LandMine

Eliot McIntire (eliot.mcintire@nrcan-rncan.gc.ca)
Alex Chubaty (achubaty@for-cast.ca)

18 May 2018; updated 6 Sep 2022

Overview

Landmine is a model created for simulating the natural range of variation for landscapes in the boreal forest (Andison 1996; Andison 1998). 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 fitted to the landscapes that are under study in the LandWeb project.

Known Species

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

Table 1: LandMine species codes.

Species	Group	Code
Jack pine	Pine (PINU)	Pinu_ban
Lodgepole pine	Pine (PINU)	Pinu_con
Unspecified pine species	Pine (PINU)	Pinu_sp
Paper birch	Deciduous (DECI)	Betu_pap
Balsam poplar	Deciduous (DECI)	Popu_bal
Trembling aspen	Deciduous (DECI)	Popu_tre
Larch/Tamarack	Deciduous (DECI)	Lari_lar
Black spruce	Black spruce (PICE_MAR)	Pice_mar
White spruce	White spruce (PICE_GLA)	Pice_gla
Fir species	Fir (ABIE)	Abie_sp

Table 2: LandMine module input objects and their descriptions.

objectName	desc
cohortData	Columns: B, pixelGroup, speciesCode (as a factor of the names), age. indicating several features about the current vegetation of stand.
fireReturnInterval	A raster layer that is a factor raster, with at least 1 column called 'fireReturnInterval', representing the fire return interval in years.
pixelGroupMap	Pixels with identical values share identical stand features
rasterToMatch	DESCRIPTION NEEDED
rasterToMatchReporting	Raster layer of study area used for plotting and reporting only. Defaults to the kNN biomass map masked with ‘studyArea‘
ROSTable	A data.table with 3 columns, 'age', 'leading', and 'ros'. The values under the 'age' column can be 'mature', 'immature', 'young' and compound versions of these, e.g., 'immature_young' which can be used when 2 or more age classes share same 'ros'. 'leading' should be vegetation type. 'ros' gives the rate of spread values for each age and type.
rstFlammable	A raster layer, with 0, 1 and NA, where 1 indicates areas that are flammable, 0 not flammable (e.g., lakes) and NA not applicable (e.g., masked)
rstTimeSinceFire	a time since fire raster layer
species	Columns: species, speciesCode, Indicating several features about species
sppColorVect	named character vector of hex colour codes corresponding to each species
sppEquiv	Multi-columned data.table indicating species name equivalencies. Default taken from LandR sppEquivalencies_CA which has names for species of trees in Canada
studyAreaReporting	multipolygon (typically smaller/unbuffered than studyArea) to use for plotting/reporting. Defaults to an area in Southwestern Alberta, Canada.

Module inputs and parameters

Input objects objects are described in Table 2.

Module parameters are described in Table 3.

Module outputs

Output objects are described in Table 4.

Table 3: LandMine module parameters and their descriptions.

paramName	paramDesc
biggestPossibleFireSizeHa	An upper limit, in hectares, of the truncated Pareto distribution of fire sizes
burnInitialTime	This describes the simulation time at which the first plot event should occur
fireTimestep	This describes the simulation time at which the first plot event should occur
flushCachedRandomFRI	If no Fire Return Interval map is supplied, then a random one will be created and cached. Use this to make a new one.
minPropBurn	Minimum proportion burned pixels to use when triggering warnings about simulated fires.
mixedType	How to define mixed stands: 1 for any species admixture; 2 for deciduous > conifer. See ?vegTypeMapGenerator.
maxRetriesPerID	Number of attempts that will be made per event ID, before abandoning. See ‘?SpaDES.tools::spread2‘.
maxReburns	Number of attempts to reburn fires that don’t reach their target fire size.
ROSother	default ROS value for non-forest vegetation classes.this is needed when passing a modified ROSTable, e.g. using log-transformed values.
sppEquivCol	The column in sim$specieEquivalency data.table to use as a naming convention
useSeed	Only used for creating a starting cohortData dataset. If NULL, then it will be randomly generated; If non-NULL, will pass this value to set.seed and be deterministic and identical each time. WARNING: setting the seed to a specific value will cause all simulations to be identical!
vegLeadingProportion	a number that define whether a species is leading for a given pixel
.plotInitialTime	This describes the simulation time at which the first plot event should occur
.plotInterval	This describes the simulation time interval between plot events
.saveInitialTime	This describes the simulation time at which the first save event should occur
.saveInterval	This describes the simulation time interval between save events
.useCache	Should this entire module be run with caching activated? This is generally intended for data-type modules, where stochasticity and time are not relevant
.unitTest	Some functions can have internal testing. This will turn those on or off, if any exist
.useParallel	Used in burning. Will be passed to data.table::setDTthreads(). NOTE: should be <= 2 as the additonal RAM overhead too high given marginal speedup.

Table 4: LandMine output objects and their description.

objectName	desc
fireInitialTime	The initial event time of the burn event. This is simply a reassignment from P(sim)$burnInitialTime.
fireSizes	A list of data.tables, one per burn event, each with two columns, size and maxSize. These indicate the actual sizes and expected sizes burned, respectively. These can be put into a single data.table with rbindlist(sim$fireSizes, idcol = 'year')
fireReturnInterval	A Raster map showing the fire return interval. This is created from the rstCurrentBurn.
fireReturnIntervalsByPolygonNumeric	A vector of the fire return intervals, ordered by the numeric representation of polygon ID
fireTimestep	The number of time units between successive fire events in a fire module.
kBest	A numeric scalar that is the optimal value of K in the Truncated Pareto distribution (rtruncpareto)
numFiresPerYear	The average number of fires per year, by fire return interval level on rstCurrentBurn.
rstCurrentBurn	A raster layer, produced at each timestep, where each pixel is either 1 or 0 indicating burned or not burned.
rstCurrentBurnCumulative	Cumulative number of times a pixel has burned
sppEquiv	Same as input, but with new column, LandMine

Usage

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

NB: Paths will have be changed for a different user.

Setup

Package installation:

```r
options(repos = c(CRAN = "https://cloud.r-project.org"))

if (!"pak" %in% rownames(installed.packages())) {
  install.packages("pak")
}

pak::pkg_install("PredictiveEcology/SpaDES@development")

## package SDM tools unavailable on CRAN since 2020-01-12; use latest archived version
if (!"SDMTools" %in% rownames(installed.packages())) {
  install.packages("https://cran.r-project.org/src/contrib/Archive/SDMTools/SDMTools_1.1-221.2.tar.gz",
                   repos = NULL)
}
```

Load necessary packages:

```r
library("Require")
Require(c("igraph", "magrittr", "raster", "SpaDES.core"))
```

Configure your file paths for this project:

```
workingDir <- file.path("~/GitHub/LandWeb") ## NOTE: change this for your project
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)[[1]]
```

```
##  [1] "SpaDES.core"
##  [2] "assertthat"
##  [3] "data.table"
##  [4] "ggplot2"
##  [5] "grDevices"
##  [6] "magrittr"
##  [7] "PredictiveEcology/LandR@development"
##  [8] "PredictiveEcology/pemisc@development"
##  [9] "PredictiveEcology/SpaDES.tools@development"
## [10] "raster"
## [11] "RColorBrewer"
## [12] "VGAM"
```

Module usage

```
studyArea <- SpaDES.tools::randomStudyArea(seed = 1234, size = 1e10)
rasterToMatch <- raster(studyArea, res = 250)
```

```
times <- list(start = 0, end = 13)

parameters <- list(
  # LandMine = list(flushCachedRandomFRI = TRUE)
)

modules <- list("LandMine")

objects <- list(
  studyArea = studyArea,
  rasterToMatch = rasterToMatch
)

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

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

ros <- raster(extent(0, 2.5e5, 0, 2.5e5), res = 250, vals = 0)
ros <- ros == 0
fireSize <- 100000
```

```r
Require(c("data.table", "DEoptim", "parallel", "SDMTools"))

source(file.path(moduleDir, "LandMine", "R", "burn.R"))

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, addlPkgs) {
  if (wantParallel) {
    library(parallel)
    if (is.null(cl)) {
      cl <- parallel::makeCluster(numClus)
    }
    funs <- c(funs)
    parallel::clusterExport(cl,  funs)
    #env <- environment()
    parallel::clusterExport(cl, "addlPkgs", envir = parent.frame())
    parallel::clusterEvalQ(cl, {lapply(addlPkgs, library, character.only = TRUE)})
    return(cl)
  }
}

wrap <- function(cl = NULL, reps, par, burn1, burnFun, objs, addlPkgs) {
  funNames <- c(deparse(substitute(burn1)), deparse(substitute(burnFun)))
  cl <- makeParallel(TRUE,
                     numClus = if (length(cl)) length(cl) else detectCores() - 1,
                     cl, c(funNames, objs), addlPkgs)
  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[], na.rm = TRUE) < fireSizes[bfCount])
    # it needs to get to above 90,000 HA for it to count
  })
  a <- sum(unlist(res))# * log10(fireSizes)) # weigh larger ones more
  attr(a, "bfs1") <- bfs1
  a
}

fitSN2 <- function(par, ros, centreCell, fireSizes = 10^(2:5),
                   desiredPerimeterArea = 0.003, spreadProb = 0.9) {
  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, burn1, spreadProb)
  })
  res <- lapply(seq(bfs1), function(bfCount) {
    abs(log(bfs1[[bfCount]]$LM[1, "perim.area.ratio"]) -
          log(desiredPerimeterArea)) +
      100*(sum(bfs1[[bfCount]]$burnedMap[], na.rm = TRUE) < 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
maxRetriesPerID <- 4 ## 4 retries (5 attempts total)
spreadProb <- 0.9
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,
                              row = nrow(ros) / 4 * rep(1:3, 3),
                              col = ncol(ros) / 4 * rep(1:3, each = 3))

centreCell <- cellFromRowCol(ros,
                             row = nrow(ros) / 2,
                             col = ncol(ros) / 2)

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

```
########################################
### SET UP CLUSTER FOR PARALLEL
numCores <- min(detectCores() / 2, 120L)
if (Sys.info()["sysname"] == "Windows") {
  cl <- parallel::makeCluster(numCores)
} else if (grepl("W-VIC-A1053|pinus[.]for-cast[.]ca", Sys.info()["nodename"])) {
  Require("future")

  ## TODO: customize this to make a cluster on several machines via ssh
  clNames <- c(
    #rep("localhost", 0),
    rep("picea.for-cast.ca", 20),
    rep("pseudotsuga.for-cast.ca", 50)
  )
  stopifnot(length(clNames) >= 70) ## 10 populations per parameter; 7 params.
  cl <- makeClusterPSOCK(clNames, homogeneous = FALSE, verbose = TRUE)

  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,  "addlPkgs", envir = parent.frame())
  parallel::clusterEvalQ(cl, {
    lapply(addlPkgs, library, character.only = TRUE)
  })
}
```

```
opt_sn <- 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)

opt_sn$optim$bestmem ## best param values

saveRDS(opt_sn, file.path(moduleDir, "LandMine", "data", paste0(Sys.Date(), "_DEoptim_250m.rds")))

## assign with suffix to facilitate multiple DEoptim runs
assign(paste0("opt_sn_", format(Sys.time(), "%Y%M%d%H%M%S")), opt_sn)
```

```
fs_sn <- c(10, 100, 1000, 10000, 100000)
fit_sn <- fitSN(sna = c(-1, -1, -1, -2, 2, 4, 0.9),
                ros = ros,
                centreCell = centreCell,
                fireSizes = fs_sn,
                desiredPerimeterArea = 0.003,
                burnFun = burnFun,
                burn1 = burn1)
bfs1_sn <- purrr::transpose(attr(fit_sn, "bfs1"))
LM_sn <- do.call(rbind, bfs1_sn$LM)
plot(fs_sn, LM_sn[, "perim.area.ratio"]) ## NOTE: visual inspection - not too round; not too sinuous
```

A second (alternative) version tries the optimization using fewer parameters, to test whether a simpler version gets better/different results. Although this version was not used for the final module, we preserve it here for posterity.

```
fs_optim2 <- c(0.2, 1:8)*10000
opt_sn2 <- 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_optim2, desiredPerimeterArea = 0.003)
```

```
fs_sn2 <- round(runif(10, 10, 4000))
fit_sn2 <- fitSN2(par = c(2, -0.63333, 1, 3.2, 4.4),
                  ros = ros,
                  centreCell = centreCell,
                  fireSizes = fs_sn2,
                  desiredPerimeterArea = 0.003,
                  spreadProb = 0.9)
bfs1_sn2 <- purrr::transpose(attr(fit_sn2, "bfs1"))
LM_sn2 <- do.call(rbind, bfs1_sn2$LM)
```

```
plot(fs_sn2, LM_sn2[, "perim.area.ratio"]) ## NOTE: visual inspection - not too round; not too sinuous
```

Manual inspection of optimization results

Original (2018) version

The original version was run using 100m pixels, despite the simulations being run using 250m pixels. This was corrected and rerun below.

```
## 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 spreadProb = 0.9 # Pretty GOOD!
sns <- c(-0.731520, -0.501823, -0.605968, -1.809726,  2.202732,  4.696060, 0.9) ## used in module

## With spreadProb = 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 = addlPkgs)
  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 = addlPkgs)
    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))))
```

Current (2022) version

The original version was run using 100m pixels, despite the simulations being run using 250m pixels. This version uses 250m pixels.

```
## final optimization after 170 iterations (fitSN)
sns <- unname(opt_sn$optim$bestmem)

spawnNewActive <- 10^sns[1:4]
sizeCutoffs <- 10^(sns[5:6])
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()
```

```r
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,
                               row = nrow(ros) / 2,
                               col = 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, addlPkgs = addlPkgs)
  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))
}
```

Cleaning up

```r
parallel::stopCluster(cl)
```

Code and data availability

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

References

Andison, D.W. (1996). Managing for landscape patterns in the sub-boreal forests of British Columbia. PhD Thesis. University of British Columbia, Vancouver, BC.

Andison, D.W. (1998). Temporal patterns of age-class distributions on foothills landscapes in Alberta. *Ecography*, 21, 543–550.