s=0,mpa.yes=mpa.yes,mpa.no=mpa.no,burn_in=burn_in, Fharv=NA,
 Fthresh=NA, init=init, MPA.start = MPA.start)
    init.s <- output[[1]]
    MPA.start <- output[[2]]

for(q in 1:length(speeds)){
    for(j in 1:length(thresholds)){
        # adding harvesting
            output <-
 startUp(s=0,mpa.yes=mpa.yes,mpa.no=mpa.no,burn_in=burn_in,Fharv=1,Fthresh=thresholds[j],
 init=init.s, MPA.start = MPA.start, effort_re_allocate=effort_allocate)
            init.h <- output[[1]]
            MPA.start <- output[[2]]Fharv=1

        # adding speed
            output <- longRun(s=speeds[q], mpa.yes=mpa.yes, mpa.no=mpa.no,
 Fthresh=thresholds[j], Fharv=1, init = init.h, MPA.start = MPA.start,
 generations_total=generations_total, generations_av=generations_av,
 effort_re_allocate=effort_allocate)

        # save output
            pop = output[[1]]
            pop.sd = output[[2]]
            summaries[rownumber[j,q],] <- c(pop, pop.sd, speeds[q], 1,
 ifelse(exists("thresholds"), thresholds[j],NA))
    }
}

write.csv(summaries,file = paste("Data/Thresh_",Sys.Date(),".csv",sep=""))
#------------------------------------------------------------------------#
```