LandMine

Eliot McIntire

18 May 2018

Overview

Landmine is a model created for simulating the natural range of variation for landscapes in the boreal forest. Written in the 1990s (Andison 1996), it has been widely used by the public and the private sector for various purposes. This SpaDES module is a rewrite of the fire component in native R.

Model Differences

The current version has not yet been fully tested and compared with the original version, but there are currently several known differences:

1. Fire sizes are taken from a Truncated Pareto distribution, resulting in numerous very small fires, and few large fires;
2. Parameters have been not been fitted to the landscapes that are under study in the LandWeb project.

Model Code

Code available from https://github.com/PredictiveEcology/LandMine.

Known Species

Landmine requires the following codes as inputs (the `genus_spec` form below), and converts and groups species as follows. Each of the species groups has its own Rate of Spread (ROS) for fire spreading:

- PINU
  - Pinu_ban
  - Pinu_con
  - Pinu_sp
- DECI
  - Betu_pap
  - Popu_bal
  - Popu_tre
  - Lari_lar
- PICE_MAR
  - Pice_mar
- PICE_GLA
  - Pice_gla
- ABIE
  - Abie_sp

Usage

To run this Landmine module alone, from an R console, the following should work (*iff* raster inputs for `rstStudyRegion` and `rstFlammable` are available), assuming all R packages are available.

Paths will have be changed for a different user.

```
library(reproducible)
```

```
## Creating a new generic function for 'cache' in package 'reproducible'
```

```
workingDir <- file.path("~/GitHub/LandWeb")
moduleDir <- file.path(workingDir, "m") %>% checkPath()
inputDir <- file.path(workingDir, "inputs") %>% checkPath(create = TRUE)
outputDir <- file.path(workingDir, "outputs") %>% checkPath(create = TRUE)
cacheDir <- file.path(outputDir, "cache") %>% checkPath(create = TRUE)
```

Package dependencies

To determine which packages are used by `LandMine`, use:

```
SpaDES.core::packages(modules = "LandMine", paths = moduleDir)
```

```
## $LandMine
## [1] "SpaDES.core"
## [2] "data.table"
## [3] "grDevices"
## [4] "magrittr"
## [5] "PredictiveEcology/SpaDES.tools@prepInputs"
## [6] "raster"
## [7] "RColorBrewer"
## [8] "VGAM"
```

Module usage

```
reproducible::Require(c("igraph", "SpaDES.core", "raster"))

times <- list(start = 0, end = 13)
parameters <- list(
  #.progress = list(type = "text", interval = 1), # for a progress bar
  ## If there are further modules, each can have its own set of parameters:
  #LandMine = list(flushCachedRandomFRI = TRUE)
)
modules <- list("LandMine")
rstFlammable <- raster(extent(0,1e5, 0, 1e5), res = 250, vals = 0)

objects <- list(rstFlammable = rstFlammable)
paths <- list(
  cachePath = cacheDir,
  modulePath = moduleDir,
  inputPath = inputDir,
  outputPath = outputDir
)
```

```
mySim <- simInit(times = times, params = parameters, modules = modules,
                 objects = objects, paths = paths)

dev()
mySimOut <- spades(mySim, .plotInitialTime = times$start, debug = TRUE)
```

Testing the burn algorithm

```
libs <- c("DEoptim", "SDMTools", "data.table", "raster", "parallel")
Require(libs)
s <- simInit(times = times, params = parameters, modules = modules,
                 objects = objects, paths = paths)
attach(s@.envir)
attach(s@.envir$LandMine)
if (FALSE) {
  detach(s@.envir)
  detach(s@.envir$LandMine)
}
#ros <- raster(extent(0,3e4, 0, 3e4), res = 250, vals = 0)
ros <- raster(extent(0,2.5e5, 0, 2.5e5), res = 250, vals = 0)
ros <- ros == 0
fireSize <- 100000
```

```
burnFun <- function(ros, centreCell, fireSize,
                    spawnNewActive, sizeCutoffs, burn1,
                    spreadProb) {

    burned <- burn1(
      landscape = ros,
      startCells = centreCell,# c(quarterCell, centreCell),
      fireSizes = fireSize,
      spreadProb = spreadProb,
      spreadProbRel = ros,
      spawnNewActive = spawnNewActive,
      sizeCutoffs = sizeCutoffs
    )
    burnedMap <- raster(ros)
    burnedMap[] <- NA

    burnedMap[burned$pixels] <- burned$initialPixels
    LM <- SDMTools::PatchStat(burnedMap, cellsize = res(burnedMap)[1])
    list(burnedMap = burnedMap, LM = LM)
}

makeParallel <- function(wantParallel, numClus, cl, funs, libs) {
  if (wantParallel) {
    library(parallel)
    if (is.null(cl)) {
      cl <- parallel::makeCluster(numClus)
    }
    funs <- c(funs)
```

```
    parallel::clusterExport(cl,  funs)
    #env <- environment()
    parallel::clusterExport(cl,  "libs", envir = parent.frame())
    parallel::clusterEvalQ(cl, {lapply(libs, library, character.only = TRUE)})
    return(cl)
  }
}

wrap <- function(cl = NULL, reps, par, burn1, burnFun, objs, libs) {
  funNames <- c(deparse(substitute(burn1)), deparse(substitute(burnFun)))
  cl <- makeParallel(TRUE,
                     numClus = if (length(cl)) length(cl) else detectCores() - 1,
                     cl, c(funNames, objs), libs)
  objs <- append(mget(objs, envir = parent.frame()), list(burn1 = burn1))
  parallel::clusterExport(cl,  "objs", envir = parent.frame())
  burnMapList <- parallel::clusterApplyLB(cl, reps, function(r) {
    do.call("burnFun", objs)
  })
  return(list(cl = cl, out = burnMapList))
}

######################################################################
fitSN <- function(sna, ros, centreCell, fireSizes = 10^(2:5),
                  desiredPerimeterArea = 0.004, burnFun, burn1) {

  sizeCutoffs <- 10^sna[5:6]
  spreadProb <- sna[7]
  sna <- c(10^(sna[1]), 10^(sna[2]), 10^(sna[3]), 10^(sna[4]))
  bfs1 <<- lapply(fireSizes, function(fireSize) {
   burnFun(ros, centreCell, fireSize,
                  sna, sizeCutoffs, burn1 = burn1, spreadProb = spreadProb)
  })
  res <- lapply(seq(bfs1), function(bfCount) {
    abs(log(bfs1[[bfCount]]$LM[1,"perim.area.ratio"]) -
          log(desiredPerimeterArea)) +
      100*(sum(bfs1[[bfCount]]$burnedMap[]) < fireSizes[bfCount])
    # it needs to get to above 90,000 HA for it to count
  })
  sum(unlist(res))# * log10(fireSizes)) # weigh larger ones more

}

######################################################################
fitSN2 <- function(par, ros, centreCell, fireSizes = 10^(2:5),
                   desiredPerimeterArea = 0.003) {
  sizeCutoffs <- 10^c(par[4], par[5])
  bfs1 <- lapply(fireSizes, function(fireSize) {
    sna <- min(-0.15, par[1] + par[2]*log10(fireSize))
    sna <- 10^c(sna*par[3], sna*2*par[3], sna*3*par[3], sna*4*par[3])
    #sna <- -1
    burnFun(ros, centreCell, fireSize,
            sna, sizeCutoffs, spreadProb)
  })
```

```
  res <- lapply(seq(bfs1), function(bfCount) {
    abs(log(bfs1[[bfCount]]$LM[1,"perim.area.ratio"]) -
          log(desiredPerimeterArea)) +
      100*(sum(bfs1[[bfCount]]$burnedMap[]) < fireSizes[bfCount])
    # it needs to get to above 90,000 HA for it to count
  })
  a <- sum(unlist(res))
  #attr(a, "bfs1") <- bfs1
  a
}
```

Optimizing parameters

The following code chunk tries to find values of `spawnNewActive` that creates "reasonable" fire shapes at all sizes.

```
wantParallel <- TRUE
spawnNewActive <- c(0.46, 0.2, 0.26, 0.11)
sizeCutoffs <- c(8e3, 2e4)
#spawnNewActive <- c(0.1, 0.04, 0.025, 0.01)
#sizeCutoffs <- c(8e3, 2e4)

NineCorners <- cellFromRowCol(ros,
                              rownr = nrow(ros) / 4 * rep(1:3, 3),
                              colnr = ncol(ros) / 4 * rep(1:3, each = 3))

centreCell <- cellFromRowCol(ros,
                             rownr = nrow(ros) / 2,
                             colnr = ncol(ros) / 2)

## Set variables
objs <- c("ros", "centreCell", "fireSize",
          "spawnNewActive", "sizeCutoffs", "spreadProb")
funs <- c("burnFun", "burn1")
libs <- c("SpaDES.tools", "data.table", "raster", "SDMTools")
```

```
########################################
### SET UP CLUSTER FOR PARALLEL
numCores <- detectCores() - 1
if (Sys.info()["sysname"] == "Windows") {
  cl <- parallel::makeCluster(numCores)
} else if (grepl("W-VIC-A1053", Sys.info()["nodename"])) {
  Require("future")
  NcoresOnEach <- 24
  # make a cluster on 3 machines
  machine1 <- "132.156.148.172"
  machine2 <- "132.156.148.171"
  machine3 <- "localhost"
  clNames <- c(rep(machine1, NcoresOnEach),
               rep(machine2, NcoresOnEach),
               rep(machine3, NcoresOnEach))
  cl <- makeClusterPSOCK(clNames,
                         homogeneous = FALSE, verbose = TRUE)
```

```
  #cl <- parallel::makeCluster(numCores, type = "FORK")
  clusterSetRNGStream(cl, sample(1e8,1))
} else {
  cl <- parallel::makeCluster(numCores, type = "FORK")
  clusterSetRNGStream(cl, sample(1e8,1))
}

if (!inherits(cl[[1]], "forknode")) {

  #parallel::clusterExport(cl, funs)
  #parallel::clusterExport(cl, objs )
  #env <- environment()
  parallel::clusterExport(cl,  "libs", envir = parent.frame())
  parallel::clusterEvalQ(cl, {lapply(libs, library, character.only = TRUE)})
}
```

```
#################### OPTIMIZATION
outs <- DEoptim(fitSN, lower = c(-2, -3, -3, -3, 1, 3.5, 0.75),
                upper = c(-0.1, -0.5, -0.5, -1, 3.5, 5, 1),
        control = DEoptim.control(VTR = 0.001, itermax = 170, cluster = cl, strategy = 6),
        ros = ros, centreCell = centreCell, fireSizes = c(10, 100, 1000, 10000, 100000),
        desiredPerimeterArea = 0.003, burnFun = burnFun, burn1 = burn1)

assign(paste0("outs", sample(1e6,1)), outs)
```

```
fitSN(sna = c(-1,-1,-1,-2,2,4,0.9),
      ros = ros, centreCell = centreCell, fireSizes = c(10, 100, 1000, 10000, 100000),
        desiredPerimeterArea = 0.003, burnFun = burnFun, burn1 = burn1)
```

```
fs <- round(runif(10, 10, 4000))
out <- fitSN2(par = c(2, -0.63333, 1, 3.2, 4.4),
              ros, centreCell, fireSizes = fs, 0.003)
bfs1 <- purrr::transpose(attr(out, "bfs1"))
LM <- do.call(rbind,bfs1$LM)
plot(fs, LM[,"perim.area.ratio"])
```

```
#################### OPTIMIZATION
fs <- c(0.2,1:8)*10000
outs <- DEoptim(fitSN2, lower = c(1, -1, 1, 3, 4), upper = c(3, -0.3, 3, 4, 5),
        control = DEoptim.control(VTR = 0.001, itermax = 40,
                                  cluster = cl, strategy = 6),
        ros = ros, centreCell = centreCell,
        fireSizes = fs, desiredPerimeterArea = 0.003)
```

```
############## MANUAL RUN

#10,000 hectares burns gave this
spawnNewActive[2:3] <- c(0.0235999945606232, 0.0263073265505955)

#100,000 hectare burns gave this
#spawnNewActive <- 10^c(-1.264780,   -1.970946,   -1.288213,   -2.202580)
spawnNewActive <- 10^c(-0.586503645288758, -1.08108837273903,
                       -2.14391896536108, -1.00221184641123)
sizeCutoffs <- 10^c(3.37711253212765, 4.52040993282571)
```

```
sns <- c(-1.733262, -0.933318, -2.562183, -2.493687, 3.064458, 4.812305)
spawnNewActive <- 10^sns[1:4]
sizeCutoffs <- 10^sns[5:6]
#spawnNewActive <- 10^c(-1.646419, -1.815395, -2.809013, -2.613337)
#sizeCutoffs <- 10^c(3.888317,    4.641961)

## 100,000 pixel fires -- the next worked, but I think we can get better
# sns <- structure(
#   c(-1.652459,-0.962121,-0.964879,-2.304902, 3.522345, 4.173242),
#   .Names = c("par1",
sns <- structure(
  c(-1.641197,-1.152821,-0.697335,-1.751917, 3.720378, 4.034059),
  .Names = c("par1", "par2", "par3", "par4", "par5", "par6"))
spawnNewActive <- 10^sns[1:4]
sizeCutoffs <- 10^(sns[5:6])
fireSize <- 30000

## 100
sns <- c(-0.77716149196811, -0.769325340166688, -1.2772046867758,
         -1.99332102853805, 3.14260408212431, 4.46155184064992)

## 1000
sns <- c(-0.775107,-1.031760,-0.599669,-1.958105, 3.048958, 4.275831)

## seemed good for 100,000, pretty good for 1e3
sns <- c(-1.54885, -0.97052, -1.38305, -1.93759, 3.20379, 4.13237)

## good for 100 000, 10 000 ha -- too sinuous for 1000 and 100 ha
sns <- c(-1.537203,-1.462981,-0.524957,-1.002567, 3.642046, 4.501754)

## good for 100 000, 10 000 ha (except some fires @ 1e5 don't make it to full size)
## -- too sinuous for smaller
sns <- c(-1.484338,-1.220440,-2.948275,-2.155940, 3.945281, 4.904893)

sns <- c(-1.495247,-0.800494,-1.582350,-2.270646, 3.530671, 4.663245)

## final optimization after 75 iterations, Good: 1e5, 1e4
sns <- c(-1.47809, -0.86224, -1.34532, -1.93568, 3.27149, 4.20741)

## based on equal weights 10^(1:5)
sns <- c(-0.923528, -1.804549, -1.760455, -1.793594,  1.683355,  4.466668)

## With sprearProb = 0.9 # Pretty GOOD!
sns <- c(-0.731520, -0.501823, -0.605968, -1.809726,  2.202732,  4.696060, 0.9)

## With sprearProb = 0.9 # Optimal
sns <- c(-0.978947, -0.540946, -0.790736, -1.583039,  2.532013,  4.267547,  0.946730)

spawnNewActive <- 10^sns[1:4]
sizeCutoffs <- 10^(sns[5:6])
if (length(sns) == 7) spreadProb <- sns[7]

# from linear model version
```

```
par <- c(1.548899,-0.396904, 2.191424, 3.903082, 4.854002)
sizeCutoffs <- 10^c(par[4], par[5])
sna <- min(-0.15, par[1] + par[2]*log10(fireSize))
sna <- 10^c(sna*par[3], sna*2*par[3], sna*3*par[3], sna*4*par[3])
spawnNewActive <- sna


##########################
clearPlot()
dev()

for (i in 1:5) {
  fireSize <- 10^i
  dim <- round(sqrt(fireSize)*5 * 250)
  ros <- raster(extent(0, dim,0,dim), res = 250, vals = 1)
  centreCell <- cellFromRowCol(ros,
                               rownr = nrow(ros) / 2,
                               colnr = ncol(ros) / 2)
  reps <- paste0("rep", 1:4 + (log10(fireSize) - 1)*4)
  burnedMapList <- wrap(cl = cl, reps = reps, par = TRUE, burn1 = burn1,
                        burnFun = burnFun, objs = objs, libs = libs)
  names(burnedMapList$out) <- reps
  burnedMapList <- purrr::transpose(burnedMapList$out)
  do.call(rbind, burnedMapList$LM)

  Plot(burnedMapList$burnedMap, cols = c("red"), new = FALSE, na.color = "white",
       legend = FALSE, visualSqueeze = 0.7, title = paste0("Fire size: ", fireSize))
}
```

```
reps <- paste0("rep", 1:1)
perims <- list()
perm <- list()
mod <- list()
dev()
clearPlot()
fireSizes <-  10^(4)

for (fs in fireSizes) {
  for (i in 1:1) {
    ros <- raster(extent(0, 2e5,0,2e5), res = 250, vals = 1)
    NineCorners <-
      cellFromRowCol(ros,
      rownr = nrow(ros) / 4 * rep(1:3, 3),
      colnr = ncol(ros) / 4 * rep(1:3, each = 3))
    centreCell <- NineCorners
    ran <- runif(4, -3, -1)
    spawnNewActive <- 10^ran
    #spawnNewActive <- 10^c(-0.1, -0.75, -1.2, ran*2.5)
    fireSize = rep(fs, length(centreCell))
    sizeCutoffs <- 10^c(1,3)
    burnedMapList <- wrap(cl = cl, reps = reps, par = TRUE, burn1 = burn1,
                          burnFun = burnFun, objs = objs, libs = libs)
    names(burnedMapList$out) <- reps
    burnedMapList <- purrr::transpose(burnedMapList$out)
```

```
    do.call(rbind, burnedMapList$LM)
    Plot(burnedMapList$burnedMap, new = TRUE, zero.color = "white")
    perims[[i]] <- data.frame(perim = burnedMapList$LM$rep1$perim.area.ratio,
                              spawnNewActive = mean(spawnNewActive),
                              others = t(spawnNewActive))
  }
  fsChar <- as.character(fs)
  perm[[fsChar]] <- rbindlist(perims)
  perm[[fsChar]]$perim <- log10(perm[[fsChar]]$perim)
  perm[[fsChar]]$spawnNewActive <- perm[[fsChar]]$spawnNewActive
  Plot(perm[[fsChar]]$perim, perm[[fsChar]]$spawnNewActive, new = TRUE,
       addTo = paste0("fs",fsChar))
  mod[[fsChar]] <- lm(spawnNewActive ~ perim, data = perm[[fsChar]])
}

(predict(mod[["10"]], data.frame(perim = log10(0.003))))
(predict(mod[["100"]], data.frame(perim = log10(0.003))))
log10(predict(mod[["1000"]], data.frame(perim = log10(0.003))))
```

References