

Package: fireSenseUtils (via r-universe)

October 15, 2026

Type Package

Title Utilities for Working With the 'fireSense' Group of 'SpaDES' Modules

Description Utilities for working with the 'fireSense' group of 'SpaDES' modules.

Date 2026-09-28

Version 0.2.3.9098

Depends R (>= 4.3)

Imports climateData, clusters (>= 0.0.75), cowplot, crayon, data.table, EnvStats, fastDummies, fireregimetools, ggplot2, googledrive, LandR (>= 1.2.0.9022), magrittr, methods, parallel, parallelly, pemisc (>= 0.0.4.9011), purrr, quickPlot, reproducible (>= 3.2.1.9072), Require (>= 0.3.1), RhpcBLASctl, sf, sp, SpaDES.core (>= 3.2.1.9027), SpaDES.tools (>= 2.1.3.9013), stats, terra, tidyr, utils, withr

Suggests archive, callr, codetools, curl, deldir, digest, DEoptim, dismo, dplyr, hexbin, knitr, kSamples, pkgload, PtProcess, raster, rasterVis, RColorBrewer, remotes, rmarkdown, scfmutils, spatstat, spelling, testthat

Remotes FOR-CAST/fireregimetools@main,
PredictiveEcology/climateData@development,
PredictiveEcology/clusters@development,
PredictiveEcology/LandR@development,
PredictiveEcology/pemisc@development,
PredictiveEcology/reproducible@development,
PredictiveEcology/Require@development,
PredictiveEcology/scfmutils@development,
PredictiveEcology/SpaDES.core@development,
PredictiveEcology/SpaDES.tools@development

VignetteBuilder knitr, rmarkdown

Encoding UTF-8

Language en-CA

LazyData true

License GPL-3

BugReports <https://github.com/PredictiveEcology/fireSenseUtils/issues>

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.0

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libglpk-dev make default-jdk libicu-dev libjpeg-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libproj-dev libsdl3-dev libudunits2-dev

Repository <https://predictiveecology.r-universe.dev>

Date/Publication 2026-10-15 13:11:23 UTC

RemoteUrl <https://github.com/PredictiveEcology/fireSenseUtils>

RemoteRef development

RemoteSha 7cc9645bb8e31da7d531582d7810d254c6aebf39

Contents

fireSenseUtils-package	5
.objfunSpreadFit	6
abbreviateSpNames	12
adWeightAuto	12
annualStackToDTx1000	13
assertCovariateRange	14
assessFuelClasses	15
bufferEdge	16
bufferToArea	17
bufferToAreaRast	18
buildCohortBurnHistory	19
calcNonForestYoungAge	20
calcYoungAge	22
castCohortData	23
chk_duplicatedStartPixels	24
chooseDomSecFuelClasses	25
cleanUpSpreadFirePoints	26
climateCovRanges	26
climateRasterToDataTable	27
cohortsToFuelClasses	27
combine_fuel_classes	29
combinePlans	29
covsX1000AndSetDF	30
dataYearFor	31
defaultFireYears	32
deoptimPopulationSize	32
dtReplaceNAwith0	33
ELFireCounts	33
ELFireOverlay	34

ELFfitStatus	35
ELFflammableArea	36
elfLabels	36
ELFmergedName	37
ELFmergePlan	38
ELFneighbours	38
ELFregionStats	39
ELFrunName	40
ELFsArctic	41
ELFsExcluded	41
ELFsInStudyArea	42
ELFsizePlan	43
ELFsSkipped	44
ELFtemplateRaster	44
emptySpreadCovariates	45
escapedFires	46
escapeSizePixels	46
fewFireELFs	47
firePixelsByYear	48
fireRecordShapefile	49
fireSenseCloudParameters	49
fireSenseCloudParametersMap	50
fireSenseCovariatesCreate	51
fireSenseNonflammableLCC	55
fireSenseSharedDefaults	56
fireSenseUtils-constants	57
fitDiagnostics	58
fitLedgerFolders	61
fixedLogisticPars	62
fuelClassPrep	62
fuelLinearRange	63
fuelLogToLinear	63
getFirePoints_NFDB	64
getFirePoints_NFDB_V2	65
getFirePolygons	66
harmonizeBufferAndPoints	67
harmonizeFireData	68
harmonizeFireDataDeps	69
hindcastObserved	69
hindcastSimulated	70
ignitionFitFilenameFor	71
igOrEscNames	72
isLinearFuelRange	73
isYoungAge	73
latestNBACUrl	74
lccFlammableShare	75
logistic4p	75
logisticParamNames	77

logMinB	77
makeELFs	78
makeFireIDs	80
makeFireSenseLCC	81
makeFireSenseLCCDeps	83
makeGLM	84
makeLandcoverDT	85
makeLociList	85
makeMutuallyExclusive	86
makeRastersFromCD	87
makeTSD	87
mergeELFs	88
mergePreparedCovs	89
multiplier	92
mustFit	93
nfdbPointUrl	93
omitNullArgs	94
paramsSeparate	94
plotBurnSummary	95
plotCumulativeBurns	96
plotHindcastDistributions	97
plotHindcastMap	97
plotHindcastSizes	98
plotHistoricFires	99
predictIgnition	100
prepare_FuelCovsCoarse	100
prepare_FuelCovsCoarseByYear	102
prepare_ignitionClimate	103
prepare_LightningData	105
prepareCovariatesOuter	107
putBackIntoRaster	109
rasterFireBufferDT	111
rasterFireSpreadPoints	112
rbetaBetween	112
readLightningData	113
removeBufferedFiresOutsideRTM	114
rescaleCovariates	115
rescaleKnown2	116
rescaleVarsByMagnitude	117
rescorePopulation	117
resolveDoFit	118
runDEoptim	119
runELFs	124
sizeRuleLandCover	126
spreadCovCentre	126
spreadDesignCols	127
spreadFitAdditionalColNames	128
spreadFitFilenameFor	128

spreadFitValidationData	129
spreadFitWorkerMemoryGB	132
spreadIndicatorCols	133
spreadProbBounds	133
spreadProbFromIntegerCovs	134
spreadProbGates	136
spreadProbGateTest	137
stackAndExtract	138
toX1000	138
updateStackYearNames	139
visualizeDE	139
youngAgeAtYear	140
youngAgeExclusiveCols	141
Index	142

fireSenseUtils-package

fireSenseUtils *package*

Description

Utilities for working with the 'fireSense' group of 'SpaDES' modules.

Author(s)

Maintainer: Eliot J B McIntire <eliot.mcintire@nrca-nrcan.gc.ca> ([ORCID](#))

Authors:

- Eliot J B McIntire <eliot.mcintire@nrca-nrcan.gc.ca> ([ORCID](#))
- Jean Marchal <jean.d.marchal@gmail.com>
- Alex M Chubaty <achubaty@for-cast.ca> ([ORCID](#))
- Ian Eddy <ian.eddy@nrca-nrcan.gc.ca> ([ORCID](#))

See Also

Useful links:

- Report bugs at <https://github.com/PredictiveEcology/fireSenseUtils/issues>

.objfunSpreadFit	<i>Objective function for fireSense_spreadFit module</i>
------------------	--

Description

Objective function for fireSense_spreadFit module

Usage

```
.objfunSpreadFit(
  par,
  landscape,
  annualDTx1000,
  nonAnnualDTx1000,
  formulaToFit,
  historicalFires,
  fireBufferedListDT,
  covMinMax = NULL,
  maxFireSpread = spreadProbCeiling,
  lowerSpreadProb = spreadProbFloor,
  minFireSize = 2,
  escapeSizeHa = NULL,
  jumpTries = 0,
  jumpMeanDist = 0,
  yearAreaWeight = 0,
  areaDistWeight = 0,
  returnTerms = FALSE,
  tests = "snll_fs",
  Nreps = 10,
  mutuallyExclusive = list(youngAge = c("class", "nf")),
  doAssertions = TRUE,
  plot.it = FALSE,
  objFunCoresInternal = 1,
  lanscape1stQuantileThresh = 0.265,
  thresh = 550,
  pruneAbove = Inf,
  covCentre = NULL,
  weighted = TRUE,
  sizeLik = "kde",
  sizeLikDf = 5,
  adWeight = "auto",
  link = NULL,
  returnSims = FALSE,
  returnBurned = FALSE,
  penaliseRunaways = TRUE,
  runawayEdgeFrac = fireSenseRunawayEdgeFrac,
  runawayEdgeMin = fireSenseRunawayEdgeMin,
```

```

runawaySize = NULL,
runawayBufferMultiple = NULL,
fitYearSpreadSD = NULL,
verbose = 2,
...
)

```

Arguments

<code>par</code>	parameters: the logistic parameters (<code>maxAsymptote</code> first, then <code>upperTail1</code> if the link has one; without <code>hillSlope1</code> and <code>inflectionPoint1</code> , which are fixed at 1 – see <code>fixLogisticPars()</code>), then the covariate coefficients (the intercept's first, when the formula has one – see <code>spreadDesignCols()</code>), and optionally <code>yearSpreadSD</code> last (see <code>fitYearSpreadSD</code>).
<code>landscape</code>	A <code>SpatRaster</code> with extent, resolution, and projection (<code>crs</code>) used for <code>SpaDES.tools::spread2</code> .
<code>annualDTx1000</code>	A list of <code>data.table</code> class objects. Each list element is data from a single calendar year, and whose name takes the form "yearxxxx", where xxxx is the four-digit year. The columns in the <code>data.table</code> must integers, that are 1000x their actual values as this function will divide by 1000.
<code>nonAnnualDTx1000</code>	Like <code>annualDTx1000</code> , but with where each list element will be used for >1 year. The names of the list elements must be of the form "yearxxxx_yearyyyy_yearzzzz" where xxxx, yyyy, and zzzz represent the four-digit calendar years for which that list element should be used. The columns are variables that are used for more than 1 year.
<code>formulaToFit</code>	Formula, put provided as a character string, not class formula. (if it is provided as a class formula, then it invariably will have an enormous amount of data hidden in the formula environment; this is bad for <code>DEoptim::DEoptim</code>)
<code>historicalFires</code>	A named list of <code>data.frames</code> (one per year, names matching those of <code>annualDTx1000</code>), each with columns <code>cells</code> (pixel indices of ignitions) and <code>size</code> (fire size in pixels). The observed fire record against which the simulation is compared.
<code>fireBufferedListDT</code>	A named list of <code>data.tables</code> (one per year, names matching <code>annualDTx1000</code>), each with columns <code>pixelID</code> , <code>buffer</code> (1 if the pixel is inside a buffered fire footprint, 0 otherwise), and <code>ids</code> (a unique identifier per fire). Restricts the set of pixels used when fitting and scoring spread.
<code>covMinMax</code>	This is a 2 row by multiple column <code>data.frame</code> indicating the minimum and maximum values of the original covariate data values. These will be used to rescale the covariates internally so that they are all between 0 and 1. It is important to not simply rescale internally here because only 1 year is run at a time; all years must be rescaled for a given covariate by the same amount.
<code>maxFireSpread</code>	A value for <code>spreadProb</code> that is considered impossible to go above. Default <code>spreadProbCeiling</code> , also the upper bound of <code>maxAsymptote</code> . <code>maxAsymptote</code> is the spread-probability ceiling in a typical year. The year random effect (<code>yearSpreadSD</code>) is added on the logit of the final spread probability, after the link, so in a given

year *p* can exceed *maxAsymptote* or fall below *lowerSpreadProb*; that is intended. There is no absolute cap, because *spreadCpp* does not need one; *maxAsymptote* is bounded because runaway fires are slow to simulate and wasted if the parameters are wrong.

<i>lowerSpreadProb</i>	Numeric. Lower bound for <i>spreadProb</i> ; if a candidate <i>spreadProb</i> falls at or below this, the call exits early to avoid wasted simulation work. Default spreadProbFloor .
<i>minFireSize</i>	Integer. Minimum fire size (in pixels) to include when scoring simulated against historical fires; fires smaller than this are filtered out of the comparison. Default 1.
<i>escapeSizeHa</i>	Numeric or NULL. Size (ha) a fire must reach to count as escaped. NULL (the default) keeps the historical behaviour. When set, only observed fires of at least escapeSizePixels() pixels are fitted, and every simulated fire starts as an escaped fire: it burns its first escapeSizePixels() cells regardless of <i>spreadProb</i> (<i>SpaDES.tools::spreadCpp(minSize =)</i>), then spreads with the fitted probabilities. So the spread model is fitted as "given the fire escaped".
<i>jumpTries</i> , <i>jumpMeanDist</i>	Used with <i>escapeSizeHa</i> : passed to <i>SpaDES.tools::spreadCpp()</i> , so a fire stuck below the escape size (no burnable neighbour left) can jump to nearby burnable land. <i>jumpTries</i> is how many times it may try, <i>jumpMeanDist</i> the mean jump distance in pixels. 0 (default) is off.
<i>yearAreaWeight</i>	Weight of the annual-area term: each fit year's observed area burned scored against that year's simulated totals, one per replicate, with the per-fire size likelihood (<i>yearAreaNLL()</i>). 0 (default) leaves it out; "auto" is (number of fitted fires) / (number of fit years), so the year view and the per-fire view carry equal total weight.
<i>areaDistWeight</i>	Weight of the area-weighted size-distribution term (<i>areaWeightedCvM()</i>), which compares simulated and observed fires by the share of area burned they make up. 0 (default) leaves it out; "auto" uses the AD term's weight, <i>adWeightAuto()</i> .
<i>returnTerms</i>	Logical. TRUE returns the objective with each term separately (diagnostics), and two fields about the first block of years (the one thresh stops early): <i>firstBlockSNLL</i> , its SNLL per year as thresh is compared with it (so a trial with <i>thresh = Inf</i> reports what it would have been judged on), and <i>bailed</i> , 1 if a year of that block was refused as "too burny" or "not spread out enough" or the block's SNLL was above the threshold, else 0.
<i>tests</i>	One or more of "mad", "adTest", "SNLL", or "snll_fs". Default: "snll_fs".
<i>Nreps</i>	Integer. The number of replicates, per ignition, to run.
<i>mutuallyExclusive</i>	Named list of vectors describing groups of model terms that must not be active together (mutually exclusive in the formula). Default <i>list("youngAge" = c("class", "nf"))</i> .
<i>doAssertions</i>	Logical. If TRUE, the default, the function will test a few minor things for consistency. This should be set to FALSE for operational situations, as the assertions take some small amount of time.

<code>plot.it</code>	Passed to <code>SpaDES.core::Plots()</code> , so will show (TRUE or "screen") or save files of several plots, using <code>ggplot2</code> .
<code>objFunCoresInternal</code>	Internally, this function can use <code>mcmapply</code> to run multiple parallel spread function calls. This should only be $> 1L$ if there are spare threads. It is highly likely that there won't be. However, sometimes the <code>DEoptim::DEoptim</code> is particularly inefficient, it starts X cores, and immediately several of them are stopped inside this function because the parameters are so bad, only two years are attempted. Then the core will stay idle until all other cores for the <code>DEoptim::DEoptim</code> iteration are complete. Similarly, if only physical cores are used for <code>DEoptim::DEoptim</code> , the additional use of hyperthreaded cores here, internally will speed things up (i.e., this maybe could be $2L$ or $3L$).
<code>landscape1stQuantileThresh</code>	A <code>spreadProb</code> value that represents a threshold for the 1st quantile of the <code>spreadProbs</code> on the landscape; if that quantile is above this number, then the <code>.objFunSpreadFit</code> will bail because it is "too burny" a landscape. Default 0.265, meaning if only 25%% of the pixels on the landscape are below this <code>spreadProb</code> , then it will bail.
<code>thresh</code>	Threshold multiplier used in SNLL fire size (" <code>snll_fs</code> ") test. Default 550.
<code>pruneAbove</code>	Numeric. An upper bound on the accumulated SNLL after the <i>first</i> batch of fire years, past which the evaluation stops early and returns the fail value. Default <code>Inf</code> , which leaves <code>thresh * <years done></code> as the only bound – i.e. no change in behaviour. A caller that runs one <code>DEoptim</code> generation per call knows the current population, and can pass the worst objective value that population would accept. Doing so is <i>exact</i> , not a heuristic: the second batch contributes a non-negative SNLL, so the first batch's accumulated value is a lower bound on the evaluation's final value. A trial exceeding <code>pruneAbove</code> would therefore have finished at or above a value every parent already beats, and selection would have discarded it. Pruning cannot change which trials <code>DEoptim</code> keeps; it only avoids finishing an evaluation whose outcome is already decided. This matters because a generation is synchronous – its wall time is the slowest of its NP evaluations, not the median one. Lowering the threshold value will be more restrictive, but being too restrictive will result in <code>DEoptim::DEoptim</code> rejecting more tests and using the "fail value" of 10000. Too high a threshold, and more years will be run and it will take longer to find values.
<code>covCentre</code>	Optional named list or vector of values subtracted from the rescaled covariates of the same name. NULL (default) does no centring. Must be the same at fit and predict. <code>spreadCovCentre()</code> gives the centres to use with an intercept.
<code>weighted</code>	Should each fire's log-likelihood be multiplied by a weight that grows with its observed size, to give large fires more influence on the SNLL? FALSE: no. TRUE or "log": <code>log(size)</code> . "sqrt": <code>sqrt(size)</code> . Weights are divided by their mean over the fitted fires, so the SNLL keeps its scale against the other tests.
<code>sizeLik</code>	How a fire's likelihood is taken from its <code>Nreps</code> simulated sizes, on the square-root scale. "kde" (default): their kernel density at the observed size (<code>EnvStats::demp()</code>); it is zero away from the simulated sizes, so a fire they never reach scores the

	minLik floor however far off it is. "t": a Student-t centred on their mean with their standard deviation as scale; no floor, and a miss costs more the further it is.
sizeLikDf	Degrees of freedom of the Student-t when sizeLik = "t". Default 5. Lower values forgive a missed fire more.
adWeight	What the "adTest" statistic is multiplied by before it is added to the fire-size SNLL. "auto" (default) uses adWeightAuto() , which gives the two terms about equal influence for any sizeLik and weighted. A number is used as given; the former fixed default, 50, balanced the two only for sizeLik = "kde" with weighted = FALSE.
link	Passed to logisticAll() : NULL for the form implied by the logistic parameters, or "logistic3pUpper" for the 3-parameter logistic with Stukel's upper tail, whose fourth logistic parameter is upperTail1. Pass it explicitly when fitting: par may arrive unnamed.
returnSims	If TRUE, return the simulated fires instead of the objective value: a <code>data.table</code> with one row per fire year x fire x replicate – yr, rep, initialLocus, ids, the simulated size sim and the observed size, in pixels. A year the objective declines to simulate (its spread probabilities fail the sanity checks) keeps its fires with rep and sim NA, so the caller can count them. Every year is simulated: the early stop (thresh) is not applied. For diagnostics and validation; the value is not an objective. The result carries <code>attr(, "spreadProb")</code> : the spread probabilities of every fitted pixel-year, summarised by spreadProbSummary() , for linkSaturation() .
returnBurned	If TRUE, also record which pixels burned: implies returnSims = TRUE, and the result carries <code>attr(, "burned")</code> , a named list with one element per simulated fire year, each a <code>data.table</code> with the replicate rep and the pixelID of every pixel that burned in it (ignition cells included). A year the objective declines to simulate is NULL. For spreadFitValidationData() . FALSE (default) records nothing and changes nothing.
penaliseRunaways	If TRUE (default), a simulated fire that burns at least k pixels of the edge ring of its own fire's buffer (see runawayEdgeFrac, runawayEdgeMin) is censored: it reached the outer edge of the area it is allowed to spread in, so it is "at least this big", a runaway. The edge ring is the buffer pixels with a queen neighbour outside the fire's own buffer (see bufferEdge()). In the per-fire size likelihood a runaway has no density at the observed size (the likelihood is that of the other replicates times their share; 0, i.e. minLik, if fewer than two replicates remain). The other size-based terms ("adTest", yearAreaWeight, areaDistWeight, "mad") score the size it burned, unless runawayBufferMultiple or runawaySize is given. FALSE scores the simulated size everywhere. Fires are not capped at any size: spread is bounded only by the year's buffers. Has no effect with returnSims = TRUE (the simulated sizes are returned as simulated). The per-fire likelihood is floored at minLik (1e-29), which bounds the penalty per fire. The ring is computed once per fit (runDEoptim() adds an edge column to fireBufferedListDT); a table without that column gets it computed on every call. With the penalty, a fire stops burning once it is a runaway (it has burned k of its ring pixels; <code>SpaDES.tools::spreadCpp(stopCells =)</code>), as nothing it burns after that is scored. Not with returnSims = TRUE.

<code>runawayEdgeFrac, runawayEdgeMin</code>	A replicate is a runaway only if it burns at least k distinct pixels of its fire's edge ring, where $k = \max(\text{runawayEdgeMin}, \text{ceiling}(\text{runawayEdgeFrac} * n))$ and n is the number of ring pixels of that fire; k is never more than n . One touched pixel is luck, not a fire that wants to leave its buffer. Defaults fireSenseRunawayEdgeFrac and fireSenseRunawayEdgeMin . Only the classification changes: what is done to a runaway is as described for <code>penaliseRunaways</code> .
<code>runawaySize</code>	NULL (default), or a size (pixels) given to a runaway replicate in the "adTest", <code>yearAreaWeight</code> , <code>areaDistWeight</code> and "mad" terms when <code>penaliseRunaways = TRUE</code> . It used to default to the landscape's non-NA pixel count; that made one runaway replicate, which typically had burned 1-3 times its observed fire's size, count as burning the whole landscape, and the fit chose parameters that rarely reach a buffer edge, so it under-burned (held-out ELF, 2026-10-01: simulated/observed area 1.33 -> 0.54).
<code>runawayBufferMultiple</code>	NULL (default), or a number: with <code>penaliseRunaways = TRUE</code> , a runaway replicate counts in the "adTest", <code>yearAreaWeight</code> , <code>areaDistWeight</code> and "mad" terms as this multiple of the size (pixels) of its own fire's buffer, i.e. all rows of that fire's ids in the year's <code>fireBufferedListDT</code> table, burned and unburned. The size it burned up to the early stop was too cheap: fits that matched the observed area inside the buffers ran away on the whole landscape (ELF 5.4 hind-cast, 2026-10-06: fires of 3-6 Mha, median 2-12 times the observed area). Give either this or <code>runawaySize</code> , not both.
<code>fitYearSpreadSD</code>	If TRUE, the last element of <code>par</code> is the sd of a per-year random effect on the logit of spread probability, a seasonal departure: in each replicate the year draws one $\text{eps} \sim N(0, \text{sd}^2)$ and every pixel of that year spreads with <code>plogis(qlogis(p) + eps)</code> , so all of the year's fires are bigger or smaller together. Without it the simulated fire sizes were too alike: medians too large and the largest fires too small at once (2026-09-23). <code>sd = 0</code> is the model without it, exactly (no random numbers are drawn). NULL (default): TRUE when <code>par</code> is named and includes <code>yearSpreadSD</code> . <code>DEoptim</code> passes <code>par</code> unnamed, so <code>runDEoptim()</code> sets it explicitly.
<code>verbose</code>	If ≥ 2 , then this will show more information about <code>spreadProb</code> fitting.
<code>...</code>	This is not used here, but allows for extraneous arguments to not break this function. The removed arguments <code>capSizes</code> and <code>penaliseCapHits</code> arrive here: they are ignored with a deprecation warning (fires are no longer capped at a size; use <code>penaliseRunaways</code>).

Value

Attempting a weighted likelihood, <https://stats.stackexchange.com/questions/267464/algorithms-for-weighted>
 With `log(fireSize) * likelihood` for each fire.

abbreviateSpNames	<i>Abbreviate species names for fuel classes</i>
-------------------	--

Description

Abbreviate species names for fuel classes

Usage

```
abbreviateSpNames(df)
```

Arguments

df	data.table
----	------------

Value

data.table

adWeightAuto	<i>Automatic weight for the Anderson-Darling term</i>
--------------	---

Description

The multiplier that gives the "adTest" term and the fire-size SNLL about equal influence on the objective, $c * \sqrt{nFires}$. Both terms are sums over the fitted fires, so both grow with how many there are; measured across six ELF's (138 to 1624 fires, 2,708 to 1,114,569 pixels burned, 14 members of each fit's final population), the spread of the SNLL across members grew with $nFires^{0.5}$ to $nFires^1$, and neither term scaled with the ELF's area or its area burned. c comes from the same measurements, one per sizeLik and weighted combination. Under it the AD term's share of the influence stayed within 0.36-0.68 of the total, against 0.16-0.94 under the fixed 50.

Usage

```
adWeightAuto(nFires, sizeLik = "kde", weighted = FALSE)
```

Arguments

nFires	Number of fires being fitted (after minFireSize).
sizeLik	As in .objfunSpreadFit() .
weighted	As in .objfunSpreadFit() ; TRUE uses the unweighted constants.

Details

The constants were measured on populations fitted with an older objective, and a spread estimated from 14 members is coarse, so treat the result as the right order of magnitude rather than exact.

Value

A single numeric.

annualStackToDTx1000 *Convert list of annual SpatRaster to data.table*

Description

Convert list of annual SpatRaster to data.table

Usage

```
annualStackToDTx1000(x, whNotNA, ...)

## S3 method for class 'SpatRaster'
annualStackToDTx1000(x, whNotNA, ...)

## S3 method for class 'Raster'
annualStackToDTx1000(x, whNotNA, ...)

## S3 method for class 'list'
annualStackToDTx1000(x, whNotNA, ...)
```

Arguments

x	RasterStack or list of rasters to convert to data.table and multiply by 1000 to save space.
whNotNA	Pixel indexes that should go through this process (i.e. not NA)
...	Not currently used

Value

data.table of the SpatRaster or the list

Examples

```
withr::local_package("raster")

r1 <- raster(extent(0, 10, 0, 10), vals = 1:100)
r2 <- raster(extent(0, 10, 0, 10), vals = 100:1)
r3 <- raster(extent(0, 10, 0, 10), vals = 200:101)
r4 <- raster(extent(0, 10, 0, 10), vals = 300:201)

# list of Rasters
lRast <- list(r1, r2, r3)
lRast[[1]][5] <- NA
whNotNA <- setdiff(1:ncell(r1), 5)
```

```

# unnamed -- should error
try(out1 <- annualStackToDTx1000(lRast, whNotNA))

# named
names(lRast) <- c("OneToHun", "HunToOne", "TwoHunToOneHun")
out1 <- annualStackToDTx1000(lRast, whNotNA)

# RasterStack
out2 <- annualStackToDTx1000(raster::stack(lRast), whNotNA)

# List of RasterStacks
s1 <- raster::stack(r1, r2)
names(s1) <- names(lRast)[1:2]
s2 <- raster::stack(r4, r3)
names(s2) <- c(names(lRast)[3], "ThreeHunToTwoHun")
out3 <- annualStackToDTx1000(list(s1 = s1, s2 = s2), whNotNA) ## named list required

# With duplicated names -- to remove duplicates;
# actually, this doesn't make sense: RasterStack can't have duplicated names
names(lRast) <- c("OneToHun", "OneToHun", "TwoHunToOneHun")
out4 <- annualStackToDTx1000(raster::stack(lRast), whNotNA)

## cleanup
withr::deferred_run()

```

assertCovariateRange *Assert that rescaled spread covariates are usable*

Description

Covariates were historically required to lie in $[0, 1]$. That upper bound was a convention of the log-biomass parameterisation, not a requirement of the objective function: covariates enter only through $\exp(\text{mat} \%*\% \text{covPars})$ inside `logistic3p()`, which saturates gracefully ($\exp(\text{Inf})^{(-b)}$ is 0, giving `maxAsymptote`) and never produces NaN. Fuel covariates expressed as biomass / $1\text{e}4$ legitimately exceed 1, so only non-negativity and finiteness are enforced.

Usage

```
assertCovariateRange(dt, colsToUse)
```

Arguments

<code>dt</code>	A <code>data.table</code> of rescaled covariates.
<code>colsToUse</code>	Character vector of covariate column names to check.

Value

`invisible(TRUE)`, or an error.

assessFuelClasses	<i>Semi-automated selection of fuel classed based on GLMs</i>
-------------------	---

Description

Semi-automated selection of fuel classed based on GLMs

Usage

```
assessFuelClasses(
  landscape,
  fuelCol,
  sppEquiv,
  sppEquivCol,
  targetNonForestClasses = 2,
  targetFuelClasses = 5,
  nonforestLCC,
  pValue = 0.001,
  lccShare = NULL,
  minCovariateProp = 0.05
)
```

Arguments

landscape	data.table created by prepFuelClasses
fuelCol	the column in sppEquiv with default fuel classes
sppEquiv	table with species name equivalencies between the kNN and final naming formats. See data("sppEquivalencies_CA", "LandR"). Only necessary if mixedType == 2. If not provided and mixedType == 2, will attempt to use data("sppEquivalencies_CA", "LandR").
sppEquivCol	the column name to use from sppEquiv.
targetNonForestClasses	the number of non-forest fuel classes to generate
targetFuelClasses	target number of treed fuel classes to generate
nonforestLCC	vector or list of vectors of non-forest fuel classes
pValue	for glm coefficient significance when deciding to merge fuel classes
lccShare	Optional named numeric: each land-cover class's share of the ELF's flammable pixels, named by class code (see lccFlammableShare()). NULL (default): every non-forest class is clustered.
minCovariateProp	With lccShare, a non-forest class whose share is below this is too rare to place by its own coefficient: it is left out of the clustering and joins the group of the class whose burn coefficient is nearest, as missingForest does. When every non-forest class is that rare, they form a single group.

Value

a list of three objects:

1. `modSppEquiv`, a `data.table` with assigned fuel classes for each tree species;
2. `nonForestedLCCGroups`, a named list of non-forest land cover classes grouped by fuel class;
3. `missingLCCgroup`, the class in `nonForestLCCGroups` to assign forested pixels missing from `cohortData`;

<code>bufferEdge</code>	<i>The edge ring of each fire's buffer</i>
-------------------------	--

Description

A simulated fire that reaches the outer edge of its own buffer has run away: the spread objective (`.objfunSpreadFit()`) censors it. `bufferEdge()` marks the pixels that make that edge: a pixel of a fire's buffer (burned or not) with at least one of its eight (queen) neighbours outside that fire's own buffer. "Outside" is anything that is not a row of the same fire in the table: unburned land of another fire's buffer, pixels removed as non-flammable (see `removeBufferedFiresOutsideRTM()`), NA cells, and the raster's boundary.

Usage

```
bufferEdge(fireBufferedDT, r)

addBufferEdge(fireBufferedListDT, landscape)
```

Arguments

<code>fireBufferedDT</code>	A table of one fire year with columns <code>ids</code> (the fire) and <code>pixelID</code> (cell number in <code>r</code>), as in <code>fireBufferedListDT</code> .
<code>r</code>	A raster with the grid <code>pixelID</code> refers to; only its size is used.
<code>fireBufferedListDT</code>	A list of tables as <code>fireBufferedDT</code> , one per fire year.
<code>landscape</code>	As <code>r</code> .

Value

`bufferEdge()`: a logical vector, one element per row of `fireBufferedDT`. `addBufferEdge()`: `fireBufferedListDT` with that vector as a column `edge` in every table that does not have one already (the tables are copied, not modified).

bufferToArea*Create buffers around polygons based on area target for buffer*

Description

Create buffers around polygons based on area target for buffer

Usage

```
bufferToArea(  
  poly,  
  rasterToMatch,  
  areaMultiplier,  
  verb = FALSE,  
  polyName = NULL,  
  field = NULL,  
  minSize = 500,  
  cores = 1,  
  ...  
)  
  
## S3 method for class 'list'  
bufferToArea(  
  poly,  
  rasterToMatch,  
  areaMultiplier = 10,  
  verb = FALSE,  
  polyName = NULL,  
  field = NULL,  
  minSize = 500,  
  cores = 1,  
  ...  
)  
  
## S3 method for class 'SpatialPolygons'  
bufferToArea(  
  poly,  
  rasterToMatch,  
  areaMultiplier = 10,  
  verb = FALSE,  
  polyName = NULL,  
  field = NULL,  
  minSize = 500,  
  cores = 1,  
  ...  
)
```

```
## S3 method for class 'sf'
bufferToArea(
  poly,
  rasterToMatch,
  areaMultiplier = 10,
  verb = FALSE,
  polyName = NULL,
  field = NULL,
  minSize = 500,
  cores = 1,
  ...
)
```

Arguments

poly	sf polygons or a list of sf containing polygons to buffer.
rasterToMatch	A SpatRaster with res, origin, extent, crs of desired outputted pixelID values.
areaMultiplier	Either a scalar that will buffer $\text{areaMultiplier} * \text{fireSize}$ or a function of fireSize. Default is 1. See multiplier() for an example.
verb	Logical or numeric related to how much verbosity is printed. FALSE or 0 is none. TRUE or 1 is some. 2 is much more.
polyName	Optional character string of the polygon layer name (not the individual polygons on a sf polygon object)
field	Passed to <code>fasterize::fasterize</code> . If this is unique (such as polygon id), then each polygon will have its buffer calculated independently for each unique value in field
minSize	The absolute minimum size of the buffer & non-buffer together. This will be imposed after areaMultiplier.
cores	number of processor cores to use
...	passed to <code>fasterize::fasterize</code>

Value

A data.table (or list of data.tables if poly was a list) with 2 columns: buffer and pixelID. buffer is either 1 (the original polygon) or 0 (in the buffer).

bufferToAreaRast	<i>Create a variable sized buffer around a set of pixels belonging to the same fire ID</i>
------------------	--

Description

Create a variable sized buffer around a set of pixels belonging to the same fire ID

Usage

```
bufferToAreaRast(fireIDraster, areaMultiplier, minSize, flammableRTM, verb = 1)
```

Arguments

fireIDraster	a SpatRaster with values representing distinct fires in a year
areaMultiplier	A scalar that will buffer $\text{areaMultiplier} * \text{fireSize}$
minSize	The absolute minimum size of the buffer & non-buffer together. This will be imposed after areaMultiplier.
flammableRTM	a rasterToMatch with binary values where 1 represents flammable pixels, 0 nonflammable, and NA no data
verb	Logical or numeric related to how much verbosity is printed. FALSE or 0 is none. TRUE or 1 is some. 2 is much more.

Value

a data.table with fire ID, buffer status, and pixelID

```
buildCohortBurnHistory
```

Modify cohortData with burn column

Description

Modify cohortData with burn column

Usage

```
buildCohortBurnHistory(cohortData, pixelGroupMap, firePolys, year)
```

Arguments

cohortData	either a cohortData object or list of cohortData objects named by year.
pixelGroupMap	either a SpatRaster with pixelGroups or list of SpatRasters named by year.
firePolys	the output of <code>getFirePolygons()</code> with YEAR column.
year	length-two vector giving temporal period used to subset firePolys. Closed interval.

Value

cohortData modified with burn status

calcNonForestYoungAge *Zero-out young non-forest pixels across LCC layers and add a youngAge mask*

Description

Identifies pixels that have **any non-forest presence** (based on the non-pixelID columns of landcoverDT) and whose **age** is **below** cutoffForYoungAge. It then sets those pixels to 0 across **all layers** of LCCras and appends a new single-layer raster named youngAge with 1 for the affected pixels and 0 elsewhere (preserving NA where present in the template).

Usage

```
calcNonForestYoungAge(landcoverDT, NFTSD, LCCras, cutoffForYoungAge)
```

Arguments

landcoverDT	data.table. A data.table containing a pixelID column of integer cell indices (1-based) and one or more non-forest columns (all columns except pixelID are interpreted as non-forest for the row-sum).
NFTSD	numeric. A numeric vector (or similar) that can be indexed by pixelID to obtain per-cell age values.
LCCras	SpatRaster. A terra::SpatRaster (single- or multi-layer) whose cells correspond to pixelID. Pixels identified as "young non-forest" are set to 0 in all layers. A new youngAge layer is appended.
cutoffForYoungAge	numeric(1). Age threshold; pixels with age <= cutoffForYoungAge and non-forest presence are flagged as young.

Details

The function proceeds as follows:

1. Determines the set of non-forest columns as all columns in landcoverDT except pixelID, and computes a per-row sum (sumRows).
2. Retrieves pixel ages from NFTSD using pixelID as indices.
3. Selects pixels where sumRows > 0 **and** age <= cutoffForYoungAge; these pixels are considered "young non-forest".
4. Sets those pixels to 0 in **every layer** of LCCras using cell indexing.
5. Builds a new single-layer raster with 1 at those pixel locations and 0 elsewhere (respecting NA from the template), and adds it to LCCras as a layer named youngAge.

Assumptions / requirements

- LCCras is a terra::SpatRaster (possibly multi-layer). Cell indexing with LCCras[pixToChange] <- 0 applies across all layers for the indexed cells.

- `landcoverDT` is a `data.table` with:
 - a `pixelID` column of integer cell indices (1-based) consistent with `LCCras` cell numbering, and
 - all other columns representing non-forest variables whose row-wise sum indicates presence (`sumRows > 0`).
- `NFTSD` is a numeric vector (or similar) indexable by `pixelID`, providing the age (in the same units as `cutoffForYoungAge`) for each cell.
- `cutoffForYoungAge` is a scalar numeric threshold; pixels with age $\leq$ `cutoffForYoungAge` are flagged as "young".

Notes

- The function modifies `LCCras` in place (by reference to the object passed in) and returns the modified raster with an extra `youngAge` layer.
- NA cells in `LCCras` remain NA in the created `youngAge` layer.
- The `landcoverDT` temporary columns `sumRows` and `age` are removed prior to return.

Value

A `terra::SpatRaster` identical to `LCCras` but with:

- all "young non-forest" pixels set to 0 across its existing layers, and
- an additional layer named `youngAge` where flagged pixels are 1, others are 0, and NAs follow the template mask.

References

- Terra package reference manual and vignettes for `SpatRaster` operations (Hijmans, R.J.).
- `data.table` reference: `.SD`, `:=`, and efficient grouping/filtering (Dowle, M., & Srinivasan, A.).

See Also

- `terra::rast()`, `terra::values()`, `terra::setValues()` for raster creation and manipulation.
- `data.table` semantics for `.SD`, in-place updates (`:=`), and fast row/column operations.

Examples

```
## Not run:
library(terra)
library(data.table)

# Build a small template raster
r <- rast(nrows = 3, ncols = 3)
values(r) <- c(NA, 0, 0,
               0, 0, NA,
               0, 0, 0)

# Make a 2-layer LCC raster (copy template twice)
```

```

LCC <- c(r, r)
names(LCC) <- c("LCC1", "LCC2")

# A data.table with pixel indices and two non-forest columns
dt <- data.table(
  pixelID = c(2, 3, 4, 5, 7, 8, 9),
  NF1 = c(1, 0, 0, 2, 0, 0, 1),
  NF2 = c(0, 1, 0, 0, 0, 3, 0)
)

# Age vector indexed by cell ID (length ncell(LCC))
ages <- seq_len(ncell(LCC)) * 5 # toy ages

out <- calcNonForestYoungAge(
  landcoverDT = dt,
  NFTSD = ages,
  LCCras = LCC,
  cutoffForYoungAge = 15
)

out
names(out) # "LCC1" "LCC2" "youngAge"
plot(out$youngAge)

## End(Not run)

```

calcYoungAge

Iteratively calculate youngAge column in FS covariates

Description

Deprecated: use [youngAgeAtYear\(\)](#). This one treats NA ages as young, only sees the fires in fireBufferedListDT (those being fitted), and needs a raster copy per year.

Usage

```

calcYoungAge(
  years,
  annualCovariates,
  standAgeMap,
  fireBufferedListDT,
  cutoffForYoungAge = fireSenseYoungAgeCutoff
)

```

Arguments

years the years over which to iterate

annualCovariates list of data.table objects with pixelID
 standAgeMap template SpatRaster
 fireBufferedListDT data.table containing non-annual burn and buffer pixelIDs
 cutoffForYoungAge Numeric. Default is 15. This is the age below which the pixel is considered "young" -> youngAge column will be 1 if age <= 15

Value

a raster layer with unified stand age and time-since-disturbance values

castCohortData	<i>preparing covariates for fitting modules</i>
----------------	---

Description

preparing covariates for fitting modules

Usage

```
castCohortData(
  cohortData,
  pixelGroupMap,
  lcc,
  ageMap = NULL,
  missingLCC,
  year = NULL,
  cutoffForYoungAge = fireSenseYoungAgeCutoff
)
```

Arguments

cohortData A data.table with columns: pixelGroup, ecoregionGroup, speciesCode, and optionally age, B, mortality, aNPPAct, and sumB.
 pixelGroupMap A RasterLayer with pixel values equal to a pixel group number that corresponds exactly to pixelGroup column in cohortData.
 lcc data.table of dummified landcover
 ageMap a stand age map to assign ages to non-forest LCC used during predict
 missingLCC LCC class to assign forested pixels absent from cohortData must be a character matching a nonForestedLCC group, e.g. 'nonForest_highFlam'
 year numeric representing the year represented by cohortData
 cutoffForYoungAge Numeric. Default is 15. This is the age below which the pixel is considered "young" -> youngAge column will be 1 if age <= 15

Value

a trimmed cohortData with wide-layout and rows for every pixel in lcc

```
chk_duplicatedStartPixels
```

Data checks and assertions for spreadFitRun

Description

chk_duplicatedStartPixels enforces the invariant that no more than one fire can be ignited in a given pixel within a single time interval. When duplicate cells are detected it issues a warning and keeps only the largest fire (by size) for each duplicated pixel.

Usage

```
chk_duplicatedStartPixels(cells, size)

.doDataChecks(moduleName, envir, attribs, fml)
```

Arguments

cells	Integer vector of raster cell indices (pixel IDs) at which fires are scheduled to start.
size	Integer vector the same length as cells, giving the size of each fire (number of pixels). Used to break ties when duplicates exist.
moduleName	Character. Name of the calling SpaDES module, used as a prefix in any error messages thrown.
envir	An environment (typically a SpaDES module's sim object) that is expected to contain the data object named "fireAttributesFireSense_SpreadFit".
attribs	Character. Name of the data object being checked, included in error messages so the caller can identify which input failed.
fml	A model formula passed through to <code>stats::is.empty.model()</code> .

Value

A list with two elements: loci (deduplicated cells) and sizes (corresponding size values).

Called for its side effects. Throws an informative error if any check fails; returns invisibly otherwise.

chooseDomSecFuelClasses

Choose an ELF's dominant and secondary fuel classes

Description

Ranks fuel classes by total treed AGB over a single representative data year (the fit study area), so `fireSenseCovariatesCreate(fuelCovariates = "domSecWetland")` can be told the same `domClass/secClass` for every data year of a fit, instead of choosing independently (and possibly differently) each time it is called. Ties are broken alphabetically by fuel-class name for a deterministic result.

Usage

```
chooseDomSecFuelClasses(
  cohortData,
  pixelGroupMap,
  flammableRTM,
  landcoverDT,
  sppEquiv,
  fuelClassCol,
  sppEquivCol,
  cutoffForYoungAge,
  requiredFuelClasses = NULL
)
```

Arguments

<code>cohortData</code>	A <code>data.table</code> with columns: <code>pixelGroup</code> , <code>ecoregionGroup</code> , <code>speciesCode</code> , and optionally <code>age</code> , <code>B</code> , <code>mortality</code> , <code>aNPPAct</code> , and <code>sumB</code> .
<code>pixelGroupMap</code>	A <code>RasterLayer</code> with pixel values equal to a pixel group number that corresponds exactly to <code>pixelGroup</code> column in <code>cohortData</code> .
<code>flammableRTM</code>	a <code>rasterToMatch</code> with binary values where 1 represents flammable pixels, 0 nonflammable, and NA no data
<code>landcoverDT</code>	As in fireSenseCovariatesCreate() .
<code>sppEquiv</code>	table with species name equivalencies between the kNN and final naming formats. See <code>data("sppEquivalencies_CA", "LandR")</code> . Only necessary if <code>mixedType == 2</code> . If not provided and <code>mixedType == 2</code> , will attempt to use <code>data("sppEquivalencies_CA", "LandR")</code> .
<code>fuelClassCol</code>	As in fireSenseCovariatesCreate() .
<code>sppEquivCol</code>	the column name to use from <code>sppEquiv</code> .
<code>cutoffForYoungAge</code>	As in fireSenseCovariatesCreate() .
<code>requiredFuelClasses</code>	As in fireSenseCovariatesCreate() .

Value

list with domClass and secClass (character); secClass is NA when fewer than two fuel classes are present, and both are NA when there are none (e.g. an ELF with no tree species at all).

cleanUpSpreadFirePoints

*Ensure fire points are located on flammable pixels inside a fire polygon
Intended to be run using Map*

Description

Ensure fire points are located on flammable pixels inside a fire polygon Intended to be run using Map

Usage

```
cleanUpSpreadFirePoints(firePoints, bufferDT, flammableRTM, idCol)
```

Arguments

firePoints	a sf points object representing annual ignitions
bufferDT	a data.table of burned cells, output from bufferToArea
flammableRTM	a rasterToMatch with binary values where 1 represents flammable pixels, 0 nonflammable, and NA no data
idCol	Character. Name of the column in firePoints that uniquely identifies each fire event (e.g. "FIRE_ID").

Value

a list of harmonized points and polygons

climateCovRanges

Fixed ranges of the spread covariates

Description

climateCovRanges is the one table of fixed c(min, max) ranges, one per climate variable of the spread model, that fireSense_spreadFit uses by default (its covFixedRange) instead of the range of each ELF's data, so a coefficient means the same thing in every ELF. The variables and their units: CMD, CMDsm and CMDsp are climate moisture deficit in mm (annual, summer-to-date and spring); cumMDC is a cumulative Monthly Drought Code, an index with no unit (it is not mm). The values are provisional: all four are 0-100 for now, to be replaced after the cumMDC values have been checked. A climate covariate selected for the spread fit that is not in this table stops the fit; add its range here.

Usage

climateCovRanges

Format

A named list of c(min, max).

climateRasterToDataTable	<i>Converts stacks of climate rasters to data.table and optionally subsets to index</i>
--------------------------	---

Description

Converts stacks of climate rasters to data.table and optionally subsets to index

Usage

climateRasterToDataTable(historicalClimateRasters, Index = NULL)

Arguments

- | | |
|--------------------------|--|
| historicalClimateRasters | named list of SpatRaster objects |
| Index | optional list of data.table objects named by fireYear and containing fire buffer indices |

Value

a long-layout data.table of climate values in each pixel and year, at full precision: the x1000 integer storage ([toX1000\(\)](#)) is where they are rounded, as fireSense_spreadPredict does, so a value is stored as the same integer in the fit and in the prediction

cohortsToFuelClasses	<i>Classify pixelGroups by flammability</i>
----------------------	---

Description

Classify pixelGroups by flammability

Usage

```
cohortsToFuelClasses(
  cohortData,
  pixelGroupMap,
  flammableRTM,
  landcoverDT = NULL,
  sppEquiv,
  sppEquivCol,
  cutoffForYoungAge,
  fuelClassCol = fireSenseFuelClassCol,
  requiredFuelClasses,
  asTable = FALSE
)
```

Arguments

cohortData	A data.table with columns: pixelGroup, ecoreregionGroup, speciesCode, and optionally age, B, mortality, aNPAct, and sumB.
pixelGroupMap	A RasterLayer with pixel values equal to a pixel group number that corresponds exactly to pixelGroup column in cohortData.
flammableRTM	a rasterToMatch with binary values where 1 represents flammable pixels, 0 nonflammable, and NA no data
landcoverDT	Optional table of non-forest land cover classes and pixel indices. It will override pixel values in cohortData, if supplied.
sppEquiv	table with species name equivalencies between the kNN and final naming formats. See data("sppEquivalencies_CA", "LandR"). Only necessary if mixedType == 2. If not provided and mixedType == 2, will attempt to use data("sppEquivalencies_CA", "LandR").
sppEquivCol	the column name to use from sppEquiv.
cutoffForYoungAge	age at and below which pixels are considered 'young'
fuelClassCol	the column in sppEquiv that describes unique fuel classes
requiredFuelClasses	A character vector of all the fuel classes that are expected. This is needed because a fuel class may disappear (e.g., Popu_tre) due to succession, causing a downstream failure because the prediction model requires values for it.
asTable	if TRUE, return a data.table with pixelID and one column per layer, restricted to cells where at least one layer is not NA – exactly as .data.table(as.data.frame(<the SpatRaster>, with cell renamed to pixelID, without building the raster. Used by fireSenseCovariatesCreate() .

Value

a SpatRaster of biomass by fuel class as determined by fuelClassCol and cohortData (or a data.table, see asTable).

combine_fuel_classes	<i>Merging and assignment of fuel classes</i>
----------------------	---

Description

Merging and assignment of fuel classes

Usage

```
combine_fuel_classes(df, targetFuelClasses = 5, lowThreshold = 0.05)
```

Arguments

df	created during assessFuelClasses
targetFuelClasses	the number of classes at which to stop further merging
lowThreshold	species with above10PctRelB below this threshold will be merged with like fuel classes, when available, regardless of coefficient

Value

a data.table with assigned fuel classes for each species

combinePlans	<i>Add the size merges to the fire-count plan</i>
--------------	---

Description

A size merge covers every ELF in its members, so a fire-count merge whose members are all in one is replaced by it. Skips stay. The size merges get the fire counts of their members from status.

Usage

```
combinePlans(firePlan, sizePlan, status)
```

Arguments

firePlan	Plan from ELFmergePlan() .
sizePlan	Plan from ELFsizePlan() .
status	ELFfitStatus() of the original ELFs.

Value

A data.table in [ELFmergePlan\(\)](#)'s format.

covsX1000AndSetDF	<i>Convert covariates to/from integers for smaller footprint; make data.frame</i>
-------------------	---

Description

Convert covariates to/from integers for smaller footprint; make data.frame

Usage

```
covsX1000AndSetDF(
  annualList,
  nonAnnualList,
  fireBufferedList,
  fireLocilist,
  paramOrder,
  toX1000Integer = TRUE
)
```

Arguments

annualList	Named list of data.tables of covariates that vary annually. Must have a pixelID column. List name is year.
nonAnnualList	Named list of data.tables of covariates that do not vary each year. Must have a pixelID column. List name is a concatenated set of years separated by underscore.
fireBufferedList	Named list of data.tables of buffered perimeters with pixelID. Must have a pixelID column. List name is yearXXXX where XXXX is the 4 digit year. Column names must be pixelID, buffer with every row a 1 or 0 as to whether that pixel is in or outside the buffer (1 is in the buffer!), and ids with a unique identifier for each fire.
fireLocilist	Named list of data.table of pixelID columns where fires occurred. Columns are size which is number of pixels in the fire
paramOrder	Character vector of the order of covariates that will be used.
toX1000Integer	Logical. If TRUE, then it multiples by 1000 and coerces to integer. If FALSE, then it does the inverse.

Value

a data.table with relevant columns made mutually exclusive

dataYearFor	<i>The data year a year's vegetation comes from</i>
-------------	---

Description

The one rule for matching a year to a vegetation or land-cover data year: the data year closest to, and not after, the year. `fireSense_dataPrepFit` assigns each fire year its data year with it and decides a fit's model structure on it; `fireSense_dataPrepPredict` takes the vegetation of `time(sim)` with it; so a fit and every prediction, hindcast or forecast, read the same data for the same year.

Usage

```
dataYearFor(dataYears, years)

dataYearElement(x, year)
```

Arguments

dataYears	integer years with data, in any order.
years	integer years.
x	a list (anything with names) with one element per data year, named <code>year<dataYear></code> (<code>yearTxt</code>) as <code>fireSense_dataPrepFit</code> names <code>cohortDatas</code> , <code>rstLCCs</code> , ...; any name whose digits are the year will do.
year	one integer year.

Value

`dataYearFor()`: integer, one data year per element of `years`. Stops if a year precedes the first data year: it has no vegetation data.

`dataYearElement()`: the name of the element of `x` that year uses, the one of `dataYearFor(<the years in names(x)>, years)`.

Examples

```
dataYearFor(c(1985, 1990, 2000, 2010, 2020), c(1985, 1999, 2015, 2024))
dataYearElement(list(year2010 = "a", year2020 = "b"), 2015)
```

defaultFireYears	<i>Default fire years for the fireSense modules</i>
------------------	---

Description

The years of fire records that fireSense_dataPrepFit fits to, and over which fireSense_ELFS counts fires to merge ELF's with too few of them. Both modules take their fireYears default from here so that the ELF's and the fit always use the same years.

Usage

```
defaultFireYears()
```

Details

The years run from 1985, the first SCANFI V2 year, to the latest year with historical climate for every tile (`climateData::latestHistoricalYear()`); climate is the last of the inputs to reach a year.

Value

An integer vector of years.

deoptimPopulationSize	<i>Default DEoptim population size for a fit</i>
-----------------------	--

Description

The number of workers, and so the DEoptim population NP, `runDEoptim()` asks for when nCoresNeeded is NULL: $20 + 10 * \text{floor}((nParams + 2) / 5)$, i.e. 30 for 4 to 7 parameters, 40 for 8 to 12 and 50 for 13 to 15. Ten times the number of parameters was tested against other sizes; 40 was a good middle with very little improvement above it, and the current spread ledgers have 10 to 12 parameters, so every current ELF gets 40.

Usage

```
deoptimPopulationSize(nParams)
```

Arguments

nParams	Integer. The number of parameters estimated (length(lower)).
---------	--

Value

An integer.

dtReplaceNAwith0	<i>Replace NAs in a data.table with zeros</i>
------------------	---

Description

Modifies DT in place: for each column in colsToUse, every NA is overwritten with 0 via `data.table::set()`. Useful when a covariate table is later passed to a model that does not tolerate NAs.

Usage

```
dtReplaceNAwith0(DT, colsToUse = NULL)
```

Arguments

DT	A data.table to be modified by reference.
colsToUse	Character vector of column names in DT to process. NULL (default) means every column.

Value

DT, invisibly modified by reference; columns are unchanged where no NAs were present.

ELFFireCounts	<i>Count fires in each ELF</i>
---------------	--------------------------------

Description

Counts natural-cause ignitions, escaped natural ignitions and fire polygons falling in each ELF's fitting area (core plus buffer), per year. An escaped ignition is one that `escapedFires()` keeps: the escape and spread fits use only these.

Usage

```
ELFFireCounts(
  rasWhole,
  firePoints,
  firePolys,
  fireYears,
  pixelAreaHa,
  naturalCauses = c("L", "N"),
  escapeSizeHa = 50,
  overlay = NULL
)
```

Arguments

rasWhole	SpatRaster of ELF layers on one common grid, one layer per ELF, as built by <code>makeELFs()</code> (<code>sim\$ELFs\$rasWhole</code>). Cells are 0 (outside), 1 (buffer) or 2 (core).
firePoints	SpatVector of fire points, as returned by <code>fireregimetools::load_nfdb_points()</code> . Needs columns YEAR, CAUSE and SIZE_HA.
firePolys	SpatVector of fire perimeters, as returned by <code>fireregimetools::load_nbac_polys()</code> . Needs column YEAR. May be NULL when overlay is given.
fireYears	integer vector of the years being fitted.
pixelAreaHa	area of one fitting pixel, in hectares. A polygon counts when its area inside the ELF exceeds this.
naturalCauses	causes treated as natural ignitions.
escapeSizeHa	size (ha) a fire must reach to count as escaped, as in <code>escapedFires()</code> (<code>pixelAreaHa</code> is its pixel area).
overlay	the table from <code>ELFFireOverlay()</code> for <code>rasWhole</code> , <code>firePolys</code> and <code>fireYears</code> , or NULL (default) to compute it. Passing it lets the clipping be done once and cached.

Details

Points are counted on the grid of `rasWhole` with a single `terra::cellFromXY()` lookup. Polygons are intersected with each ELF's area and their clipped area is compared with `pixelAreaHa`, mirroring the size filter the spread fit applies (`fireSense_dataPrepFit.R:966-982`).

Value

A data.table with one row per ELF and year: ELF, year, naturalIgnitions, escapes, firePolygons.

ELFFireOverlay	<i>Clip fire polygons to each ELF, once</i>
----------------	---

Description

Intersects the fire polygons with each ELF's fitting area (core plus buffer) and with its core, and returns one row per clipped piece. `ELFFireCounts()` counts the pieces larger than a pixel in the fitting area, and `ELFRegionStats()` sums the core pieces into a burn rate, so the slow intersection is done once for both. A polygon wholly inside a zone is not clipped; only those crossing its edge are, which is most of the saving.

Usage

ELFFireOverlay(rasWhole, firePolys, fireYears)

Arguments

rasWhole	SpatRaster of ELF layers on one common grid, as in ELFFireCounts() .
firePolys	SpatVector of fire perimeters with column YEAR.
fireYears	integer vector of the years to keep.

Value

A data.table: ELF, zone ("fit" for core plus buffer, "core"), YEAR and ha, the geodesic area of the piece. ELF's no polygon reaches have no rows.

ELFFitStatus	<i>Classify each ELF as zero, few or ok on its fire counts</i>
--------------	--

Description

An ELF with no natural ignitions, or no fire polygons, over the whole fitting window cannot be fitted and is "zero". One below any threshold is "few": it can be fitted, but the thin end of the data drives the escape model's 5-fold CV (`fireSense_ignitionFit.R:463,889`), which needs at least 10 ignited cell-years before some fold is left holding a single row. The escape and spread fits use only escaped fires (see [escapedFires\(\)](#)), so an ELF with many small fires and few large ones is "few" through `minEscapes`.

Usage

```
ELFFitStatus(
  counts,
  minNaturalIgnitions = 50,
  minFirePolygons = 50,
  minEscapes = 5
)
```

Arguments

counts	data.table from ELFFireCounts() .
minNaturalIgnitions, minFirePolygons, minEscapes	thresholds below which an ELF is "few".

Value

A data.table with one row per ELF: ELF, naturalIgnitions, escapes, firePolygons, yearsWithFire, status ("zero", "few" or "ok").

ELFflammableArea	<i>Measure flammable area in each ELF</i>
------------------	---

Description

Counts the land-cover pixels an ELF has that the fit would treat as flammable, i.e. those not in nonflammableLCC. Unclassified cover (VLCE2 code 0) is nonflammable by default, which is why ELFs outside VLCE2’s mapped extent measure at or near zero (8.2, 10.3.2 and 3.2.4 are 0; no-tree-elfs.md §2, elf-8.2-ecoregion.md).

Usage

```
ELFflammableArea(rasWhole, rstLCC, nonflammableLCC = fireSenseNonflammableLCC)
```

Arguments

- rasWhole SpatRaster of ELF layers, as in [ELFfireCounts\(\)](#).
- rstLCC SpatRaster of land cover, the same source the fit uses.
- nonflammableLCC land-cover codes that cannot burn. The default, [fireSenseNonflammableLCC](#), matches fireSense_dataPrepFit’s nonflammableLCC parameter.

Details

This does **not** gate anything. Per Eliot’s 2026-09-11 decision there is no flammable floor – fireSense_spreadFit is allowed to try regardless – so the result is a report, and an input to deciding whether merging an ELF with a neighbour would give it enough to fit.

Value

A data.table with one row per ELF: ELF, flammablePixels, totalPixels, flammableFraction, flammableAreaHa.

elfLabels	<i>Text to print on ELFs</i>
-----------	------------------------------

Description

Text to print on ELFs

Usage

```
elfLabels(ids, labels = c("code", "name", "none"))
```

Arguments

ids	Character. ELF codes, e.g. <code>c("13.1", "6.2.1")</code> .
labels	"code" returns ids; "name" returns the name of the ecozone each ELF lies in, with the code added in brackets where two of ids share a zone name, so that labels stay distinct; "none" returns "".

Value

A character vector the length of ids.

Examples

```
elfLabels(c("13.1", "5.4", "6.2.1", "6.2.2"), "name")
```

ELFmergedName	<i>Name of a merged ELF</i>
---------------	-----------------------------

Description

Name of a merged ELF

Usage

```
ELFmergedName(members)
```

Arguments

members	ids of the ELFs being merged; they must share one base.
---------	---

Value

The shared base followed by the members' last parts in numeric order, e.g. `c("3.2.4", "3.2.1")` gives "3.2.1_4" and `c("12.2", "12.1")` gives "12.1_2".

ELFmergePlan	<i>Decide which ELF's with too few fires to merge, and which not to fit</i>
--------------	---

Description

Every ELF that `ELFfitStatus()` calls "zero" or "few" is considered once, in ELF order. Its partner is the neighbouring ELF with the same base and the same depth – another piece of the same split ecoprovince, or another whole ecoprovince of the same ecozone – that shares the longest core border (ties: more fires, then ELF order) and is not already part of a merge. If the two together reach all three thresholds they are merged; otherwise neither is fitted. An ELF with no such neighbour is not fitted.

Usage

```

ELFmergePlan(
  status,
  neighbours,
  minNaturalIgnitions = 50,
  minFirePolygons = 50,
  minEscapes = 5
)

```

Arguments

status	data.table from <code>ELFfitStatus()</code> .
neighbours	data.table from <code>ELFneighbours()</code> .
minNaturalIgnitions, minFirePolygons, minEscapes	thresholds a merged ELF must reach, as in <code>ELFfitStatus()</code> .

Value

A data.table with one row per decision: action ("merge" or "skip"), ELF (the merged ELF's name, NA for a skip), members (list of the ELF ids involved), naturalIgnitions, escapes and firePolygons (their totals) and reason.

ELFneighbours	<i>Shared core borders between ELF's</i>
---------------	--

Description

Measures how much core border each pair of ELF's shares on the common grid of rasWhole: the number of horizontally or vertically adjacent cell pairs where one cell is core (2) in one ELF and the other is core in the other.

Usage

```
ELFneighbours(rasWhole)
```

Arguments

rasWhole ELF layers on one common grid, as a multi-layer SpatRaster or a named list of SpatRasters (sim\$ELFs\$rasWhole). Cells are 0 (outside), 1 (buffer) or 2 (core).

Value

A data.table with one row per pair of ELFs whose cores touch: ELF1, ELF2, sharedEdges (cell edges) and sharedLength (in the grid's map units).

ELFregionStats	<i>Area, land cover and burn rate of each region</i>
----------------	--

Description

Statistics for [ELFsizePlan\(\)](#). A region is the core (cells equal to 2) of one layer of rasWhole. Land-cover shares come from a regular grid of up to maxSample points in the core, so the result is deterministic. The burn rate is the area of firePolys inside the core, as a percentage of the core per year.

Usage

```
ELFregionStats(
  rasWhole,
  landCover,
  firePolys = NULL,
  fireYears = NULL,
  classes = c(20, 30, 40, 50, 100, 210, 220, 230),
  maxSample = 20000L,
  overlay = NULL,
  members = NULL
)
```

Arguments

rasWhole SpatRaster of region layers on one grid, as in [ELFfireCounts\(\)](#). Cells are 0, 1 or 2 (core). Its map units must be metres.

landCover SpatRaster of land-cover class codes.

firePolys SpatVector of fire perimeters with column YEAR, or NULL (burn rate NA, unless overlay is given).

fireYears years over which firePolys are counted; their number is the divisor of the burn rate.

classes	land-cover codes whose shares are returned. The default is SCANFI's: water, rock, bryoids, shrubs, herbs, coniferous, broadleaf, mixedwood.
maxSample	largest number of sample points per region.
overlay	table from ELFireOverlay() (its "core" rows are used) instead of firePolys, so the clipping is not done again. It is for the ELF's the regions are made of, see members.
members	named list giving, for each region (name of a layer of rasWhole), the ids of the ELF's in overlay it is made of. Default: each region is its own ELF.

Value

A data.table with one row per region: ELF, areaKm2, burnRate (%/yr) and landCover (list column of named numeric vectors of shares, names classes).

ELRunName	<i>The ELF to run for an ELF id</i>
-----------	-------------------------------------

Description

The ELF to run for an ELF id

Usage

```
ELRunName(ELF, plan)
```

Arguments

ELF	an ELF id.
plan	data.table from ELFmergePlan() .

Value

The merged ELF's name if ELF was merged, otherwise ELF.

ELFsArctic	<i>ELFs permanently left out of fireSense</i>
------------	---

Description

ELFs of ecozones 1 and 2 (ids 1.* and 2.*) are never fitted: `runELFs()` leaves them out of every list and map, and `ELFmergePlan()` neither merges them nor uses them as partners.

Usage

```
ELFsArctic(ids)
```

Arguments

ids ELF ids.

Value

The ids in ids that are left out.

ELFsExcluded	<i>ELFs that cannot be fitted</i>
--------------	-----------------------------------

Description

ELFs that cannot be fitted

Usage

```
ELFsExcluded(status)
```

Arguments

status data.table from `ELFfitStatus()`.

Value

character vector of ELF names with status "zero".

ELFsInStudyArea	<i>Identify and rasterize Ecologically-based Low Fractal-dimensional objects within a study area</i>
-----------------	--

Description

Intersects a study area with a set of Ecologically-based Low Fractal-dimensional (ELF) polygons, identifies which ELFs overlap the study area, and constructs a single categorical raster combining each ELF's per-cell classification. A cell in an ELF's core (2) is labelled with that ELF. A cell in the buffer (1) of one or more ELFs is labelled with the ELF, among those, whose core is nearest. All other in-domain cells are 0.

Usage

```
ELFsInStudyArea(studyArea, inputPath, ELFsRaster = NULL, ELFsPolygon = NULL)
```

Arguments

studyArea	A <code>SpatVector</code> (or object coercible to one) defining the area of interest. Will be reprojected to the CRS of ELFs internally.
inputPath	Character. Directory in which to store any downloads produced by <code>ELFtemplateRaster()</code> when <code>ELFsPolygon</code> is not supplied.
ELFsRaster	Optional named list (one entry per ELF) of <code>SpatRaster</code> objects containing the integer values 0, 1, and 2, accessed via <code>ELFsRaster\$rasWhole</code> . If <code>NULL</code> , the per-ELF rasters must be loaded by the caller before invoking this function.
ELFsPolygon	Optional <code>SpatVector</code> of ELF polygons. If <code>NULL</code> (the default), ELFs are generated by calling <code>makeELFs()</code> with a 20 km buffer around <code>rastTemplate</code> .

Details

The function assumes that ELFs (a list-like object with `rasWhole` rasters and an ID field) and `rastTemplate` are available in the calling environment, along with `inputPath`. ELF rasters are expected to contain the integer values 0, 1, and 2, where 2 indicates the core area of the ELF and 1 indicates the buffer for that ELF.

The three layers are combined with `terra::cover()` so that:

- cores override everything else,
- a buffer cell takes the ELF with the nearest core (`terra::distance()`), and
- the 0 background fills remaining in-domain cells.

Value

A categorical `SpatRaster` with a value/ELF ind levels table, where `ELFind` gives the ELF identifier (or "none" for background).

See Also

[makeELFs\(\)](#), [terra::distance\(\)](#), [terra::mosaic\(\)](#), [terra::cover\(\)](#)

 ELFsizePlan

Merge regions that are too small with a similar neighbour

Description

A region with `areaKm2 < minAreaKm2` is small. Its candidate partners are the neighbours of the same group whose land-cover (Bray-Curtis) distance is at most `maxLandCoverDist` and whose burn-rate ratio (larger over smaller) is at most `maxBurnRatio`. For each candidate the merge is simulated (areas summed, land cover and burn rate weighted by area, neighbours combined) and the small regions left with no candidate partner, the orphans, are counted. The candidate with the fewest orphans is accepted, ties going to the smaller land-cover distance and then the smaller burn ratio. This repeats until no candidate is left, so a merged region that is still small can merge again and a group can grow beyond two regions.

Usage

```
ELFsizePlan(
  stats,
  neighbours,
  minAreaKm2 = 35000,
  maxLandCoverDist = 0.35,
  maxBurnRatio = 6,
  group = .ELFparent
)
```

Arguments

<code>stats</code>	data.table from ELFregionStats() : ELF, areaKm2, burnRate, landCover. An optional list column members names the original ids a region stands for (default: its own id), so a region that is already a merge keeps a name ELFmergedName() can build.
<code>neighbours</code>	data.table from ELFneighbours() , with ELF1 and ELF2.
<code>minAreaKm2</code>	regions with a smaller core area are small. NULL or NA turns the rule off.
<code>maxLandCoverDist, maxBurnRatio</code>	the similarity limits above.
<code>group</code>	function of the region ids giving the group within which regions may merge. The default is the ELF base (ELFmergePlan() 's rule).

Value

A data.table with one row per merged region, as [ELFmergePlan\(\)](#) returns for merges: action ("merge"), ELF (the merged name), members (list of the original ids) and reason, plus areaKm2. [mergeELFs\(\)](#) applies it. Zero rows if nothing merges.

ELFsSkipped	<i>ELFs not fitted after merging</i>
-------------	--------------------------------------

Description

ELFs not fitted after merging

Usage

```
ELFsSkipped(plan)
```

Arguments

plan	data.table from ELFMergePlan() .
------	--

Value

character vector of the ELF ids in the plan’s skip decisions.

ELFtemplateRaster	<i>Build a Canada-wide raster template for ELF construction</i>
-------------------	---

Description

Downloads the SCANFI Douglas-fir raster (by default the v2 2020 file on the Arbutus object store), aggregates it by a factor of 8, turns it into a 1/NA mask, loads it into memory, and writes it to disk as a reusable template for building Ecologically-based Low Fractal-dimensional (ELF) objects. Both the remote download and the aggregation step are cached via `reproducible::Cache()`, keyed on `url` and the remote file’s hash, so the template is only rebuilt when the source changes.

Usage

```
ELFtemplateRaster(  
  inputPath,  
  url =  
    "https://object-arbutus.alliancecan.ca/529feda02d404836b0e91b1d360b4df0:predictiveecology/SCANFI_  
)
```

Arguments

- | | |
|-----------|---|
| inputPath | Character. Directory in which to store the downloaded source raster and the resulting <code>rastTemplate_Canada.tif</code> . |
| url | Character. The source raster. It is used only for its footprint and grid, so any SCANFI layer with the national extent will do. The default is the v2 Douglas-fir 2020 file, which has the same extent, 30 m resolution and CRS as the v1.2 file on <code>ftp.maps.canada.ca</code> it replaced, and is smaller (163 MB vs 226 MB). |

Details

The Douglas-fir layer is used purely for its spatial footprint and grid, so the returned values are a mask, not cover. At fact = 8, the v1.2 file has 156,026,794 non-NA cells and the v2 file 156,482,565; 460,121 cells are NA only in v1.2 and 4,350 only in v2, so ELF maps and rasterToMatch built from the template differ by about 0.3% of cells between the two. Remote metadata is checked at most once per week (`notOlderThan = Sys.time() - 60*60*24*7`) to avoid unnecessary network calls.

Value

A `SpatRaster` pointing to `rastTemplate_Canada.tif` inside `inputPath`. The raster covers Canada at roughly 8x the native SCANFI resolution; it is 1 where the aggregated source has data (including zero cover) and NA elsewhere, and is suitable as a template for downstream ELF construction.

See Also

`reproducible::prepInputs()`, `reproducible::Cache()`, `terra::aggregate()`, `terra::writeRaster()`

`emptySpreadCovariates` *Spread covariates that carry no information*

Description

A covariate is empty when it is zero everywhere, or, for fuel biomass, when it is on the `logMinB()` floor everywhere: fuel is stored as log biomass, so a class with no biomass in the buffers is 3.6 everywhere, not 0. The floor is recognised by its value, so the test does not depend on how a column is named. An indicator with any 1, or a fuel class with any biomass, is not empty. NAs are ignored, as `colSums(na.rm = TRUE)` did.

Usage

```
emptySpreadCovariates(dt, cols)
```

Arguments

<code>dt</code>	data.table of covariates.
<code>cols</code>	character vector: the columns of <code>dt</code> to test.

Value

the names among `cols` that are empty.

escapedFires	<i>Fires that escaped</i>
--------------	---------------------------

Description

Fires that escaped

Usage

```
escapedFires(fires, escapeSizeHa, pixSizeHa, sizeCol = "SIZE_HA")
```

Arguments

fires	Fire points or polygons (sf, SpatVector or data.frame) with a size column in ha.
escapeSizeHa	Size (ha) a fire must reach to count as escaped.
pixSizeHa	Pixel area (ha). A fire must also be larger than one pixel.
sizeCol	The size column.

Value

The rows of fires that escaped.

escapeSizePixels	<i>Pixels a fire must burn to count as escaped</i>
------------------	--

Description

The smallest whole number of pixels whose area is at least escapeSizeHa. Shared by the spread fit ([.objfunSpreadFit\(\)](#)) and fireSense's burning, so "escaped" means the same in both.

Usage

```
escapeSizePixels(escapeSizeHa, landscape)
```

Arguments

escapeSizeHa	Size (ha) a fire must reach to count as escaped.
landscape	A SpatRaster (or anything with terra::res()) in metres.

Value

An integer, at least 1.

Examples

```
r <- terra::rast(nrows = 10, ncols = 10, xmin = 0, xmax = 2400, ymin = 0, ymax = 2400)
escapeSizePixels(50, r) # 240 m pixels are 5.76 ha: 9 pixels
```

fewFireELFs

Count fires per ELF and merge ELFs with too few

Description

Merges each ELF with too few fires with a neighbour that shares its base ([ELFmergePlan\(\)](#)), then each region still smaller than `minRegionAreaKm2` with a similar neighbour ([ELFsizePlan\(\)](#)). Arctic ELFs ([ELFsArctic\(\)](#)) are neither counted nor merged.

Usage

```
fewFireELFs(
  ELFs,
  fireYears,
  pixelAreaHa,
  destinationPath,
  nfdbUrl = nfdbPointUrl(),
  nbacUrl = latestNBACUrl(),
  minNaturalIgnitions = 50,
  minFirePolygons = 50,
  escapeSizeHa = 50,
  minEscapes = 5,
  landCoverFile = NULL,
  minRegionAreaKm2 = 35000,
  maxLandCoverDist = 0.35,
  maxBurnRatio = 6
)
```

Arguments

ELFs	List from makeELFs() .
fireYears	Integer vector of years to count fires over.
pixelAreaHa	Area of one pixel of the ELF rasters, in ha.
destinationPath	Directory the fire records are downloaded to.
nfdbUrl, nbacUrl	URLs of the NFDB point and NBAC polygon archives.
minNaturalIgnitions	Fewer natural-cause ignitions than this is too few.
minFirePolygons	Fewer fire polygons than this is too few.

escapeSizeHa	Size (ha) a fire must reach to count as escaped.
minEscapes	Fewer escaped natural fires than this is too few.
landCoverFile	Path to a land-cover map in class codes (SCANFI's), or NULL. With minRegionAreaKm2 it switches on the size rule.
minRegionAreaKm2	After the fire-count plan, regions with a core smaller than this (km2) are merged with a similar neighbour. NULL or NA turns the rule off.
maxLandCoverDist, maxBurnRatio	How similar a neighbour must be for the size rule: largest land-cover (Bray-Curtis) distance and largest burn-rate ratio.

Details

The fire records are read with `reproducible::prepInputs()`, which caches them, and the clipped fire polygons of every ELF ([ELFFireOverlay\(\)](#), the slow step) are cached on the ELF maps, the NBAC release and `fireYears`, so a change to the plan rules does not repeat either.

Value

List: ELFs (the merged maps), status ([ELFFitStatus\(\)](#)), plan (the fire-count and size merges and the skips, in [ELFmergePlan\(\)](#)'s format) and excluded (names of ELFs not fitted).

firePixelsByYear	<i>Pixel IDs burned in each year</i>
------------------	--------------------------------------

Description

Turns fire polygons or a fire-year raster into the per-year pixel lists [youngAgeAtYear\(\)](#) reads. Polygons are rasterized at cell centres, as [makeTSD\(\)](#) does.

Usage

```
firePixelsByYear(firePolys = NULL, fireRaster = NULL, template = NULL)
```

Arguments

firePolys	list named by year ("year2001") of SpatVectors, or NULL entries for years without fires.
fireRaster	SpatRaster whose values are fire years; an alternative to firePolys. It holds one fire per pixel, so earlier fires on the same pixel are not seen.
template	SpatRaster giving the pixels; required with firePolys.

Value

list named by year ("2001") of integer pixelIDs; years with no burned pixel are absent.

fireRecordShapefile	<i>Path to the shapefile in a fire-record archive</i>
---------------------	---

Description

Downloads the archive if needed. The shapefile's name carries the release, so callers key their Cache on the name.

Usage

```
fireRecordShapefile(url, destinationPath, ...)
```

Arguments

url	URL of the archive.
destinationPath	Directory to download and extract into.
...	Passed to <code>reproducible::preProcess()</code> .

Value

Character path(s) of the .shp file(s) in the archive.

fireSenseCloudParameters	<i>Get current state of FireSense Fit parameters as a map</i>
--------------------------	---

Description

Download the FireSense parameter object, strip the list columns and convert to a SpatVector or SpatRaster.

Usage

```
fireSenseCloudParameters(
  url = paste0("https://drive.google.com/file/d/",
    "1oJX7V5wPSj49C6wt59yD9RBplfZ7Cp-f/view?usp=drive_link"),
  targetFile = "fireSenseParams.rds",
  destinationPath = ".",
  useCache = TRUE
)
```

Arguments

url	Http url of the fireSense object with parameters on Google Drive: either the file itself (the default) or the folder that contains targetFile.
targetFile	The name of the file to find when url is a folder. The default is correct for fireSense parameters.
destinationPath	A path where a copy of the downloaded file is saved.
useCache	Ignored. Kept so existing calls still work: the file is always downloaded fresh.

Details

The file is downloaded from Google Drive on every call, so the result is always the current state of the shared parameter file, never a copy left on disk by an earlier call.

Value

The object and the rds file saved to destinationPath.

See Also

[fireSenseCloudParametersMap\(\)](#)

fireSenseCloudParametersMap

Get current state of FireSense Fit parameters as a map

Description

Download the FireSense parameter object, strip the list columns and convert to a SpatVector or SpatRaster. If plot = TRUE, this will also download a map that represents the forested lands of canada, to be plotted on the plot device with the fireSense parameter ELF's that currently have estimated parameters.

Usage

```
fireSenseCloudParametersMap(rasterize = FALSE, plot = TRUE, res = 5000, ...)
```

Arguments

rasterize	Logical. If TRUE, then the function returns the rasterized map. FALSE, the default, is a SpatVector.
plot	Logical. If TRUE, the default, then the map will be plotted to device.
res	The resolution of the raster if rasterize = TRUE
...	Other parameters passed to fireSenseCloudParameters

Value

The map, either SpatRaster or SpatVector

See Also

[fireSenseCloudParameters\(\)](#)

fireSenseCovariatesCreate

Create FireSense Input Covariates

Description

Constructs pixel-level covariates required by FireSense (e.g., mutually exclusive fuel-class indicators, transformed by logMinB, and a youngAge indicator) by combining cohort-derived fuel classes with land cover, flammability, and (optionally) non-forest time-since-disturbance.

Usage

```
fireSenseCovariatesCreate(
  cohortData,
  pixelGroupMap,
  flammableRTM,
  sppEquiv,
  landcoverDT,
  fuelClassCol,
  requiredFuelClasses = NULL,
  sppEquivCol,
  missingLCCgroup,
  nonForestedLCCGroups,
  nonForest_timeSinceDisturbance,
  cutoffForYoungAge,
  nonForestCanBeYoungAge,
  studyAreaName,
  useCache = TRUE,
  rstLCC = NULL,
  treedWetlandLCC = 81,
  treedWetland = TRUE,
  fuelCovariates = c("species", "domSecWetland"),
  domClass = NULL,
  secClass = NULL,
  youngAge = TRUE,
  fuelClassTable = NULL
)
```

Arguments

<code>cohortData</code>	data.table (or similar) of stand/age cohorts used to derive fuel classes.
<code>pixelGroupMap</code>	A raster (SpatRaster) or mapping structure linking cohort groups to pixels.
<code>flammableRTM</code>	A raster (SpatRaster) mask of flammable pixels (non-flammable typically NA/0).
<code>sppEquiv</code>	Species equivalency table used to map species codes to fuel-class groupings.
<code>landcoverDT</code>	data.table with at least pixelID and one or more land-cover indicators (e.g., logical or numeric columns for LCC groups) used in the covariate set.
<code>fuelClassCol</code>	Character name of the fuel-class column produced/expected by sppEquiv.
<code>requiredFuelClasses</code>	A character vector of all the fuel classes that are expected. This is needed because a fuel class may disappear (e.g., Popu_tre) due to succession, causing a downstream failure because the prediction model requires values for it.
<code>sppEquivCol</code>	Character name of the species-equivalency column in sppEquiv.
<code>missingLCCgroup</code>	A single unquoted column name (passed to data.table:::= via eval) that will be set to 1 where all covariates are zero (forested LCC absent from cohortData).
<code>nonForestedLCCGroups</code>	Named vector or list where names correspond to the non-forest land-cover columns in landcoverDT (used to detect non-forest pixels).
<code>nonForest_timeSinceDisturbance</code>	A raster/vector of time-since-disturbance (TSD) for non-forest pixels; indexed by pixelID and used when nonForestCanBeYoungAge = TRUE.
<code>cutoffForYoungAge</code>	Numeric threshold (years). Pixels with TSD $\leq$ this value are considered youngAge.
<code>nonForestCanBeYoungAge</code>	Logical. If TRUE, non-forest pixels can be flagged as youngAge based on nonForest_timeSinceDisturbance and cutoffForYoungAge.
<code>studyAreaName</code>	Character tag used for caching/user tags.
<code>useCache</code>	Logical. If TRUE, enable caching in Cache() calls.
<code>rstLCC</code>	Optional land-cover SpatRaster on the same grid as pixelGroupMap. When given and fuelCovariates = "species", a treedWetland column (see treedWetlandTxt) is added: 1 where the land cover is treedWetlandLCC. When fuelCovariates = "domSecWetland", all tree AGB (all fuel classes) on those pixels is moved into treedWetland_agb (see treedWetlandAgbTxt) instead, and removed from dom_agb_*/sec_agb_* there.
<code>treedWetlandLCC</code>	Land-cover code(s) of treed wetland. Default 81 (NTEMS; also what makeFireSenseLCC() assigns on SCANFI land cover from the wetland layer).
<code>treedWetland</code>	Logical. TRUE (default): treed wetland is a covariate when rstLCC is given, as described there. FALSE: it is not, and its tree AGB stays in the fuel columns like any other forest. A fit sets this once per ELF (fireSense_dataPrepFit's minCovariateProp: too little treed wetland to estimate its coefficient); a prediction follows the fit.

<code>fuelCovariates</code>	"species" (default; one covariate column per fuel class, as assignedFuelClass in sppEquiv) or "domSecWetland": only the AGB columns <code>dom_agb_<domClass></code> and <code>sec_agb_<secClass></code> (the ELF's two fuel classes with the most total treed AGB) and, when <code>rstLCC</code> is supplied, <code>treedWetland_agb</code> . AGB of any other fuel class is not a covariate.
<code>domClass, secClass</code>	Only used when <code>fuelCovariates = "domSecWetland"</code> . The fuel classes to use for <code>dom_agb_*/sec_agb_*</code> . NULL (the default) chooses them from this call's own data (highest total AGB first); a prediction should instead pass the classes the fit chose (e.g. from <code>chooseDomSecFuelClasses()</code>), so the columns it builds match the fitted model's terms even when a different class dominates the prediction area. <code>secClass = NA_character_</code> means no secondary class (fewer than two fuel classes present).
<code>youngAge</code>	logical. TRUE (default): build <code>youngAge</code> from cohort ages and (with <code>nonForestCanBeYoungAge</code>) non-forest time since disturbance, and zero fuel, non-forest land cover and treed wetland where it is 1. FALSE: no <code>youngAge</code> column and nothing zeroed, so the caller can resolve <code>youngAge</code> per year (see <code>youngAgeAtYear()</code>) and apply <code>makeMutuallyExclusive()</code> itself; <code>cutoffForYoungAge</code> and <code>nonForestCanBeYoungAge</code> are then unused.
<code>fuelClassTable</code>	Optional, the result of <code>cohortsToFuelClasses(..., asTable = TRUE)</code> for this call's <code>cohortData</code> , <code>pixelGroupMap</code> , <code>flammableRTM</code> , <code>landcoverDT</code> , <code>sppEquiv</code> , <code>sppEquivCol</code> , <code>fuelClassCol</code> , <code>requiredFuelClasses</code> and <code>cutoffForYoungAge</code> . It is used instead of building it again, so a caller making several covariate sets (e.g. "species" and "domSecWetland") from the same cohorts builds the fuel classes once. It is copied, not modified. With <code>youngAge = FALSE</code> it must have been made with <code>cutoffForYoungAge = -1</code> .

Details

The function:

1. Calls `cohortsToFuelClasses()` to derive fuel-class rasters from cohorts.
 2. Converts these to a long data table keyed by `pixelID` and joins to `landcoverDT`.
 3. Verifies there are no NA values across covariate columns and raises an error otherwise.
 4. Marks pixels with all-zero covariates as the provided `missingLCCgroup`.
 5. Enforces mutual exclusivity among the specified fuel/landcover columns.
 6. Applies `logMinB` to all fuel-class columns (excluding `youngAge`).
 7. If `nonForestCanBeYoungAge = TRUE`, augments `youngAge` for non-forest pixels using `nonForest_timeSinceDisturbance` and the `cutoffForYoungAge`.
- All covariate columns must be non-NA; the function stops with an error if any NA are detected after joining `fuelClasses` and `landcoverDT`.
 - Pixels where all covariates sum to zero are flagged by setting `missingLCCgroup := 1`.
 - When `nonForestCanBeYoungAge = TRUE`, `youngAge` for non-forest is derived using `nonForest_timeSinceDisturbance` and `cutoffForYoungAge`. Any existing `youngAge` from fuel classes is combined with this non-forest contribution.

- Once youngAge is final, mutual exclusivity is enforced via makeMutuallyExclusive() using all fuel-class columns, the land-cover columns (excluding pixelID) and treedWetland (if rstLCC is supplied); youngAge is made exclusive to them.
- Fuel-class columns (excluding youngAge) are transformed with logMinB.
- When fuelCovariates = "domSecWetland", the per-species fuel-class columns are collapsed to dom_agb_*, sec_agb_* (other classes are dropped; and treedWetland_agb, if rstLCC is supplied) before mutual exclusivity and logMinB are applied to them, exactly as for the per-species columns.

Value

A data.table named spreadCovariates containing one row per pixelID, with:

- Mutually exclusive fuel-class covariate columns (log-transformed by logMinB).
- Land-cover indicator columns from landcoverDT.
- A youngAge indicator (possibly augmented by non-forest TSD logic).

Expectations/Assumptions

- fuelClasses returned by cohortsToFuelClasses() includes one column per fuel class, plus optionally youngAge, and a cell (renamed to pixelID here).
- landcoverDT must contain a pixelID column aligned to rasters by cell index.
- nonForestedLCCGroups names correspond to column names in landcoverDT.
- nonForest_timeSinceDisturbance is indexable by pixelID (e.g., vector aligned to raster cells).

See Also

[cohortsToFuelClasses](#), [makeMutuallyExclusive](#), [logMinB](#), [putBackIntoRaster](#), [calcNonForestYoungAge](#)

Examples

```
## Not run:
covs <- fireSenseCovariatesCreate(
  cohortData = cohortsDT,
  pixelGroupMap = pgRas,
  flammableRTM = flammRas,
  sppEquiv = sppEquivDT,
  landcoverDT = landcoverDT,
  fuelClassCol = "fuelClass",
  sppEquivCol = "sppEquivCol",
  missingLCCgroup = "LCC_missing",
  nonForestedLCCGroups = c(NonForest = 1),
  nonForest_timeSinceDisturbance = NF_TSD,
  cutoffForYoungAge = 20,
  nonForestCanBeYoungAge = TRUE,
  studyAreaName = "MyArea",
  useCache = TRUE
)
```

```
## End(Not run)
```

```
fireSenseNonflammableLCC
```

Non-flammable land-cover codes for the FireSense land cover

Description

The single source of truth for which codes in the land cover built by `makeFireSenseLCC()` are non-flammable. `fireSense_dataPrepFit` and `fireSense_dataPrepPredict` take their `nonflammableLCC` parameter default from this constant, and `makeFireSenseLCC()` and `ELFFflammableArea()` take theirs from it too, so the land cover a fit/prediction is built from and the land cover it is run on cannot silently disagree on what counts as flammable.

Usage

```
fireSenseNonflammableLCC
```

Format

numeric vector, `c(0, 20, 30, 31, 32, 33)`.

Details

Every code below comes from `makeFireSenseLCC()`'s two land-cover sources, `LandR::prepInputs_NTEMS_LCC_FA0()` and `LandR::prepInputs_SCANFI_LCC_FA0()` (documented in the "Data codes" comment in `LandR/R/prepInputs_NTEMS.F` and `LandR/R/maps.R`), plus SCANFI's raw-class recode in `LandR::convert_SCANFI_LCC_codes()`:

- 0: no data / unclassified – non-flammable.
- 20: water – non-flammable.
- 30: rock/exposed land, SCANFI's combined rock+exposed class (`LandR::convert_SCANFI_LCC_codes()`) – non-flammable. This was missing from the modules' old default, so SCANFI-derived rock entered fits as flammable non-forest.
- 31: snow/ice (NTEMS) – non-flammable.
- 32: rock/rubble (NTEMS) – non-flammable.
- 33: exposed/barren land (NTEMS) – non-flammable.
- 40: bryoids – flammable ground cover, not included.
- 50: shrubs – flammable, not included.
- 80, 81: wetland, wetland-treed – flammable, not included.
- 100: herbs – flammable, not included.
- 210, 220, 230: coniferous, broadleaf, mixedwood forest – flammable, not included.
- 240: FAO-forest disturbed code (disturbedCode in `LandR::prepInputs_SCANFI_LCC_FA0()`/`LandR::prepInputs_N`) – flammable forest, not included.

fireSenseSharedDefaults

Defaults shared by the fireSense modules and functions

Description

The fireSense_dataPrepFit and fireSense_dataPrepPredict modules each have parameters with these names, and a fit and the predictions made from it are only consistent if both use the same values; fireSense_spreadFit takes its runawayEdge* defaults from here too. The modules take their parameter defaults from these constants, as they do nonflammableLCC from [fireSenseNonflammableLCC](#), and the functions here that take the same arguments default to them too.

Usage

fireSenseForestedLCC

fireSenseYoungAgeCutoff

fireSenseNonForestCanBeYoungAge

fireSenseFlammabilityThreshold

fireSenseFuelClassCol

fireSenseIgAggFactor

fireSenseSCANFIVersion

fireSenseRunawayEdgeFrac

fireSenseRunawayEdgeMin

Format

Vectors of length 1, except fireSenseForestedLCC (numeric, length 5).

Details

- fireSenseForestedLCC: land-cover codes treated as forest, in the land cover built by [makeFireSenseLCC\(\)](#): treed wetland (81), coniferous (210), broadleaf (220), mixedwood (230) and disturbed forest land (240).
- fireSenseYoungAgeCutoff: age (years) at and below which a pixel is youngAge.
- fireSenseNonForestCanBeYoungAge: whether burned non-forest is youngAge until it passes fireSenseYoungAgeCutoff.
- fireSenseFlammabilityThreshold: minimum proportion of flammable fine-resolution land cover for a coarser pixel to be flammable.

- fireSenseFuelClassCol: the column of sppEquiv that defines fuel classes.
- fireSenseIgAggFactor: aggregation factor (cells per side) for the ignition and escape co-variates.
- fireSenseSCANFIVersion: the SCANFI land-cover version `makeFireSenseLCC()` reads for non-forest land cover (passed to `LandR::prepInputs_SCANFI_LCC_FAO()`'s `dataVersion`).
- fireSenseRunawayEdgeFrac, fireSenseRunawayEdgeMin: the defaults of `runawayEdgeFrac` and `runawayEdgeMin` of `.objfunSpreadFit()` and `runDEoptim()`: a simulated fire is a runaway when it burns at least $\max(\text{runawayEdgeMin}, \text{ceiling}(\text{runawayEdgeFrac} * n))$ of the n pixels of its buffer's edge ring.

fireSenseUtils-constants

Hard-coded name/text constants

Description

Character (or character-vector) constants used across the package and downstream modules to keep column names, identifiers, and other free-text tokens consistent. All such constants end in Txt.

Format

- polygonIDTxt: character(1). Name of the polygon-ID column (currently "polygonID").
- nonNFColNamesTxt: character vector of bookkeeping column names that are *not* non-forest landcover classes (currently "pixelID" and polygonIDTxt); used with `base::setdiff()` to select the non-forest landcover columns when summing/aggregating across rows (see `makeTSD()`).
- yearTxt: character(1). Token used as a year-column name or as a prefix on year-suffixed layer/column names (currently "year").
- youngAgeTxt: character(1). Name of the "young-age" cohort class (currently "youngAge").
- ignitionsTxt: character(1). Name of the ignitions column (currently "ignitions").
- escapesTxt: character(1). Name of the escapes column (currently "escapes").
- spreadFitAdditionalColNamesTxt: character vector of extra simList-slot/column names attached to spread-fit outputs ("numIterations", "objFunVal", "params", "sppEquiv", "nonForestedLCCGroups", "missingLCCgroup", "covMinMax_spread"). `covMinMax_spread` is what prediction needs to rescale covariates exactly as the fit did.
- ignitionFitAdditionalColNamesTxt: character vector of the list-column names attached to ignition-fit ledger rows ("fireSense_IgnitionFitted", "fireSense_EscapeFitted").
- spreadInterceptTxt: character(1). Name of the spread model's intercept, in a formula's design (`spreadDesignCols()`), in lower/upper and in a ledger row's params (currently "(Intercept)", what `stats::lm()` calls it).
- spreadFitCovCentreTxt: character(1). Name of the ledger column that holds the co-variate centres of a fit made with an intercept (currently "covCentre_spread"). It is not in `spreadFitAdditionalColNamesTxt`: a fit without an intercept writes no such column.
- treedWetlandAgbTxt: character(1). Name of the pooled treed-wetland biomass column built by `fireSenseCovariatesCreate(fuelCovariates = "domSecWetland")` (currently "treedWetland_agb").

fitDiagnostics	<i>Diagnostics of a fitted spread model</i>
----------------	---

Description

The checks made by hand on the FireSense phase-2 fits (September 2026), as functions, so that fireSense_spreadFit can make them after every fit.

Usage

```
spreadProbSummary(p, ceiling, tol = 0.002, binWidth = 0.001)

combineSpreadProbSummaries(x)

simulateFireSizes(
  pop,
  fn = .objfunSpreadFit,
  fnArgs = list(),
  cl = NULL,
  seed = 1L
)

scoreFireSizes(
  sims,
  probs = c(0.01, 0.1, 0.25, 0.5, 0.75, 0.9, 0.99),
  largeFire = 174
)

linkSaturation(sims, probsP = c(0.01, 0.1, 0.25, 0.5, 0.75, 0.9, 0.99))

coefIdentifiability(pop, lower, upper, zMin = 1)

profileCoefficients(
  best,
  pop,
  fn = .objfunSpreadFit,
  reps = 10L,
  cl = NULL,
  seed = 1L,
  fnArgs = list(),
  coefs = covariateCoefs(names(best)),
  probsPop = c(0.05, 0.25, 0.5, 0.75, 0.95)
)

identifiedInIsolation(ident, profile, seMult = 2)

fitConvergence(DE)
```

Arguments

p	Numeric. Spread probabilities.
ceiling	The link's upper asymptote, <code>maxAsymptote</code> .
tol	A probability within <code>tol</code> of ceiling counts as at the ceiling.
binWidth	Width of the bins in which p is counted, so summaries of different years can be added up and still give quantiles.
x	A list of <code>spreadProbSummary()</code> results with the same <code>binWidth</code> .
pop	A matrix or <code>data.table</code> of parameter sets, one per row, with column names.
fn	The objective; <code>.objfunSpreadFit()</code> .
fnArgs	Named list of further arguments to <code>fn</code> , as for <code>rescorePopulation()</code> . The fit data (<code>landscape</code> , <code>annualDTx1000</code> , ...) are taken from the workers' global environment if absent, so a held-out prediction passes its own years here.
cl	A cluster, or <code>NULL</code> to run in this session.
seed	Every parameter set is evaluated with the same seeds (<code>seed</code> , or <code>seed + i</code> for replicate <code>i</code> of the profile), so differences between them are not luck.
sims	The output of <code>simulateFireSizes()</code> .
probs	Quantiles of fire size to compare.
largeFire	Size in pixels above which a fire counts as large (174 pixels = 1000 ha at 5.76 ha per pixel).
probsP	Quantiles of spread probability to report.
lower, upper	The bounds the fit used, named as the columns of <code>pop</code> .
zMin	The <code>zPop</code> at or above which the population is taken to pin a coefficient's sign.
best	Named numeric: the parameter set to profile around, normally the best member by replicated mean (<code>bestByReplicatedMean()</code>).
reps	Evaluations of each profile point.
coefs	The coefficients to profile; all covariate coefficients by default.
probsPop	Population quantiles to evaluate each coefficient at, besides 0.
ident	The output of <code>coefIdentifiability()</code> .
profile	The output of <code>profileCoefficients()</code> .
seMult	Dropping a coefficient must worsen the objective by more than <code>seMult</code> standard errors.
DE	The output of <code>runDEoptim()</code> : a list of <code>DEoptim</code> results, one per chunk of generations.

Details

- `simulateFireSizes()` simulates the observed fires from each of several parameter sets, with the "too burny" gate lifted, and returns the simulated sizes.
- `scoreFireSizes()` compares those with the observed sizes: per fire, per year, and by quantile. It never uses the objective's value.

- `linkSaturation()` reports how many pixel-years sit at the spread-probability ceiling. When most do, fire size is decided by the ceiling alone ("too many medium, too few large fires").
- `coefIdentifiability()` measures, for each covariate coefficient, how tightly the final population pins it down.
- `profileCoefficients()` sets each coefficient in turn to 0 and to values across the population's range, other coefficients held at the best member, and re-scores.
- `identifiedInIsolation()` combines the two: a coefficient is identified in isolation when the population pins its sign ($zPop \geq zMin$) and dropping it (setting it to 0) makes the objective clearly worse.
- `fitConvergence()` summarises the objective across the fit's generations.

Value

`spreadProbSummary()`: a list with `n`, `atCeiling` (a count), `counts` (per bin) and `binWidth`.

`combineSpreadProbSummaries()`: their sum, as one summary.

`simulateFireSizes()`: a `data.table`, one row per parameter set $\times$ fire $\times$ replicate: `member` (row of `pop`), `yr`, `rep`, `initialLocus`, `ids`, simulated `sim` and observed size, in pixels. A year the objective will not simulate even with its gate lifted keeps its fires with `sim NA`. `attr(, "spreadProb")` is a list of `spreadProbSummary()`s, one per member.

`scoreFireSizes()`: a one-row `data.table`. `fireBias` and `fireRMSE` are the mean and root mean square of $\log_{10}(\text{simulated} / \text{observed})$ per fire (0.3 is a factor of 2); `year*` the same for annual area burned; `yearIn90pct` the share of years whose observed area lies within the 5-95% range of simulated years; `simsOver10xObs` and `simsUnderTenthObs` the share of simulated fires more than 10 times too large or too small; `AD` the Anderson-Darling statistic between simulated and observed sizes; `qObs_*/qSim_*` the quantiles in probs.

`linkSaturation()`: a `data.table`, one row per member: `pixelYears`, `propAtCeiling` and the spread-probability quantiles `p_*` (bin midpoints).

`coefIdentifiability()`: a `data.table`, one row per covariate coefficient: the population median `popMedian`; `spread90`, the 5-95% range across members as a share of the bound width; `atBound`, the share of members within 5% of the width from a bound; $zPop = |popMedian| / ((q95 - q5) / 3.29)$, the median in units of the population's spread (a normal's 5-95% range is 3.29 sd); `rSlope`, the correlation across members with `hillSlope1` (the link uses `hillSlope1 * X b`, so the two trade off by construction; NA when the population has no `hillSlope1` column, as now that it is fixed at 1); and `signPinned = zPop >= zMin`. With three runs each of four ELFs, `zPop >= 1` found the coefficients whose sign every run agreed on with 94% precision and 44% recall.

`profileCoefficients()`: a `data.table`, one row per coefficient $\times$ profile point: `coef`, `at` (its value there; `isZero` marks the drop test), mean of the objective over reps, and `delta` and `deltaSE`: the mean paired difference from best (same seeds) and its standard error. A positive `delta` means the point fits worse than best. A reference row (`coef = ""`) holds best itself.

`identifiedInIsolation()`: `ident` with `dropDelta`, `dropDeltaSE`, `dropMatters` and `identified` (`signPinned & dropMatters`) added.

`fitConvergence()`: a `data.table`, one row per chunk: `generation` (its last), `bestval` (the best value so far) and the median and standard error of the median of the population's values (fail values of `1e6` and above left out).

fitLedgerFolders

The Google Drive folder of the shared fit ledgers

Description

The folder that holds the spread-fit and ignition-fit ledger files (see [spreadFitFilenameFor\(\)](#) and [ignitionFitFilenameFor\(\)](#)). The modules that write and read the ledgers (`fireSense_spreadFit`, `fireSense_ignitionFit`, `fireSense_dataPrepFit`, `fireSense_ELFS`) use these as the defaults of their `spreadFitGoogleDriveFolder` and `ignitionFitGoogleDriveFolder` parameters, and [latestSpreadFits\(\)](#) and [latestIgnitionFits\(\)](#) use them as the default `cloudFolderID`.

Usage

```
spreadFitLedgerFolder(
  folder = getOption("fireSenseUtils.spreadFitLedgerFolder",
    "https://drive.google.com/drive/folders/1X9-mRjyLMNpgkP_cfqhbr_AQEPOsVCHF")
)

ignitionFitLedgerFolder(
  folder = getOption("fireSenseUtils.ignitionFitLedgerFolder", spreadFitLedgerFolder())
)
```

Arguments

`folder` The folder (url or id). An explicit value wins over the option.

Value

A character string, the folder's url or id.

Options

To use another folder, set an option; every module and function then follows it.

`fireSenseUtils.spreadFitLedgerFolder` the spread-fit ledger folder (url or id). Default: the shared folder of the `fireSense` project.

`fireSenseUtils.ignitionFitLedgerFolder` the ignition-fit ledger folder (url or id). Default: the value of the spread-fit ledger folder, so one option moves both.

<code>fixedLogisticPars</code>	<i>The logistic parameters fixed at 1, not fitted</i>
--------------------------------	---

Description

hillSlope1 and inflectionPoint1, in the position they take in the link, right after maxAsymptote. DEoptim's par, lower and upper do not contain them; the objective, the validation and the gates insert them (fixLogisticPars()), and fireSense_spreadFit restores them in the ledger's parameter sets from this vector.

Usage

`fixedLogisticPars`

Format

A named numeric vector.

<code>fuelClassPrep</code>	<i>Calculate proportional burn of landcover and tree species</i>
----------------------------	--

Description

Calculate proportional burn of landcover and tree species

Usage

```
fuelClassPrep(  
  pixelGroupMap,  
  cohortData,  
  rstLCC,  
  nonflammableLCC,  
  fires,  
  nonforestLCC,  
  yearRange  
)
```

Arguments

- | | |
|----------------------------|--|
| <code>pixelGroupMap</code> | A RasterLayer with pixel values equal to a pixel group number that corresponds exactly to pixelGroup column in cohortData. |
| <code>cohortData</code> | A data.table with columns: pixelGroup, ecoregionGroup, speciesCode, and optionally age, B, mortality, aNPPAct, and sumB. |
| <code>rstLCC</code> | a landcover map |

```

nonflammableLCC      nonflammable landcover in rstLCC
fires                 a single sf or SpatVector object of fire polygons containing a YEAR column
nonforestLCC          vector or list of vectors of flammable non-forest LCC
yearRange             the range of years represented by this landscape

```

Value

data.table with cell, biomass, tree species or LCC for non-forest, and year of fire

Examples

```

# fuelClassPrep(
#   sim$pixelGroupMap2011, sim$cohortData2011, sim$rstLCC2011,
#   rstLCC = sim$rstLCC2011, nonForestLCC = sim$nonForestLCCGroups,
#   nonflammableLCC = P(sim)$nonflammableLCC,
#   fires = do.call(rbind, sim$spreadFirePolys),
#   yearRange = c(2012, 2020)
# )

```

fuelLinearRange	<i>Range that puts linear fuel biomass on the scale the spread fit uses</i>
-----------------	---

Description

Fuel covariates of the spread model are biomass on the LINEAR scale, divided by a fixed 1e4. The division is not done to the data: it is the covMinMax given to [rescaleKnown2\(\)](#), which is affine and does not clamp, so $c(0, 1e4)$ is exactly biomass / 1e4. The constant is fixed, not $\max(\text{biomass})$, so that `fireSense_spreadPredict` reproduces the fit's scaling in every year from the stored `covMinMax_spread` alone. It is also how a fit is recognised as linear: see [isLinearFuelRange\(\)](#).

Usage

```
fuelLinearRange
```

fuelLogToLinear	<i>Fuel biomass from the log scale back to the linear scale</i>
-----------------	---

Description

`fireSenseCovariatesCreate()` returns fuel-class biomass as [logMinB\(\)](#): logged, with everything below $\exp(\log(100) - 1)$ (36.8) raised to that floor. On that scale the spread model cannot use the fuel gradient: on ELF 5.3.2, 45% of pixels sit on the floor and the treed ones fall in 16% of the covariate range (rescaled sd 0.05), so the coefficient estimates little more than treed against treeless. On the linear scale the treed pixels span twice the range and an absent fuel is exactly 0, so it contributes exactly nothing.

Usage

```
fuelLogToLinear(x)
```

Arguments

x Numeric vector of fuel biomass as returned by [logMinB\(\)](#).

Details

This undoes the log where the spread model needs it, instead of removing it at the source, because `fireSenseCovariatesCreate()` also builds the ignition covariates and its output is cached for every fitted polygon. Both `fireSense_spreadFit` and `fireSense_spreadPredict` call this function, so the fit and the prediction cannot drift apart.

Value

Biomass on the linear scale; values on the [logMinB\(\)](#) floor become 0.

getFirePoints_NFDB	<i>Get Fire SpatialPoints from Canadian Fire Database</i>
--------------------	---

Description

Get Fire SpatialPoints from Canadian Fire Database

Usage

```
getFirePoints_NFDB(
  url = NULL,
  studyArea = NULL,
  rasterToMatch = NULL,
  redownloadIn = 1,
  years = 1991:2017,
  fireSizeColName = "SIZE_HA",
  NFDB_pointPath
)
```

Arguments

url	Passed to <code>prepInputs</code>
studyArea	A <code>SpatialPolygons*</code> object used as the principle study region, passed to prepInputs .
rasterToMatch	A <code>RasterLayer</code> objects to use as the template for all subsequent raster operations (i.e., the one used throughout the simulation).
redownloadIn	Numeric Time in YEARS that we tolerate the data to be "old" i.e. 0.5 would mean "redownload data older than 6 months"
years	Numeric vector of consecutive years to fetch.

fireSizeColName
Character describing the name of the column containing fire size information.

NFDB_pointPath Passed to destinationPath in prepInputs

Value

A sf spatial points object, retaining all columns of the source data (notably CAUSE, YEAR and SIZE_HA). When rasterToMatch is supplied, an additional size column gives fire size in pixels.

getFirePoints_NFDB_V2 *Get Fire SpatialPoints from Canadian Fire Database*

Description

Get Fire SpatialPoints from Canadian Fire Database

Usage

```
getFirePoints_NFDB_V2(
  url = NULL,
  studyArea = NULL,
  redownloadIn = 1,
  years = 1991:2017,
  fireSizeColName = "SIZE_HA",
  NFDB_pointPath = NULL,
  fun = "sf::st_read",
  plot = FALSE
)
```

Arguments

url	Passed to prepInputs
studyArea	A SpatialPolygons* object used as the principle study region, passed to prepInputs .
redownloadIn	Numeric Time in YEARS that we tolerate the data to be "old" i.e. 0.5 would mean "re-download data older than 6 months"
years	Numeric vector of consecutive years to fetch.
fireSizeColName	Character describing the name of the column containing fire size information.
NFDB_pointPath	Passed to destinationPath in prepInputs
fun	Character. The function (as a string, e.g., "sf::st_read") used to load the downloaded data. Forwarded to prepInputs(fun = ...), and used to load already-downloaded data, so the class returned does not depend on whether a download was needed.
plot	logical indicating whether to produce plot of fire points. Default FALSE.

Value

A sf spatial points object (or whatever class fun returns), retaining all columns of the source data (notably CAUSE, YEAR and SIZE_HA).

getFirePolygons

Download and prepare fire data from National Fire Database

Description

Download and prepare fire data from National Fire Database

Usage

```
getFirePolygons(url, years, useInnerCache = FALSE, ...)
```

Arguments

url	Character (or missing). URL to the fire polygons archive (typically an NBAC *_shp.zip on the CFS server). When missing, latestNBACurl() is queried so the newest NBAC release available on the CFS download index is used; this means re-running the same code later may pick up a newer dataset. Pass an explicit URL to pin to a specific release when reproducibility matters.
years	years to filter fire polygons by
useInnerCache	logical indicating whether to cache the prepInputs call
...	additional arguments passed to reproducible::prepInputs()

Value

list of fire polygons by year

See Also

[latestNBACurl\(\)](#) for the discovery helper used when url is missing.

`harmonizeBufferAndPoints`*Cleaning up the polygon points*

Description

Mostly this is about 2 things:

1. remove fires that were so small that they take less than 1 pixel so they are not in the buff object but are in the cent object.
2. the centroid cell is in a buffer or otherwise nonburnable cell (e.g., water). For 1) remove these from the centroid data. For 2) this function will search in the neighbourhood for the next closest pixel that has at least 7 available neighbours that can burn. If not, remove these.

Mostly this is about 2 things:

1. remove fires that were so small that they take less than 1 pixel so they are not in the buff object but are in the cent object.
2. the centroid cell is in a buffer or otherwise nonburnable cell (e.g., water). For 1) remove these from the centroid data. For 2) this function will search in the neighbourhood for the next closest pixel that has at least 7 available neighbours that can burn. If not, remove these.

Usage

```
harmonizeBufferAndPoints(cent, buff, ras, idCol = "FIRE_ID")
```

```
harmonizeBufferAndPoints(cent, buff, ras, idCol = "FIRE_ID")
```

Arguments

<code>cent</code>	List of points as <code>SpatialPointsDataFrame</code>
<code>buff</code>	List of data.table objects with 3 columns, "buffer" which is 1 (in the fire) or 0 (in a buffer), ids which are the fire ids which MUST match the ids in the cent.
<code>ras</code>	The raster that created the pixelIDs in the buff.
<code>idCol</code>	The column name as a character string with the fire ids. Defaults to "FIRE_ID".

harmonizeFireData	<i>Harmonize fire data</i>
-------------------	----------------------------

Description

Outer wrapper on spread fire polygon data munging that does several things:

1. ensure buffered fires are entirely in studyArea;
2. ensure every fire has a corresponding ignition point, and vice versa;
3. ensure these points are flammable.

Usage

```
harmonizeFireData(
  firePolys,
  flammableRTM,
  spreadFirePoints,
  areaMultiplier,
  minSize,
  pointsIDcolumn = "FIRE_ID",
  cores = 1
)
```

Arguments

firePolys	the semi-processed fire polys, with field matching pointsIDcolumn
flammableRTM	a rasterToMatch with binary values where 1 represents flammable pixels, 0 nonflammable, and NA no data
spreadFirePoints	the ignition points corresponding to firePolys
areaMultiplier	Either a scalar that will buffer areaMultiplier * fireSize or a function of fireSize. See ?fireSenseUtils::bufferToArea.
minSize	an alternative to areaMultiplier, typically used when fires are small
pointsIDcolumn	the name of the column denoting fire ids in both spreadFirePoints and firePolys
cores	Positive integer.Number of CPU cores (threads) to use for parallel operations.

harmonizeFireDataDeps *Functions whose code affects what [harmonizeFireData\(\)](#) returns*

Description

`reproducible::Cache()` digests only the called function's own code, so a cached [harmonizeFireData\(\)](#) call keeps returning its old result when a function it *calls* changes. Pass this list as that call's `.cacheExtra` and those functions become part of the key.

Usage

```
harmonizeFireDataDeps()
```

Value

A list of functions, to pass as `reproducible::Cache()`'s `.cacheExtra`.

See Also

[harmonizeFireData\(\)](#), [makeFireSenseLCCDeps\(\)](#)

hindcastObserved	<i>Observed fire in an ELF core in one year</i>
------------------	---

Description

What the hindcast simulation is compared with: natural-cause ignitions, escapes, area burned and the fire sizes of year, all inside the ELF core (the buffer is excluded). Ignitions and escapes are [ELFFireCounts\(\)](#) on the core, so they use the fit's definitions: NFDB points with CAUSE in naturalCauses of any size are ignitions, and those that [escapedFires\(\)](#) keeps (larger than a pixel and at least escapeSizeHa) are escapes. Area burned is the core-clipped area of the fire polygons ([ELFFireOverlay\(\)](#)), given for the natural-cause polygons and for polygons of every cause, because the simulation ignites natural fires only but the spread model was fitted to polygons of every cause.

Usage

```
hindcastObserved(
  coreRaster,
  firePoints,
  firePolys,
  year,
  escapeSizeHa = 50,
  pixelAreaHa = prod(terra::res(coreRaster))/10000,
  naturalCauses = c("L", "N"),
  polyCauseCol = "FIRECAUS",
  naturalPolyCauses = "Natural"
)
```

Arguments

coreRaster	SpatRaster of the ELF core on the simulation grid: cells outside the core are NA or 0 (rasterToMatchELF in the fireSense simulation).
firePoints	SpatVector of fire points with columns YEAR, CAUSE and SIZE_HA (ignitionFirePoints).
firePolys	SpatVector of fire perimeters with column YEAR, or a list of them as firePolys is in the simulation (NULL elements are dropped).
year	The year to describe.
escapeSizeHa	Size (ha) a fire must reach to count as escaped.
pixelAreaHa	Area of one pixel (ha). Default: the area of a coreRaster pixel.
naturalCauses	Values of CAUSE in firePoints that are natural ignitions.
polyCauseCol	Column of firePolys holding the cause. If it is missing, the natural-cause values of the result are NA, with a warning.
naturalPolyCauses	Values of polyCauseCol that are natural.

Value

A list. counts is a one-row data.table: year, ignitions, escapes, areaBurnedHa (natural-cause polygons in the core), areaBurnedHaAllCause, nFires50 and nFires50AllCause (clipped pieces of at least escapeSizeHa). sizes is a data.table of the clipped pieces, with cause ("natural" or "allCause") and ha. polys is the SpatVector of the year's natural-cause polygons (not clipped; NULL without polygons or a cause column), to draw on a map.

hindcastSimulated	<i>One simulated year in an ELF core</i>
-------------------	--

Description

The simulated counterpart of [hindcastObserved\(\)](#), for one simulated year: the year's ignitions and escapes (ignitionsAndEscapes from fireSense_ignitionPredict) and each fire of the year's burn-ID raster (rstAnnualBurnID from fireSense_burn, one integer per fire), counted or clipped to the ELF core. A fire that burned only outside the core is not listed. The simulation burns the buffer too, so these sizes are smaller than burnSummary\$areaBurnedHa for fires crossing the core edge.

Usage

```
hindcastSimulated(burnID, coreRaster, ignitions = NULL, escapeSizeHa = 50)
```

Arguments

burnID	SpatRaster of fire IDs (NA where nothing burned).
coreRaster	SpatRaster of the ELF core; cells outside it are NA or 0. It need not share the grid of burnID: pixels are placed by their centres.
ignitions	data.table with the ignition cell numbers of burnID in pixelID and, in escaped, whether each ignition escaped. NULL gives NA ignitions and escapes.
escapeSizeHa	Size (ha) at which a fire counts in nFires50.

Value

A list. counts is a one-row data.table: ignitions, escapes, areaBurnedHa and nFires50, the columns of `hindcastObserved()`'s counts of the same names. sizes has one row per fire in the core: fire_id and ha.

ignitionFitFilenameFor

The shared IgnitionFit ledger files

Description

fireSense_IgnitionFit writes each fit (ignition and escape together) as a row of an rds file in a Google Drive folder (its ignitionFitGoogleDriveFolder); fireSense_ELFs reads them for a multi-ELF study area. With ignitionFitFilename = "latest" those modules use these functions instead of one named file. Parallel to the spread-fit ledger helpers (spreadFitFilenameFor(), latestSpreadFits()).

latestIgnitionFits() returns, for every polygon, its rows from the most recently modified ledger file that has it. Files are read newest first; with polygonIDs, reading stops once all of them are found. A file already in destinationPath with the same MD5 as on Drive is not downloaded again; one that differs is downloaded again with `reproducible::preProcess()`.

Usage

```
ignitionFitFilenameFor(fireYears)

ignitionFitFileTag

latestIgnitionFits(
  cloudFolderID = ignitionFitLedgerFolder(),
  destinationPath,
  polygonIDs = NULL
)
```

Arguments

- fireYears The fire years the fit used; only the first and last matter.
- cloudFolderID The Google Drive folder (url or id) holding the ledger files; default `ignitionFitLedgerFolder()`.
- destinationPath Local folder for the downloaded files.
- polygonIDs Optional. The polygons (ELF ids) wanted; NULL reads every file.

Details

`ignitionFitFileTag` marks the files whose fits use the current model: `xgboost` Poisson ignition and escape fits (`fireSense_IgnitionFit`'s `modelAlgorithm = "xgboost"` with objective `count:poisson`). The date is the model version. A fit made under a new model needs a new tag, so the old files drop out of "latest"; files with an older tag, e.g. `"_xgboost"` (Tweedie fits), are never read.

Value

`ignitionFitFilenameFor()`: the file name a fit over `fireYears` is written to, e.g. `"fireSenseIgnitionParams_1985-2020"`.
`latestIgnitionFits()`: the ledger rows, in the form `fireSense_IgnitionFit` writes them (a `data.frame` with a geometry column), or NULL when no file matches. Attribute `"ignitionFitFiles"` names the file each polygon came from.

igOrEscNames	<i>Construct standardized FireSense object names</i>
--------------	--

Description

Creates a standardized name by combining a pre prefix, a normalized `igOrEsc` token (case-transformed), and a post suffix.

Usage

```
igOrEscNames(  
  igOrEsc,  
  pre = "fireSense_",  
  post,  
  case = c("lower", "camel", "sentence", "title")  
)
```

Arguments

- igOrEsc character. A token such as "ignition" or "escape".
- pre character scalar. Prefix string (default "fireSense_").
- post character scalar. Suffix string to append.
- case One of `c("lower", "camel", "sentence", "title")`. Controls how `igOrEsc` is transformed before concatenation. Partial matching on the first letters is used: e.g., `"cam"`, `"sen"`, `"tit"`.

Value

character(1) constructed as `paste0(pre, transformed(igOrEsc), post)`.

Examples

```
igOrEscNames("ignition", post = "Covariates")
igOrEscNames("escape", pre = "fs_", post = "_Formula", case = "camel")
```

isLinearFuelRange	<i>Was this covariate rescaled as linear fuel biomass?</i>
-------------------	--

Description

Was this covariate rescaled as linear fuel biomass?

Usage

```
isLinearFuelRange(range)
```

Arguments

range Length-2 numeric: a covariate's entry in `covMinMax`.

Value

TRUE when range is [fuelLinearRange](#), i.e. the fit used linear fuel biomass, so a prediction must apply [fuelLogToLinear\(\)](#) to that covariate before rescaling it. FALSE for a fit made on the log scale, whose covariates must be left as they are.

isYoungAge	<i>Is an age young?</i>
------------	-------------------------

Description

The single rule for `youngAge`: an age is young when it is at or below `cutoffForYoungAge` (`age <= cutoffForYoungAge`). NA is never young. Every place in `fireSenseUtils` that decides young or not young calls this function, so fitting and prediction cannot disagree at the cutoff.

Usage

```
isYoungAge(age, cutoffForYoungAge = fireSenseYoungAgeCutoff)
```

Arguments

- age numeric vector of ages (stand age or time since disturbance), in years.
- cutoffForYoungAge age at and below which a pixel is young.

Value

logical vector the length of age; FALSE where age is NA.

latestNBACUrl	<i>Discover the URL of the most recent NBAC shapefile archive</i>
---------------	---

Description

Parses the Apache directory listing at base and returns the URL of the newest file matching NBAC_<startYear>to<endYear>, selected by the YYYYMMDD stamp embedded in the filename. If the listing cannot be read or no matching file is found, fallback is returned, so callers can rely on getting *some* URL back.

Usage

```
latestNBACUrl(  
  base = "https://cwfis.cfs.nrcan.gc.ca/downloads/nbac/",  
  fallback = paste0(base, "NBAC_1972to2025_20260513_shp.zip")  
)
```

Arguments

- base Character. Directory listing URL (must be the parent folder that serves the autoindex page).
- fallback Character. URL to return when discovery fails.

Value

A single character URL.

lccFlammableShare	<i>Each land-cover class's share of an ELF's flammable pixels</i>
-------------------	---

Description

How common each class is over the whole study area, not only where fires burned: a covariate for a class that covers a few percent of the ELF has too few pixels for its coefficient to be estimated. `fireSense_dataPrepFit` compares these shares with its `minCovariateProp` to decide which non-forest classes are clustered on their own (see [assessFuelClasses\(\)](#)) and whether treed wetland is a covariate (see [fireSenseCovariatesCreate\(\)](#)).

Usage

```
lccFlammableShare(rstLCC, flammableRTM = NULL)
```

Arguments

<code>rstLCC</code>	Land-cover <code>SpatRaster</code> (class codes as values).
<code>flammableRTM</code>	Optional <code>SpatRaster</code> on the same grid; only its cells equal to 1 count. Without it, every cell of <code>rstLCC</code> that is not NA or 0 counts.

Value

Named numeric, one element per class code (as character), summing to 1.

logistic4p	<i>Multiple-parameter versions of logistic functions</i>
------------	--

Description

Logistic functions using 2, 3, 4, or 5 parameters; `logisticAll` dispatches to one of them based on `length(logisticPars)`.

Usage

```
logistic4p(x, par)

logistic5p(x, par)

logistic3p(x, par, par1 = 0.1)

logistic3pUpper(x, par, par1 = 0.1)

logistic2p(x, par, par1 = 0.1, par4 = 0.5)

logisticAll(logisticPars, mat, covPars, lowerSpreadProb, link = NULL)
```

Arguments

<code>x</code>	Numeric vector. Linear predictor (e.g. <code>mat %*% covPars</code>) at which to evaluate the logistic.
<code>par</code>	Numeric vector of logistic parameters whose length matches the variant: 2 for <code>logistic2p</code> , 3 for <code>logistic3p</code> , etc. See logisticParamNames for the meaning of each position.
<code>par1</code>	Numeric scalar. Lower asymptote (replaces <code>par[1]</code> of the 4-parameter form).
<code>par4</code>	Numeric scalar. Asymmetry factor (replaces <code>par[4]</code> of the 4-parameter form).
<code>logisticPars</code>	Numeric vector of logistic parameters. Unless <code>link</code> names the form, its length selects the variant <code>logisticAll</code> dispatches to – except that a vector with an element named <code>upperTail1</code> is always <code>logistic3pUpper</code> , whatever its length.
<code>mat</code>	Numeric matrix of covariate values, one row per observation.
<code>covPars</code>	Numeric vector of covariate coefficients (same length as <code>ncol(mat)</code>); <code>mat %*% covPars</code> forms the linear predictor.
<code>lowerSpreadProb</code>	Numeric scalar in $[0, 1]$. The lower asymptote (<code>par1</code>) used for the 2- and 3-parameter forms.
<code>link</code>	NULL (the default) to choose the form from <code>logisticPars</code> as above, or <code>"logistic3pUpper"</code> to use the 3-parameter form with Stukel's upper tail. Pass it explicitly wherever <code>logisticPars</code> may arrive without names, as inside the objective function.

Details

The general form is the 5-parameter sigmoid (Richards' curve): $\text{par1} + (\text{par2} - \text{par1}) / (1 + (\exp(x) / \text{par3})^{(-\text{par4}))^{\text{par5}}}$. The 4-, 3-, and 2-parameter variants successively fix the lower asymptote (`par1`), the inflection-point scale (`par3 = 1`), and the asymmetry factor (`par5 = 0.5`) to reduce the parameter count.

`logistic3pUpper` is `logistic3p` with Stukel's (1988) generalized-logistic upper tail applied to the linear predictor: $\text{par1} + (\text{par}[1] - \text{par1}) / (1 + \exp(-H(\text{par}[2] * x)))^{\text{par}[3]}$, where H leaves negative values alone and, for $u = \text{par}[2] * x \geq 0$ and $a = \text{par}[4]$, is $(\exp(a * u) - 1) / a$ for $a > 0$, u for $a = 0$ and $-\log(1 - a * u) / a$ for $a < 0$ (Stukel, T. A. (1988), Generalized logistic models, JASA 83:426-431; as `sirt::pgenlogis()`). `par[4]` changes only how the curve approaches its upper asymptote: negative values slow the approach, so pixels with a high linear predictor are no longer all pressed against the ceiling; `par[4] = 0` is `logistic3p` exactly. In `logistic3p` the approach to the ceiling is set by the slope `par[2]` alone – `par[3]` shapes only the lower end – so this is the one parameter that changes the upper end independently.

Value

Numeric vector of logistic values; same length as `x` (or `nrow(mat)` for `logisticAll`).

logisticParamNames	<i>Logistic parameter names</i>
--------------------	---------------------------------

Description

Logistic parameter names

Usage

logisticParamNames

Value

A named list of length 4, with "2p", "3p", "4p", "5p" as names representing the 2-parameter etc. logistic curve.

logMinB	<i>Log with a minimum</i>
---------	---------------------------

Description

Used for transforming Biomass to the log scale

Usage

logMinB(x)

Arguments

x	Any value to be adjusted with log and a minimum B
---	---

Value

The original vector, logged with a minimum.

makeELFs

Make Ecologically-based Low Fractal-Dimensional polygons

Description

Fires have key features that are different than "normal" ecological zonations. These include: 1) the property that they spread horizontally across a landscape, regardless of human-mapped ecological areas. 2) Furthermore, even when homogeneous fire regions are mapped, the homogeneousness may be because of fuel changes that occupy the average condition for an ecological area, not necessarily the boundary conditions. 3) Small areas that encompass an area that is only "several times larger than the largest fires" will be too small to capture the potential for multiple very large fires. 4) Furthermore, very large ecological regions may encompass too much variation in fire-fuel-climate relationships that any estimates of fire behaviour will not be accurate across the whole region. 5) No matter what area is chosen to Thus, to estimate fire behaviour across a large region like Canada, we need polygons that are not too small ($\gg$ largest fires), not too big ($\ll$), are not constrained by ecological zones defined by vegetation, have buffers.

Usage

```
makeELFs(
  x,
  desiredBuffer = 20000,
  maxArea = 2.4e+11,
  destinationPath = ".",
  singleSpatVector = FALSE,
  useCache = TRUE
)

plotELFs(
  destinationPath = ".",
  which = NULL,
  fill = "green",
  plan = NULL,
  fewFires = NULL,
  fewFiresFill = "grey80",
  mergedFill = "lightblue",
  arcticFill = "#fdf2d0",
  ELFs = NULL,
  labels = c("code", "name", "none"),
  labelWhich = c("all", "highlighted"),
  buffers = TRUE,
  axes = c("m", "longlat", "none"),
  ...
)
```

Arguments

<code>x</code>	A polygon layer (SpatVector or sf) or gridded layer (SpatRaster) that maps the entire area of interest for the ELFs. If missing, it will try to use <code>scfmutils::prepInputsFireRegime</code> (= "FRU") if <code>scfmutils</code> is installed. Currently, only options with a column called FRU will work.
<code>desiredBuffer</code>	The distance in m that the buffers should be. Default is 20000.
<code>maxArea</code>	The area in m ² that will be split if the ecozones are larger than this. Default is 2.4e+11 which is about 500km x 500km plus the buffer. This means that the ELFs will be smaller than this, with lower limit being the individual ecozones plus buffer.
<code>destinationPath</code>	A path where any downloads should be put.
<code>singleSpatVector</code>	Logical. If TRUE, the function returns a single SpatVector of all ELFs (rather than the per-ELF list described in Value). Default FALSE.
<code>useCache</code>	Logical. If TRUE, then some internal steps will be cached. If FALSE, then some (or most) will not be cached.
<code>which</code>	Character. ELF names (e.g., <code>c("13.1", "4.1")</code>) whose cores <code>plotELFs()</code> fills with <code>fill</code> . Default NULL fills none.
<code>fill</code>	The colour used to fill the which ELFs.
<code>plan</code>	The merges and skips to show: <code>sim\$ELFmerges</code> of <code>fireSense_ELFs</code> , as returned by <code>ELFmergePlan()</code> or <code>fewFireELFs()</code> . <code>plotELFs()</code> always draws the state after the merges (<code>mergeELFs()</code>), with each merged ELF under its merged name (e.g. 11.1_2). NULL (default) computes the plan with <code>fewFireELFs()</code> at its default settings (Cached).
<code>fewFires</code>	ELF ids to fill as "not enough fires". NULL (default) uses the ELFs the plan skips (<code>ELFsSkipped()</code>); character(0) fills none.
<code>fewFiresFill</code>	The colour used to fill the fewFires ELFs.
<code>mergedFill, arcticFill</code>	The colours used to fill the merged ELFs (those the plan merges) and the Arctic ELFs (<code>ELFsArctic()</code> , which are not fitted).
<code>ELFs</code>	The list <code>makeELFs()</code> returns (before any merging), if the caller has it. NULL (default) builds it with <code>makeELFs()</code> (Cached). Without a poly element the outlines are drawn from <code>rasWhole</code> .
<code>labels</code>	What <code>plotELFs()</code> prints on each ELF: "code" (the ELF name, e.g. "13.1"), "name" (the name of the ecozone the ELF lies in, e.g. "Pacific Maritime"; see <code>elfLabels()</code>) or "none".
<code>labelWhich</code>	"all" labels every ELF; "highlighted" labels only those in which.
<code>buffers</code>	If FALSE, <code>plotELFs()</code> draws only the ELF cores, without the buffer rings.
<code>axes</code>	"m" shows the map's own axes (metres); "longlat" keeps the map projection and draws a graticule labelled in degrees; "none" draws no axes and no box.
<code>...</code>	Passed to <code>fewFireELFs()</code> when plan is NULL, e.g. <code>pixelAreaHa</code> (default 5.76) or <code>fireYears</code> (default <code>defaultFireYears()</code>).

Details

With these five points in mind, we define *Ecologically-based Low Fractal-dimensional polygons, "ELFs", that are constrained by large ecological zones (Canadian Ecozones), are "larg-ish", more "blobby" (low fractal dimension) than other ecological zonation in mountains (which tend to follow elevation contours), and are each buffered (default 20km).

Value

A list of length 2, with `rasWhole` and `rasCentered`. Each of these will have the complete list of ELFs. `rasWhole` is the whole map of Canada with the individual ELF highlighted, and `rasCentered` is *just* the individual ELF centred and reprojected so its projected parameters are: `long0` is the x mid, and `lat0` = `ymin` and `lat1` = `ymax` of the individual ELF. This will create ELFs with the least amount of pixel deformation.

See Also

[fireSenseCloudParameters\(\)](#)

makeFireIDs

Identify each year's individual fires and buffer them accordingly

Description

Identify each year's individual fires and buffer them accordingly

Usage

```
makeFireIDs(
  year,
  fireRaster,
  flammableRTM,
  bufferForFireRaster,
  areaMultiplier,
  minSize = 5000,
  verb = 1
)
```

Arguments

<code>year</code>	numeric fire year
<code>fireRaster</code>	a <code>SpatRaster</code> with values representing fire years
<code>flammableRTM</code>	a <code>rasterToMatch</code> with binary values where 1 represents flammable pixels, 0 nonflammable, and NA no data
<code>bufferForFireRaster</code>	buffer size used to group discrete patches of burned pixels as belonging to the same fire

areaMultiplier	A scalar that will buffer <code>areaMultiplier * fireSize</code>
minSize	The absolute minimum size of the buffer & non-buffer together. This will be imposed after <code>areaMultiplier</code> .
verb	Logical or numeric related to how much verbosity is printed. FALSE or 0 is none. TRUE or 1 is some. 2 is much more.

Value

a data.table with fire ID, buffer status, and pixelID

makeFireSenseLCC	<i>Prepare NTEMS Land Cover Data for Fire Simulation Models</i>
------------------	---

Description

Processes NTEMS Land Cover Classification (LCC) data for a specific year, adjusting non-flammable classes based on a flammability threshold during reprojection to a target raster's resolution and extent.

Usage

```
makeFireSenseLCC(
  neededYear,
  to,
  maskTo = NULL,
  nonflammableLCC = fireSenseNonflammableLCC,
  flammabilityThreshold = fireSenseFlammabilityThreshold,
  writeTo = NULL,
  overwrite = TRUE,
  destinationPath,
  lccSource = getOption("fireSense.lccSource", "SCANFI"),
  scanfiVersion = fireSenseSCANFIVersion
)
```

Arguments

neededYear	Numeric. The specific year required for the NTEMS Land Cover Classification (LCC) data.
to	A <code>SpatRaster</code> (or raster-like) template defining the resolution, extent, and CRS that the output LCC raster will be aligned to. Forwarded to <code>reproducible::postProcessTo(to = ...)</code> .
maskTo	Optional. A <code>SpatVector</code> / <code>SpatRaster</code> / polygon used to mask the output. Forwarded to <code>reproducible::postProcessTo(maskTo = ...)</code> . Default NULL (no extra mask beyond to).

nonflammableLCC	Numeric vector. LCC codes representing non-flammable land cover types (e.g., water, rock, urban). Defaults to fireSenseNonflammableLCC . Pixels with these values are initially masked or converted for thresholding.
flammabilityThreshold	Numeric. A value between 0 and 1. Target pixels where the proportion of underlying <i>flammable</i> source pixels is below this threshold will be reclassified to 0 (non-flammable) in the output raster. Defaults to 0.1.
writeTo	Character string or NULL. Optional filename for the output LCC raster, relative to destinationPath. If NULL (default), the raster is not written to disk by this function.
overwrite	Logical. Overwrite NTEMS FAO and flammable pixels rasters.
destinationPath	Character string. Directory path where source NTEMS data will be downloaded/loaded (via <code>prepInputs_NTEMS_LCC_FAO</code>) and where the output raster will be written if <code>writeTo</code> is specified.
lccSource	"SCANFI" (default) or "NTEMS"; the default can be set with <code>options(fireSense.lccSource =)</code> . SCANFI land cover (<code>LandR::prepInputs_SCANFI_LCC_FAO()</code>) has no wetland classes, so they are added from the Canadian Wetland Inventory Map (<code>LandR::prepInputs_CWIM()</code>) with <code>LandR::wetlandToLCC()</code> : wet treed pixels become 81, other wet flammable pixels 80, as in NTEMS. This matches Biomass_borealDataPrep's default land cover. "NTEMS" uses <code>LandR::prepInputs_NTEMS_LCC_FAO()</code> , which has 80 and 81 already, exactly as before.
scanfiVersion	Character. Only used when <code>lccSource = "SCANFI"</code> ; forwarded to <code>LandR::prepInputs_SCANFI_LCC_FAO</code> dataVersion. Default "V3" (annual, 1985-2025). The installed LandR is checked for dataVersion = "V3" support before it is called; an older LandR stops with a message asking for one that has it, rather than silently building V2 land cover under a "V3" request.

Details

This function performs several steps:

1. Downloads and prepares NTEMS LCC data for the specified neededYear using `LandR::prepInputs_NTEMS_LCC_FAO`, cropping and masking to the target extent.
2. Identifies the dominant *flammable* land cover class within each pixel of to. This is done by temporarily masking non-flammable classes (defined in `nonflammableLCC`) to NA in the source raster and then reprojecting using the 'mode' method.
3. Calculates the proportion of *flammable* cover within each pixel of to by reprojecting a binary (1=flammable, 0=non-flammable) version of the source raster using the 'average' method.
4. Creates the final raster: Pixels where the proportion of flammable cover (calculated in step 3) is below the `flammabilityThreshold` are assigned a value of 0 (representing non-flammable). Otherwise, pixels retain the dominant flammable LCC code identified in step 2.

Value

A list of two `SpatRaster` objects:

1. the processed land cover classification at the resolution, extent, and CRS of `to` with values representing either the modal flammable LCC code or 0 if the pixel is below the `flammabilityThreshold`, and
2. the proportion of each pixel that is flammable in the native resolution

Examples

```
## Not run:
# Requires terra and potentially LandR installed
# Need example to and maskTo objects
library(terra)

# Create dummy to and maskTo
ras <- rast(xmin = 0, xmax = 10, ymin = 0, ymax = 10, res = 1)
values(ras) <- 1
crs(ras) <- "EPSG:4326" # Example CRS
sa <- ext(ras) |> as.polygons() |> vect()
crs(sa) <- crs(ras)

# Define a destination path (replace with a real path)
destPath <- tempdir()

# Run the function (will likely try to download data)
# Note: prepInputs_NTEMS_LCC_FAO might require specific setup or data sources
# This example might fail if data download/access isn't configured
# fireLCC <- makeFireSenseLCC(neededYear = 2011, # Example year
#                               to = ras,
#                               maskTo = sa,
#                               destinationPath = destPath)
# plot(fireLCC)

## End(Not run)
```

makeFireSenseLCCDeps *Functions whose code affects what `makeFireSenseLCC()` returns*

Description

`reproducible::Cache()` digests only the called function's own code, so a cached `makeFireSenseLCC()` call keeps returning its old result when a function it *calls* changes. Pass this list as that call's `.cacheExtra` and those functions become part of the key.

Usage

```
makeFireSenseLCCDeps(lccSource = getOption("fireSense.lccSource", "SCANFI"))
```

Arguments

lccSource "SCANFI" (default) or "NTEMS"; the default can be set with `options(fireSense.lccSource = )`. SCANFI land cover (`LandR::prepInputs_SCANFI_LCC_FA0()`) has no wetland classes, so they are added from the Canadian Wetland Inventory Map (`LandR::prepInputs_CWIM()`) with `LandR::wetlandToLCC()`: wet treed pixels become 81, other wet flammable pixels 80, as in NTEMS. This matches `Biomass_borealDataPrep`'s default land cover. "NTEMS" uses `LandR::prepInputs_NTEMS_LCC_FA0()`, which has 80 and 81 already, exactly as before.

Details

Ask for the list rather than writing one out: which functions are in the path depends on `lccSource`, which is a run-time option, so a hard-coded list is wrong for the other source and goes stale when the default moves. It already did – `fireSense_dataPrepFit` pinned `LandR::prepInputs_NTEMS_LCC_FA0()` and kept it after the default became SCANFI, leaving every SCANFI and CWIM change invisible to the cache.

Value

A list of functions, to pass as `reproducible::Cache()`'s `.cacheExtra`.

See Also

`makeFireSenseLCC()`

Examples

```
## Not run:
Cache(makeFireSenseLCC(neededYear = 2020, to = rtm, destinationPath = dPath),
      .cacheExtra = makeFireSenseLCCDeps())

## End(Not run)
```

makeGLM	<i>Wrapper on glm that creates separate GLMs for each item in fuel</i>
---------	--

Description

Wrapper on `glm` that creates separate GLMs for each item in `fuel`

Usage

```
makeGLM(fuel, landscape, form)
```

Arguments

fuel	vector of tree species or LCC in landscape by which to subset
landscape	created by <code>fuelClassPrep</code>
form	the formula to use - as a character

makeLandcoverDT	<i>Create landcoverDT object to classify and track non-forest land cover classes</i>
-----------------	--

Description

Create landcoverDT object to classify and track non-forest land cover classes

Usage

```
makeLandcoverDT(rstLCC, flammableRTM, forestedLCC, nonForestedLCCGroups)
```

Arguments

rstLCC	landcover raster
flammableRTM	a rasterToMatch with binary values where 1 represents flammable pixels, 0 nonflammable, and NA no data
forestedLCC	vector of values representing forested landcover classes in rstLCC
nonForestedLCCGroups	a named list of non-forested flammable landcover groups

Value

a data.table with columns for pixelID and binary presence of landcover

makeLocilist	<i>Convert a list of SpatialPointsDataFrame object to a list of data.table objects</i>
--------------	--

Description

Must supply a raster so that points can be converted to the cells on a raster. It is assumed that the sizeCol is accurate. If not, it should be recalculated before this function call.

Usage

```
makeLocilist(
  ras,
  pts,
  idsCol = "FIRE_ID",
  dateCol = "YEAR",
  sizeCol = "POLY_HA",
  sizeColUnits = "ha",
  yearPrefix = "year"
)
```

Arguments

ras	A raster that will be the template for cells (pixel ids)
pts	A list of sf point objects
idsCol	Character string identifying column name in pts that has unique id per event (i.e., fire)
dateCol	Character string identifying column name in pts that has year
sizeCol	Character string identifying column name in pts that has size of individual event. Can be in hectares or metres squared. Should set sizeColUnits
sizeColUnits	Character string. Either "ha" or "m2".
yearPrefix	Character string prefixed to each year when naming the output list elements (e.g. "year" produces "year2001", "year2002", ...). Default "year".

Value

A list of data.table objects, each with 4 columns, "size" (in pixels), "date", "ids" from idsCol, and "cells", which are the pixel indices of the pts points.

makeMutuallyExclusive *guarantees mutually exclusive values in a data table*

Description

guarantees mutually exclusive values in a data table

Usage

```
makeMutuallyExclusive(dt, mutuallyExclusiveCols = list(youngAge = c("vegPC")))
```

Arguments

dt	a data.table with columns that should be mutually exclusive
mutuallyExclusiveCols	A named list, where the name of the list element must be a single covariate column name in dt. The list content should be a "grep" pattern with which to match column names, e.g., "vegPC". The values of all column names that match the grep value will be set to 0, whenever the name of that list element is non-zero. Default is list("youngAge" = list("vegPC")), meaning that all columns with vegPC in their name will be set to zero wherever youngAge is non-zero. Patterns are matched anchored to the start of the column name, and the key column itself (cov1) is never zeroed, even if it matches its own pattern.

Value

a data.table with relevant columns made mutually exclusive

makeRastersFromCD	<i>Put cohortData back into a SpatRaster with some extra details</i>
-------------------	--

Description

Put cohortData back into a SpatRaster with some extra details

Usage

```
makeRastersFromCD(class, cohortData, flammableRTM, pixelGroupMap)
```

Arguments

class	fuelClass from sppEquiv
cohortData	A data.table with columns: pixelGroup, ecoregionGroup, speciesCode, and optionally age, B, mortality, aNPPAct, and sumB.
flammableRTM	a rasterToMatch with binary values where 1 represents flammable pixels, 0 nonflammable, and NA no data
pixelGroupMap	A RasterLayer with pixel values equal to a pixel group number that corresponds exactly to pixelGroup column in cohortData.

Value

a SpatRaster with values equal to class biomass (B)

makeTSD	<i>Prepare a time since disturbance map from stand age and fire data</i>
---------	--

Description

Combines an initial stand age map with disturbance (fire) history to produce a time-since-disturbance (TSD) raster. Stand age is trusted for most pixels, but for pixels whose stand age is unreliable (e.g. non-forest land cover) the fire history is used instead: a recorded recent burn sets the TSD, while the absence of one marks the pixel as old (cutoffForYoungAge + 1).

Usage

```
makeTSD(
  year,
  firePolys = NULL,
  fireRaster = NULL,
  standAgeMap,
  lcc = NULL,
  cutoffForYoungAge = fireSenseYoungAgeCutoff,
  pixToUpdate = NULL,
  flammablePixels = NULL
)
```

Arguments

year	the year represented by standAge.
firePolys	list of spatialPolygon objects comprising annual fires. fireRaster will supersede firePolys if provided.
fireRaster	a RasterLayer with values representing fire years.
standAgeMap	initial stand age map
lcc	Optional data.table with landcover values, i.e., landcoverDT (see makeLandcoverDT()). When supplied, pixToUpdate and flammablePixels are derived from it unless they are passed explicitly. Specific to fireSense_dataPrepFit; for general use prefer pixToUpdate / flammablePixels.
cutoffForYoungAge	Numeric. Default is 15. This is the age below which the pixel is considered "young" -> youngAge column will be 1 if age <= 15
pixToUpdate	Optional integer vector of pixelIDs whose age should be taken from fire history rather than standAgeMap (e.g. pixels with no reliable stand age). If NULL and lcc is supplied, these are the non-forest pixels in lcc (those with a positive sum across the non-forest landcover columns).
flammablePixels	Optional integer vector of flammable pixelIDs. Pixels not in this set are set to NA in the output (non-flammable). If NULL and lcc is supplied, this is lcc\$pixelID. If NULL and lcc is not supplied, no pixels are masked to NA.

Details

Which pixels to update from fire history, and which pixels are flammable, can be supplied in one of two ways:

- directly, via pixToUpdate and flammablePixels (general purpose); or
- via lcc, the landcoverDT produced by fireSense_dataPrepFit, from which both are derived (non-forest pixels are updated, and lcc\$pixelID defines the flammable mask). This is retained for backwards compatibility.

Value

a SpatRaster with values representing time since disturbance

mergeELFs

Merge ELF maps as planned

Description

Replaces the members of each merge in [ELFmergePlan\(\)](#) by one ELF. Its rasWhole layer is the cell-by-cell maximum of the members' layers, so a cell that is core in any member is core. A buffer drawn around the union of cores equals the union of the members' buffers, so the buffer does not need redrawing. Its rasCentered layer is that map projected onto a Lambert conformal conic centred on the merged core, as [makeELFs\(\)](#) centres each ELF. Its poly rows are rebuilt from the merged layer the way [makeELFs\(\)](#) builds them.

Usage

```
mergeELFs(ELFs, plan)
```

Arguments

ELFs	list with <code>rasWhole</code> and <code>rasCentered</code> (named lists of <code>SpatRaster</code>) and optionally <code>poly</code> , as returned by <code>makeELFs()</code> .
plan	<code>data.table</code> from <code>ELFmergePlan()</code> .

Value

ELFs, with each merge's members replaced by the merged ELF.

<code>mergePreparedCovs</code>	<i>Build ignition covariates table by stacking/extracting rasters and joining lightning</i>
--------------------------------	---

Description

For each year (or year period) provided, stacks and extracts covariates from coarse fuel rasters (split into forested vs. non-forested LCC groups) and climate, at ignition point locations, then merges results across years, filters unusable pixels, appends lightning summaries, and orders columns. Results are optionally cached.

Usage

```
mergePreparedCovs(
  years,
  fuelCovsCoarse,
  ignitionFirePoints,
  nonForestedLCCGroups,
  ignitionClimateCoarse,
  lightningMap,
  digest,
  useCache = TRUE
)
```

Arguments

<code>years</code>	A vector or list of year identifiers passed to <code>stackAndExtract()</code> . Can be a list matching the structure of <code>fuelCovsCoarse/ignitionClimateCoarse</code> .
<code>fuelCovsCoarse</code>	A list (possibly nested by year) of named raster-layer collections (e.g., <code>terra::SpatRaster</code> or lists of <code>SpatRaster</code>) representing fuel-related covariates. Names must overlap with LCC group names used to split forested vs. non-forested classes.
<code>ignitionFirePoints</code>	Ignition point locations (e.g., <code>sf/sp</code> points or a <code>data.frame</code> convertible by <code>stackAndExtract()</code> used for extraction.

nonForestedLCCGroups	A named vector or list whose names identify the LCC (land-cover) classes considered non-forested. These names are used to split fuelCovsCoarse into LCC (non-forested) and fuel (forested/other) inputs.
ignitionClimateCoarse	A list (possibly by year) of coarse climate rasters (e.g., <code>terra::SpatRaster</code>) to include in the stack-and-extract process. The names of this list are used as <code>userTags</code> in caching and are later treated as climate column names when computing row-wise cover sums.
lightningMap	A named <code>terra::SpatRaster</code> (or a named list coercible to one) containing one or more lightning-derived layers (e.g., "lightningDays"). These values are looked up by <code>pixelID</code> and appended as columns.
digest	A cache key supplement (character or list) forwarded to <code>reproducible::Cache()</code> via <code>.cacheExtra</code> to ensure cache correctness for large inputs omitted from the hash.
useCache	logical (default TRUE). If TRUE, wrap the Map call in <code>reproducible::Cache()</code> with <code>omitArgs = c("fuel", "LCC")</code> , <code>.cacheExtra = digest</code> , and <code>userTags = names(ignitionClimateCoarse)</code> .

Details

Processing steps:

1. Calls `stackAndExtract()` via `Map()` for each entry in years, splitting `fuelCovsCoarse` per year into:
 - fuel: all layers not in `names(nonForestedLCCGroups)`
 - LCC: layers whose names are in `names(nonForestedLCCGroups)`
 Additional `MoreArgs` include `climate = ignitionClimateCoarse` and `fires = ignitionFirePoints`.
2. Optionally caches the list of per-year extractions with `reproducible::Cache()` (function name tagged as "stackAndExtract").
3. Combines the list to a single `data.table` via `data.table::rbindlist()`.
4. Removes rows for which the sum across all cover-type columns is zero (computed as `rowSums(.SD)` over all columns except ignitions, `yearTxt` (from `fireSenseUtils`), `cell`, and the climate variable names).
5. Renames the `cell` column to `pixelID`, coerces `year` to numeric, and appends lightning columns by indexing `terra::values(terra::rast(lightningMap))` with `pixelID`.
6. Optionally reorders columns to place `pixelID`, ignitions, climate columns (i.e., `names(ignitionClimateCoarse)`), and `youngAgeTxt` (if it exists) first.

Assumptions / Requirements:

- `fireSenseUtils::stackAndExtract()` must accept arguments `years`, `fuel`, `LCC`, `climate`, and `fires`, and return a `data.table`-like object with at least `cell`, `ignitions`, and `year` (or `yearTxt`) columns.
- `fireSenseUtils::yearTxt` is used as a column name to exclude from cover sums; ensure it exists/aligns with the data produced by `stackAndExtract()`.

- `youngAgeTxt` is referenced when establishing column order but is **not** defined in this scope; if unavailable, it is silently omitted.
- `pixelID` is assumed to be a valid row index into `lightningMap` raster values.

Lightning layer choice: The code appends all layers provided in `lightningMap`. If a single best-performing layer is known (e.g., `"lightningDays"`), subset `lightningMap` before calling this function.

Caching: Large raster inputs are excluded from the cache key via `omitArgs = c("fuel", "LCC")`; the caller-supplied digest should capture the salient identity of those omitted objects to avoid cache collisions.

Value

A data.table of ignition covariates with:

- `pixelID` (formerly `cell`)
- `ignitions`
- Climate covariate columns (names derived from `ignitionClimateCoarse`)
- Cover/class covariates retained from `stackAndExtract()` (filtered to rows with non-zero cover sum)
- Lightning-derived columns appended from `lightningMap`
- `year` coerced to numeric

Potential pitfalls

- If `pixelID` indexes are not aligned with `lightningMap` cell indices, appended lightning values will be incorrect; ensure consistent indexing/resolution.
- If `fireSenseUtils::yearTxt` is absent in the returned data, the exclusion in `.SDcols` is harmless, but confirm that the intended year column exists for analysis.
- If `youngAgeTxt` is not defined, it will be ignored when reordering columns.

See Also

`fireSenseUtils::stackAndExtract`, `reproducible::Cache`, `data.table::rbindlist`, `terra::rast`, `terra::values`

Examples

```
## Not run:
# Pseudocode, assuming you have compatible inputs prepared:
yrs <- list(`2001` = 2001L, `2002` = 2002L)
# fuelCovsCoarse[[year]] is a named list or SpatRaster with LCC class names
# ignitionClimateCoarse[[year]] is a SpatRaster of climate layers
# ignitionFirePoints are point locations (sf/sp/data.frame accepted by stackAndExtract)
# lightningMap is a named SpatRaster (e.g., "lightningDays")

out <- mergePreparedCovs(
  years = yrs,
  fuelCovsCoarse = fuelCovsCoarse,
```

```
ignitionFirePoints = ignitionFirePoints,
nonForestedLCCGroups = c("Shrub", "Grass", "Bare"),
ignitionClimateCoarse = ignitionClimateCoarse,
lightningMap = lightningMap["lightningDays"],
digest = list(seed = 1L, cfg = "v1"),
useCache = TRUE
)
data.table::str(out)

## End(Not run)
```

multiplier	<i>Buffer-size multiplier for fire footprints</i>
------------	---

Description

Returns a per-fire buffer target (in pixels) by scaling size so that small fires get proportionally larger buffers and large fires plateau at baseMultiplier * size. The result is then clamped from below by minSize. Used when generating non-burned "control" pixels around historical fire perimeters.

Usage

```
multiplier(size, minSize = 1000, baseMultiplier = 5)
```

Arguments

- size Numeric vector. Fire sizes in pixels.
- minSize Numeric scalar. Absolute floor on the returned size; the buffer (plus burned pixels) is never smaller than this. Default 1000.
- baseMultiplier Numeric scalar. Asymptotic multiplier for large fires (as size grows, the effective multiplier tends to baseMultiplier). Default 5.

Value

Integer vector the same length as size, giving the target buffered area in pixels for each fire.

mustFit	<i>Must this run fit, given doFit and the ledger?</i>
---------	---

Description

Must this run fit, given doFit and the ledger?

Usage

```
mustFit(doFit, hasLedgerFit, polygon = "")
```

Arguments

doFit	NA, TRUE or FALSE, as returned by resolveDoFit() .
hasLedgerFit	logical: does the ledger hold a usable fit for this polygon? Evaluated lazily, only when doFit is not TRUE, so a forced fit never consults the ledger.
polygon	the polygon identifier, for the error message.

Value

TRUE to fit, FALSE to use the ledger's fit. Stops when doFit is FALSE and the ledger has no usable fit.

nfdbPointUrl	<i>URL of the National Fire Database point archive</i>
--------------	--

Description

The archive of all fire points, as a shapefile, on the CFS server. The file was NFDB_point.zip; CFS renamed it NFDB_point_shp.zip (with _txt and large_fires variants beside it), and the old name returns HTTP 404. The re-download check in [getFirePoints_NFDB\(\)](#) and [getFirePoints_NFDB_V2\(\)](#) then could not fetch a newer release, so the last copy on disk (fires to 2024) kept being used.

Usage

```
nfdbPointUrl()
```

Value

The URL, a character string.

omitNullArgs	<i>Names of the arguments that are NULL, to give to Cache(omitArgs =)</i>
--------------	---

Description

An argument added to a cached call changes its key even when it is NULL. Omitting it while it is NULL keeps the key of a call made before the argument existed, so the cached result is still found.

Usage

```
omitNullArgs(...)
```

Arguments

... Named arguments.

Value

A character vector: the names of those that are NULL.

paramsSeparate	<i>Split the character vector of parameters into covPars and logisticPars</i>
----------------	---

Description

[DEoptim::DEoptim](#) does not differentiate between the logistic parameters and the covariates. This splits the vector into the correct components. The split is based on the number of covariates. Therefore the number of logistic parameters is deduced from `length(pars) - parsModel`.

Usage

```
paramsSeparate(par, parsModel)
```

Arguments

par	Numeric vector of all parameters. The covariate parameters must be the second group.
parsModel	Integer. The number of covariates.

Value

list of 2 numeric vectors covPars and logisticPars, representing the parameters for the covariates and the logistic equation, respectively.

plotBurnSummary	<i>Plot burn summary</i>
-----------------	--------------------------

Description

Create plot with subplots showing: a) area burned; b) number of fires; c) mean fire size.

Usage

```
plotBurnSummary(
  climateScenario,
  studyAreaName,
  outputDir,
  Nreps,
  years,
  pixelSize,
  simFiles
)
```

Arguments

climateScenario	character string specifying the name of a CIMP6 climate scenario, including SSP, formatted as in ClimateNA, using underscores as separator (e.g., 'CanESM5_SSP370').
studyAreaName	character string giving the study area name
outputDir	Path specifying the directory to which outputs figures/objects should be saved.
Nreps	the number of simulation replicates/run used to produce summary figures. NOTE: mclapply is used internally, so you should set options(mc.cores = nReps) to take advantage of parallel processing.
years	integer (length 2). start and end years for comparison.
pixelSize	raster pixel (Cell) size
simFiles	Optional character vector of simulation output file paths (e.g. the file column of SpaDES::core::outputs(sim)), used to locate the per-replicate files instead of reconstructing them from outputDir. Needed when the replicate outputs do not live in <outputDir>/repNN/, e.g. when summarizing runs restored from another machine, or when the replicate directories are unpadded (rep1 rather than rep01). When supplied, the replicates present in simFiles are used, and Nreps is ignored.

Value

list of file names corresponding to the figures and/or objects written to disk

plotCumulativeBurns *Plot cumulative burn maps*

Description

Plot cumulative burn maps

Usage

```
plotCumulativeBurns(
  climateScenario,
  studyAreaName,
  outputDir,
  Nreps,
  years,
  rasterToMatch,
  simFiles
)
```

Arguments

climateScenario	character string specifying the name of a CIMP6 climate scenario, including SSP, formatted as in ClimateNA, using underscores as separator (e.g., 'CanESM5_SSP370').
studyAreaName	character string giving the study area name
outputDir	Path specifying the directory to which outputs figures/objects should be saved.
Nreps	the number of simulation replicates/run used to produce summary figures. NOTE: mclapply is used internally, so you should set options(mc.cores = nReps) to take advantage of parallel processing.
years	integer (length 2). start and end years for comparison.
rasterToMatch	A RasterLayer objects to use as the template for all subsequent raster operations (i.e., the one used throughout the simulation).
simFiles	Optional character vector of simulation output file paths (e.g. the file column of SpaDES.core::outputs(sim)), used to locate the per-replicate files instead of reconstructing them from outputDir. Needed when the replicate outputs do not live in <outputDir>/repNN/, e.g. when summarizing runs restored from another machine, or when the replicate directories are unpadded (rep1 rather than rep01). When supplied, the replicates present in simFiles are used, and Nreps is ignored.

Value

list of file names corresponding to the figures and/or objects written to disk
 a file path corresponding to the images and/or objects written to disk

plotHindcastDistributions

Simulated rep-years beside the observed year

Description

One panel each for the number of ignitions, escapes, area burned in the core and fires of at least the escape size: the histogram of the simulated rep-years with the observed value as a vertical line. Area burned and fire count have two lines, the natural-cause and the all-cause observed value.

Usage

```
plotHindcastDistributions(reps, observed, title = NULL)
```

Arguments

reps	data.table with one row per simulated rep-year and columns ignitions, escapes, areaBurnedHa and nFires50 (the rows fireSense_summary writes to hindcast_reps.csv, pooled over parameter sets).
observed	The result of hindcastObserved() .
title	Plot title, e.g. the ELF and year.

Value

A ggplot.

plotHindcastMap

Burn-frequency map with the observed fires on top

Description

The cumulative burn map of the simulated rep-years divided by their number (the fraction of rep-years each pixel burned), with the outlines of the observed fire polygons drawn over it.

Usage

```
plotHindcastMap(
  burnMap,
  nReps,
  firePolys = NULL,
  coreRaster = NULL,
  title = NULL
)
```

Arguments

burnMap	SpatRaster count of rep-years in which each pixel burned.
nReps	Number of rep-years the count is over.
firePolys	SpatVector of the observed polygons to draw, or NULL.
coreRaster	Optional SpatRaster of the ELF core: the map is masked to it.
title	Plot title.

Value

A ggplot.

plotHindcastSizes	<i>Fire-size distribution, simulated and observed</i>
-------------------	---

Description

The empirical cumulative distribution of core-clipped fire sizes (at least minHa), pooled over the simulated rep-years, with the observed natural-cause and all-cause fire sizes as step lines.

Usage

```
plotHindcastSizes(simSizes, observed, minHa = 50, title = NULL)
```

Arguments

simSizes	Numeric vector of simulated fire sizes (ha) pooled over rep-years (the ha column of hindcast_sizes.csv).
observed	The result of hindcastObserved() .
minHa	Sizes below this are left out of both sides (default: 50 ha, the escape size).
title	Plot title.

Value

A ggplot.

plotHistoricFires	<i>Plot historic ignitions, escapes, and area burned</i>
-------------------	--

Description

Plot historic ignitions, escapes, and area burned

Usage

```
plotHistoricFires(
  climateScenario,
  studyAreaName,
  outputDir,
  pixelSize,
  firePolys,
  ignitionPoints,
  simFiles
)
```

Arguments

climateScenario	character string specifying the name of a CIMP6 climate scenario, including SSP, formatted as in ClimateNA, using underscores as separator (e.g., 'CanESM5_SSP370').
studyAreaName	character string giving the study area name
outputDir	Path specifying the directory to which outputs figures/objects should be saved.
pixelSize	raster pixel (Cell) size
firePolys	A sf spatial polygons of historic fire burn areas, from the Canadian National Fire Database.
ignitionPoints	A sf spatial points of historic fire ignitions, from the Canadian National Fire Database.
simFiles	Optional character vector of simulation output file paths (e.g. the file column of <code>SpaDES.core::outputs(sim)</code>), used to locate the per-replicate files instead of reconstructing them from outputDir. Needed when the replicate outputs do not live in <code><outputDir>/repNN/</code> , e.g. when summarizing runs restored from another machine, or when the replicate directories are unpadded (rep1 rather than rep01). When supplied, the replicates present in simFiles are used, and Nreps is ignored.

Value

list of file names corresponding to the figures and/or objects written to disk

predictIgnition	<i>Predictions from ignition model</i>
-----------------	--

Description

Predictions from ignition model

Usage

```
predictIgnition(model, data, rescaleFactor, lambdaRescaleFactor)
```

Arguments

model	the model used to fit ignition data (must have a predict method)
data	data for prediction
rescaleFactor	spatial rescaling factor when predicted and fitted data are at different scales. Calculated as: $(\text{predResolution}/\text{fitResolution})^2$
lambdaRescaleFactor	If the data for fitting has been sampled for pseudo-absences, this imposes a new baseline probability of fire occurrences, hence predictions need to be adjusted. If the original fire prob. is $(\text{total no. fires})/(\text{total no. fires} + \text{total no. absences})$, and the fire probability imposed by sampling is $(\text{total no. fires})/(\text{total no. fires} + \text{no. sampled pseudo-absences})$, to adjust predicted values, one needs to multiply them by $(\text{total no. fires} + \text{no. sampled pseudo-absences})/(\text{total no. fires} + \text{total no. absences})$.

Value

vector of predicted values.

prepare_FuelCovsCoarse	<i>Create and aggregate fuel covariate rasters to a coarse resolution</i>
------------------------	---

Description

Creates fuel covariate layers from tabular inputs using `fireSenseUtils::fireSenseCovariatesCreate()`, converts the resulting data frame into a raster using `SpaDES.tools::rastFromDF()`, and aggregates the raster to a coarser spatial resolution with `terra::aggregate()` using the mean.

Usage

```
prepare_FuelCovsCoarse(..., rasTemplate, fact)
```

Arguments

<code>...</code>	Arguments forwarded directly to <code>fireSenseUtils:::fireSenseCovariatesCreate()</code> , defining the inputs and options required to construct fuel covariates.
<code>rasTemplate</code>	A raster-like object (typically <code>terra::SpatRaster</code>) used as a spatial template (extent, resolution, projection) when converting the fuel covariate data frame to a raster.
<code>fact</code>	<code>numeric(1)</code> aggregation factor passed to <code>terra::aggregate(fact = ...)</code> , controlling the spatial coarsening scale.

Details

The function performs the following steps:

1. Generates a fuel covariate data frame using `fireSenseUtils:::fireSenseCovariatesCreate(...)`.
2. Converts the data frame to a raster aligned with `rasTemplate` via `SpaDES.tools::rastFromDF()`.
3. Aggregates the raster to a coarser resolution using `terra::aggregate(fact = fact, fun = mean)`.

The aggregation is applied across all raster layers (if present), computing the mean of contributing cells.

Value

A `terra::SpatRaster` containing aggregated fuel covariate layers at the coarser spatial resolution defined by `fact`.

Assumptions and required environment

The function assumes that the following are available:

- `fireSenseUtils:::fireSenseCovariatesCreate()` (non-exported API),
- `SpaDES.tools::rastFromDF()`,
- the `terra` package.

No input validation is performed on the structure of the data returned by `fireSenseCovariatesCreate()`.

Note

This function uses a non-exported function from `fireSenseUtils` via `:::`, which may change without notice across package versions.

References

- `terra` package reference manual: raster aggregation with `aggregate()`.
- `SpaDES.tools` package reference manual: `rastFromDF()`.
- `fireSenseUtils` package source code: `fireSenseCovariatesCreate()` (internal API).

See Also

[terra::aggregate\(\)](#), [SpaDES.tools::rastFromDF\(\)](#)

Examples

```
## Not run:
fuelCovsCoarse <- prepare_FuelCovsCoarse(
  fuels = fuelTable,
  rasTemplate = templateRaster,
  fact = 4
)

## End(Not run)
```

```
prepare_FuelCovsCoarseByYear
```

Fuel covariate rasters at a coarse resolution, with youngAge resolved for each fire year

Description

As [prepare_FuelCovsCoarse\(\)](#), but the covariates are built once with [fireSenseCovariatesCreate\(youngAge = FALSE\)](#) and then, for each of years, [youngAge](#) is set from [youngAgeAtYear\(\)](#) over every pixel (time since disturbance of the data year aged to that year, reset by all fires since) and fuel, non-forest land cover and treed wetland are cleared where it is 1, as [fireSenseCovariatesCreate\(youngAge = TRUE\)](#) does for a single year. Fuel is cleared to `logMinB(0)`, the value [fireSenseCovariatesCreate\(\)](#) gives a pixel without that fuel. Each year is then aggregated with mean, so [youngAge](#) becomes the proportion of young pixels in a coarse cell.

Usage

```
prepare_FuelCovsCoarseByYear(
  ...,
  years,
  dataYear,
  nonForest_timeSinceDisturbance,
  firePixelsByYear,
  cutoffForYoungAge,
  rasTemplate,
  fact
)
```

Arguments

... Arguments for [fireSenseCovariatesCreate\(\)](#).
 years character, fire years ("year2001") to build, all described by the same data year.

dataYear	the year nonForest_timeSinceDisturbance describes.
nonForest_timeSinceDisturbance	SpatRaster of time since disturbance at dataYear (for every pixel, not only non-forest; see makeTSD()).
firePixelsByYear	all fires as pixelIDs by year, see firePixelsByYear() .
cutoffForYoungAge	as in fireSenseCovariatesCreate() .
rasTemplate	A raster-like object (typically <code>terra::SpatRaster</code>) used as a spatial template (extent, resolution, projection) when converting the fuel covariate data frame to a raster.
fact	<code>numeric(1)</code> aggregation factor passed to <code>terra::aggregate(fact = ...)</code> , controlling the spatial coarsening scale.

Value

a list named by years of `SpatRasters`, each as [prepare_FuelCovsCoarse\(\)](#) returns, with a `youngAge` layer.

```
prepare_ignitionClimate
```

Aggregate and cache ignition-related climate rasters

Description

Aggregates a list of ignition-relevant climate rasters to a coarser spatial resolution using `terra::aggregate()` (mean across source cells) and caches the resulting list with `reproducible::Cache()`. The cache key is augmented with a robust digest of the input rasters and a caller-provided upstream digest.

Usage

```
prepare_ignitionClimate(
  ignitionClimateList,
  fact,
  digest = NULL,
  useCache = TRUE
)
```

Arguments

ignitionClimateList	A named or unnamed list of raster-like objects (typically <code>terra::SpatRaster</code>) representing historical climate variables used in ignition modeling.
fact	<code>numeric(1)</code> aggregation factor passed to <code>terra::aggregate(fact = ...)</code> .

digest	A digest object (e.g., list, atomic vector) representing upstream processing context, which will be appended to the input-derived digest when constructing the cache key.
useCache	logical (default TRUE). If TRUE, wrap the per-element aggregation in <code>reproducible::Cache()</code> so that re-running with the same inputs returns the cached result. If FALSE, force recomputation.

Details

The function:

1. Computes a robust digest of `ignitionClimateList` using `.robustDigest()` to capture changes in input rasters.
2. Appends that digest to the user-supplied digest to form a composite cache key (`dig3`).
3. Aggregates each raster via `terra::aggregate(fact = fact, fun = mean)`.
4. Wraps the aggregated list in `reproducible::Cache()` with `omitArgs = "x"` (to exclude the ephemeral list element passed to `lapply`) and `.cacheExtra = dig3` to drive cache invalidation from the composite digest.

All rasters are processed independently via `lapply()`, preserving list order and names.

Value

A list of aggregated climate rasters (typically `terra::SpatRaster`), in one-to-one correspondence with `ignitionClimateList`.

Caching and reproducibility

Cache invalidation is controlled by `.cacheExtra = dig3`, which combines:

- a robust digest of the input rasters (`.robustDigest(ignitionClimateList)`), and
- the caller-provided digest (e.g., pipeline/module context).

Assumptions and required environment

The symbols `.robustDigest()` and `reproducible::Cache()` must be available. No validation is performed on input types. Aggregation is controlled by the `fact` parameter.

References

- terra package reference manual: `aggregate()` (spatial raster aggregation).
- reproducible package reference manual and vignettes: `Cache()` and `.cacheExtra`.

See Also

[terra::aggregate\(\)](#), [reproducible::Cache\(\)](#)

Examples

```
## Not run:
# Example (requires `terra` rasters and `reproducible` cache setup):
upstreamDigest <- .robustDigest(list(module = "Fire", version = "1.0.0"))
ignitionClimateCoarse <- prepare_ignitionClimate(
  ignitionClimateList = sim$historicalClimateRasters[
    sim$climateVariablesForFire$ignition
  ],
  fact = 4,
  digest = upstreamDigest
)

## End(Not run)
```

prepare_LightningData *Prepare and cache lightning rasters from Google Drive sources*

Description

Downloads, reads, aggregates, and caches multiple lightning-related rasters (e.g., daily flashes, flash density, positive CG counts) using `reproducible::prepInputs()`, `terra::aggregate()`, and `reproducible::Cache()`. The function builds a cache key that combines local processing context (rasterToMatch, igAggFactor, and the lightning reader) with the remote hashes of the Google Drive files, so cached results are invalidated whenever either local parameters or upstream files change.

Usage

```
prepare_LightningData(rtm, igAggFactor, dPath)
```

Arguments

rtm	<code>terra::SpatRaster</code> or raster-like object used as the target template (i.e., an alignment/extent/projection reference) for reading lightning rasters.
igAggFactor	<code>numeric(1)</code> aggregation factor forwarded to <code>terra::aggregate(fact = ...)</code> , typically used to coarsen lightning rasters for ignition modeling.
dPath	Character. Destination path forwarded to <code>reproducible::prepInputs(destinationPath = ...)</code> ; the directory where downloaded lightning rasters are stored / cached on disk.

Details

Lightning data is given by the authors of <https://www.tandfonline.com/doi/full/10.1080/07055900.2020.1845117>. It is currently not hosted by the authors. We therefore host it on Google Drive.

The function maintains an internal mapping of Google Drive file IDs for four lightning products:

- lightningDays
- lightningDensity
- positiveCG
- positiveCGdensity

For each product, it:

1. Resolves the Drive ID to a human URL, obtains remote metadata (hash),
2. Calls `reproducible::prepInputs()` with a reading function that should return a raster aligned to `sim$rasterToMatch`,
3. Aggregates the raster with `terra::aggregate(fact = P(sim)$igAggFactor)`,
4. Caches the result with `reproducible::Cache()` under a product-specific function label, seeding the cache with both local and remote digests.

A final `Cache()` call wraps the list of per-product results to further stabilize caching across the set.

Value

A named list of aggregated lightning rasters (class list of `terra::SpatRaster`), with names matching: `c("lightningDays", "lightningDensity", "positiveCG", "positiveCGdensity")`.

Caching and reproducibility

The cache key includes:

- The remote hash of each Google Drive resource (so cloud-side updates bust the cache),
- The local context digest of `rasterToMatch`, `igAggFactor`, and the reader, via `.robustDigest`.

This design minimizes recomputation while remaining sensitive to upstream and local changes.

Note

If `prepInputs()` expects a function for `fun`, prefer:

```
fun = function(...) rld(targetFile, to = sim$rasterToMatch)
```

instead of calling `rld(...)` immediately.

References

- **User-provided function source** (this documentation derives from the code you supplied).
- Reproducible workflows with the `reproducible` package and raster handling with `terra` as described in their package manuals and vignettes.

See Also

[reproducible::prepInputs\(\)](#), [reproducible::Cache\(\)](#), [terra::aggregate\(\)](#)

Examples

```
## Not run:
out <- prepare_LightningData(
  rtm = sim$rasterToMatch,
  igAggFactor = P(sim)$igAggFactor,
  rld = readLightningData
)
out$lightningDensity

## End(Not run)
```

```
prepareCovariatesOuter
```

Prepare and (optionally) cache scaled covariates for modeling

Description

Wraps [rescaleCovariates\(\)](#) to rescale features according to the modeling algorithm, and optionally caches results based on a robust digest of the unscaled input data. Returns a list containing the (possibly) rescaled covariates, the modeling formula (if any), per-variable rescalers (if any), the x-axis variable name(s), and the digest used for caching.

Usage

```
prepareCovariatesOuter(unscaledData, algorithm, rescaleVars, useCache = TRUE)
```

Arguments

unscaledData	A <code>data.frame</code> or <code>data.table</code> of covariates. Must include predictor columns and may include response columns such as ignitions and/or escapes (see Details).
algorithm	character. Name of the model algorithm. If it contains "xgb", covariates are standardized via base::scale() (centering and scaling); otherwise, covariates are rescaled by order of magnitude to roughly the [0, 10] range (see Details).
rescaleVars	logical. If TRUE, perform rescaling/standardization according to algorithm; if FALSE, covariates are returned unchanged.
useCache	logical (default TRUE). If TRUE, results are cached using reproducible::Cache() with an extra key derived from a digest of unscaledData.

Details

This function:

- Computes a digest of unscaledData (if useCache = TRUE) using an internal `.robustDigest()` to form a `.cacheExtra` key for [reproducible::Cache\(\)](#).

- Calls `rescaleCovariates()` with `formula = NULL` and `family = NULL` (non-xgboost code-paths were removed).
- Caches the result while omitting the large arguments "covariates" and "formula" from cache key construction (via `omitArgs`), relying on the explicit digest instead.

Expected columns:

- `ignitions`: if present, is preserved through standardization (xgboost path).
- `escapes`: if present, is preserved through standardization (xgboost path).
- `year` or `yr`: used to infer `xvar` (see `rescaleCovariates()`).

Value

A list with elements:

covariates A `data.table` with rescaled/standardized covariates.

formula `NULL` (the non-xgboost formula path is disabled).

ignitionRescalers Named numeric vector of per-variable 10-based rescalers (non-xgboost path) or `NULL` for xgboost / no-rescale.

xvar Character vector indicating which year-like column(s) were found.

digestOfData The digest used for caching, or `NULL` if `useCache = FALSE`.

See Also

`rescaleCovariates()`, `reproducible::Cache()`

Examples

```
## Not run:
library(data.table)
dt <- data.table(pixelID = 1:3,
                 year = c(2001L, 2002L, 2003L),
                 ignitions = c(0, 1, 0),
                 x1 = c(0.01, 2.3, 15))

out <- prepareCovariatesOuter(unscaledData = dt,
                             algorithm = "xgb_classifier",
                             rescaleVars = TRUE,
                             useCache = FALSE)

str(out)

## End(Not run)
```

putBackIntoRaster	<i>Rebuild land cover layers back into a SpatRaster</i>
-------------------	---

Description

Given a template raster (e.g., a flammability map) and a `data.table` containing pixel values for one or more land cover classes, this function reconstructs each class as a raster layer and returns a multi-layer `SpatRaster` whose layer names match the class names supplied in `lcc`.

Usage

```
putBackIntoRaster(lcc, landcoverDT, flammableMap)
```

Arguments

<code>lcc</code>	character vector. Names of the land cover class columns in <code>landcoverDT</code> to rebuild into raster layers (e.g., <code>c("Conifer", "Deciduous", "Shrub")</code>).
<code>landcoverDT</code>	<code>data.table</code> . A <code>data.table</code> that must contain: - a column <code>pixelID</code> with integer indices into the template raster's value vector (as returned by <code>terra::values(flamableMap, mat = FALSE)</code>), and - one numeric column per class named in <code>lcc</code>.
<code>flammableMap</code>	<code>SpatRaster</code> . A <code>terra::SpatRaster</code> used as the spatial template (geometry, extent, resolution, and NA mask) for the output layers.

Details

This function:

1. Creates a working copy of the template raster's values (initialized to 0 on all non-NA cells and keeping original NAs).
2. For each class name in `lcc`, it pulls the corresponding vector from `landcoverDT`, assigns those values into the working vector at positions indicated by `landcoverDT$pixelID`, and then writes these values back into a fresh `terra::rast(flamableMap)` layer.
3. Stacks all layers into a single `terra::rast()` and sets layer names to `lcc`.

Assumptions / requirements

- `flammableMap` is a `terra::SpatRaster` providing the spatial geometry, extent, resolution, and NA mask.
- `landcoverDT` is a `data.table` with a column named `pixelID` that indexes cell positions in the template raster's value vector (i.e., values returned by `terra::values(flamableMap, mat = FALSE)`). Indices must be valid and 1-based (R-style).
- Each element of `lcc` is the name of a numeric column in `landcoverDT`.
- All layers are created in the same CRS, resolution, and extent as `flammableMap`.

Notes

- Non-NA cells in the template are initialized to 0 before assignment, so only positions referenced by pixelID will receive class values; other non-NA positions remain 0 in each layer.
- If you want to preserve existing values outside pixelID, modify the initialization step accordingly.

Value

A multi-layer `terra::SpatRaster` where each layer corresponds to one entry in `lcc`, with layer names set to `lcc`.

See Also

`terra::rast()`, `terra::values()`, `terra::setValues()`

Examples

```
## Not run:
library(terra)
library(data.table)

# Template raster (5x5 example)
r <- rast(nrows = 5, ncols = 5, vals = NA)
# Mark some cells as valid (non-NA)
values(r) <- rep(NA_real_, ncell(r))
valid_idx <- c(3, 5, 7, 13, 17, 20)
tmp_vals <- values(r, mat = FALSE)
tmp_vals[valid_idx] <- 0
r <- setValues(r, tmp_vals)

# Data table with pixel indices and two classes
dt <- data.table(
  pixelID = valid_idx,
  Conifer = c(0.6, 0.2, 0.8, 0.1, 0.0, 0.4),
  Deciduous = c(0.1, 0.3, 0.0, 0.4, 0.5, 0.2)
)

# Rebuild layers
out <- putBackIntoRaster(
  lcc = c("Conifer", "Deciduous"),
  landcoverDT = dt,
  flammableMap = r
)

out
names(out)      # "Conifer" "Deciduous"
plot(out)

## End(Not run)
```

rasterFireBufferDT	<i>this is a wrapper to simplify caching of lapply with bufferForFireRaster. Years are iteratively processed by makeFireID.</i>
--------------------	---

Description

this is a wrapper to simplify caching of lapply with bufferForFireRaster. Years are iteratively processed by makeFireID.

Usage

```
rasterFireBufferDT(
  years,
  fireRaster,
  flammableRTM,
  bufferForFireRaster,
  areaMultiplier,
  minSize = 5000,
  verb = 1,
  cores = 1,
  seed = .bufferSeed(years, fireRaster, flammableRTM, bufferForFireRaster,
    areaMultiplier, minSize)
)
```

Arguments

years	numeric fire years
fireRaster	a SpatRaster with values representing fire years
flammableRTM	a rasterToMatch with binary values where 1 represents flammable pixels, 0 nonflammable, and NA no data
bufferForFireRaster	buffer size used to group discrete patches of burned pixels as belonging to the same fire
areaMultiplier	A scalar that will buffer $\text{areaMultiplier} * \text{fireSize}$
minSize	The absolute minimum size of the buffer & non-buffer together. This will be imposed after areaMultiplier.
verb	Logical or numeric related to how much verbosity is printed. FALSE or 0 is none. TRUE or 1 is some. 2 is much more.
cores	number of processor cores to use
seed	integer. Seeds the per-year draws of buffer pixels. The default is derived from the other inputs, so the same inputs give the same buffers in any session, whatever the state of its random number generator, and the session's random stream is left untouched. Supply a value to get a different set of buffers for the same inputs.

Value

a list of `data.tables` named by year, with cols `ids`, `buffer`, and `pixelID`

`rasterFireSpreadPoints`

Create a list of annual ignition points based on fire raster

Description

Create a list of annual ignition points based on fire raster

Usage

```
rasterFireSpreadPoints(fireBufferDT, flammableRTM)
```

Arguments

`fireBufferDT` a `data.table` with columns `buffer` (1 = burned), `id` (unique fire ID), and `pixelID`

`flammableRTM` @template `flammableRTM`

Value

a list of `sf` point objects

`rbetaBetween`

Generate random beta variates between 2 values and a mean

Description

Generate random beta variates between 2 values and a mean

Usage

```
rbetaBetween(n, l, u, m, shape1, shape2 = NULL)
```

Arguments

`n` number of observations. If `length(n) > 1`, the length is taken to be the number required.

`l` scalar numeric for the lower bound.

`u` scalar numeric for the upper bound.

`m` scalar numeric for the mean.

`shape1` non-negative parameter of the Beta distribution.

`shape2` If provided, passed to `stats::rbeta()`. If not, `m` must be.

See Also[stats::rbeta](#)

readLightningData	<i>Read lightning data</i>
-------------------	----------------------------

Description

Data is not publicly available; it is currently stored in a User Access Controlled location. Data comes from William Burrows. Please contact him to obtain data. This function will get the data (which is in a .txt format) and do all the necessary sequence of calculations to create a SpatRaster.

Usage

```
readLightningData(filename, ...)
```

Arguments

filename	The local filename to read with data.table::fread
...	Passed to postProcess, e.g., to, maskTo

Value

This will return a SpatRaster object with resolution the same as that supplied by to or projectTo. If no SpatRaster is supplied, it will default to resolution of 10000 m (10 km x 10 km).

References

Kochtubajda, B., & Burrows, W. R. (2020). Cloud-to-Ground Lightning in Canada: 20 Years of CLDN Data. Atmosphere-Ocean, 58(5), 316–332. <https://doi.org/10.1080/07055900.2020.1845117>

Examples

```
## Not run:
# The example fetches from a Google Drive folder that requires
# `drive_auth()` credentials. R CMD check runs examples
# non-interactively where no Drive token is available, so the body
# is gated under \dontrun{} to keep docs complete without breaking
# checks.
if (require(googledrive, quietly = TRUE)) {
  crsToUse <- paste("+proj=lcc +lat_0=0 +lon_0=-95 +lat_1=49 +lat_2=77",
    "+x_0=0 +y_0=0 +datum=NAD83 +units=m +no_defs")
  ras <- terra::rast(
    terra::ext(-1115000, -376750, 7267000, 7874000),
    resolution = 250,
    vals = 1,
```

```

    crs = crsToUse
  )
  ## Google Drive folder:
  ## <https://drive.google.com/drive/folders/1ftpKKU08BZcJ9ba4LDWcmF5uQkpRGmku>
  data <- list(
    lightningDays = "1jeKJquhVJsesoNk2EPP1QZkttX3Zwp5c",
    lightningDensity = "12fnhfKtER-JXkl06M4_yZ3GvpZWtQlIr",
    positiveCG = "1bn6cQ23tvPicFLHn1tz4Z3AqDzJI4r60",
    positiveCGdensity = "1GNixhXj1Ex1jT0tWXfhmxf-dX3ze1a4"
  )
  lightning <- Map(
    url = data,
    f = function(url) {
      reproducible::prepInputs(
        url = url,
        fun = readLightningData(targetFile, to = ras),
        destinationPath = file.path(tempdir(), "lightning_data")
      )
    }
  )
}

## End(Not run)

```

```
removeBufferedFiresOutsideRTM
```

Remove buffered fires in fireBufferedListDT that are outside flammableRTM

Description

Remove buffered fires in fireBufferedListDT that are outside flammableRTM

Usage

```
removeBufferedFiresOutsideRTM(fireBufferedDT, flammableRTM)
```

Arguments

fireBufferedDT data.table containing indices for buffered annual fires

flammableRTM a rasterToMatch with binary values where 1 represents flammable pixels, 0 nonflammable, and NA no data

Value

fireBufferedDT excluding fires with indices (burned or unburned) outside flammableRTM

rescaleCovariates	<i>Rescale/standardize covariates for modeling</i>
-------------------	--

Description

Applies either order-of-magnitude rescaling to approximately confine features to the [0, 10] range (non-xgboost) or standardization via `base::scale()` (xgboost path). Preserves key response columns when present.

Usage

```
rescaleCovariates(formula, covariates, rescaleVars, modelAlgorithm)
```

Arguments

formula	A model formula or NULL. Non-xgboost paths would parse and validate it, but the surrounding code path currently sets formula = NULL and disables non-xgboost modeling in the caller.
covariates	A data.frame or data.table of covariates and possibly responses (e.g., ignitions, escapes).
rescaleVars	logical. If TRUE, features are rescaled/standardized according to modelAlgorithm. If FALSE, returned unchanged (with ignitionRescalers = NULL).
modelAlgorithm	character. If the string contains "xgb", apply <code>base::scale()</code> (centering and scaling) to all columns except yearChar, ignitions, and escapes. Otherwise, log-10 order-of-magnitude rescaling is applied to all non-excluded covariates whose maxima fall outside [0, 10].

Details

Year variable inference: If any of `c("year", "yr")` match column names (case-insensitive), xvar is set to the intersection. Otherwise, the code references rows (a TODO), which is undefined here. This appears to be a bug or placeholder and will error unless rows exists in scope; consider providing a default (e.g., NULL) or removing.

Non-xgboost path:

- Excludes pixelID, ignitions, escapes, year, and yearChar from rescaling.
- Computes per-variable rescalers $\leftarrow 10^{(\text{floor}(\log_{10}(\text{abs}(\text{max}))))}$ for variables whose maxima fall outside [0, 10].
- Calls `rescaleVarsByMagnitude(covariates, ignitionRescalers)` (must exist in scope).

xgboost path:

- Standardizes numeric columns except yearChar, ignitions, and escapes.
- Preserves original ignitions and (if present) escapes columns.
- Attaches centering/scaling attributes under `attr(covariates, "scaleData")`.

External references:

- Uses `ignitionsTxt` and `escapesTxt` to refer to the expected response column names. If these are not available, ensure compatible strings are provided (see exported constants below).

Value

A list with elements:

covariates A `data.table` after rescaling/standardization.

formula The input formula (often `NULL` in current usage).

ignitionRescalers Named numeric vector of per-variable rescalers (non-xgb) or `NULL`.

xvar Character vector with detected year-like column(s) or (bug) rows.

See Also

`base::scale()`

Examples

```
## Not run:
library(data.table)
dt <- data.table(ignitions = c(0,1,0),
                 escapes = c(0,0,1),
                 year = 2001:2003,
                 x1 = c(0.05, 12, 100),
                 x2 = c(5, 6, 7))

# XGBoost-like standardization
out_xgb <- rescaleCovariates(NULL, dt, rescaleVars = TRUE, modelAlgorithm = "xgb_tree")
str(out_xgb)

# Non-xgboost magnitude rescaling
out_other <- rescaleCovariates(NULL, dt, rescaleVars = TRUE, modelAlgorithm = "glm")
str(out_other)

## End(Not run)
```

rescaleKnown2

rescale function no.2

Description

rescale function no.2

Usage

```
rescaleKnown2(x, minNew, maxNew, minOrig, maxOrig)
```

Arguments

x	a vector to be rescaled
minNew	the minimum of the new range
maxNew	the max of the new range
minOrig	the minimum of the original data
maxOrig	the maximum of the original data

Value

the rescaled vector

rescaleVarsByMagnitude	<i>Rescale a data.table according to a named vector of rescalers</i>
------------------------	--

Description

Rescale a data.table according to a named vector of rescalers

Usage

```
rescaleVarsByMagnitude(dt, rescalers)
```

Arguments

dt	a data.table of covariates
rescalers	a named vector of rescalers

Value

the data.table with rescaled values

rescorePopulation	<i>Re-score a population with replication, and pick the best members by their mean</i>
-------------------	--

Description

DEoptim evaluates each surviving member only once, so on a noisy objective the values it holds are partly luck: a lucky low draw survives and is reported as "best". Re-scoring the final population several times and ranking by the mean sees through that. In eight FireSense fits (2026-09-17), DEoptim's best member ranked 1st to 6th of 60 by its replicated mean, and the ledger's "5 best" were five copies of that one member.

Usage

```
rescorePopulation(pop, fn, reps = 10L, cl = NULL, seed = 1L, fnArgs = list())

bestByReplicatedMean(pop, scores, n = 5L)
```

Arguments

pop	A matrix, one parameter set per row, with the parameter names as column names.
fn	The objective function; called as <code>fn(par, ...)</code> with <code>fnArgs</code> .
reps	Evaluations per member.
cl	A cluster from <code>parallel::makeCluster()</code> , or <code>NULL</code> to evaluate here.
seed	Base seed.
fnArgs	Named list of further arguments to <code>fn</code> .
scores	The output of <code>rescorePopulation()</code> for <code>pop</code> .
n	How many members to return.

Details

`rescorePopulation()` evaluates every row of `pop` `reps` times; each evaluation has its own seed (`seed + its index`), so the result is reproducible. With a cluster the evaluations run on its workers, which must already hold whatever `fn` reads from their global environment (as for `DEoptim`).

Value

`rescorePopulation()`: a `data.table` with member (row of `pop`), rep and value.

`bestByReplicatedMean()`: a list with `params` (a `data.table`, one row per member, best first), `objFunVal` (their replicated means), `objFunValSD` and `member` (their rows in `pop`). Identical rows of `pop` count once, so the members returned are distinct.

 resolveDoFit

Resolve the doFit parameter of the FireSense fit modules

Description

`fireSense_spreadFit` and `fireSense_ignitionFit` decide whether to fit from one three-valued parameter, `doFit`: `NA` (default) fits only when the ledger has no usable fit for the polygon, `TRUE` always fits (replacing a ledger fit), `FALSE` never fits (use the ledger fit; stop when there is none). `refitExisting` and `stopIfNoPreRunFit` are deprecated aliases: `refitExisting = TRUE` is `doFit = TRUE`; `stopIfNoPreRunFit = TRUE` is `doFit = FALSE` (`stopIfNoPreRunFit = FALSE` only allowed a fit, which is what `NA` does); when both are set `refitExisting` wins. An explicit (non-`NA`) `doFit` is never overridden by an alias. A module that does not have an alias leaves it `NA`.

Usage

```

resolveDoFit(
  doFit = NA,
  refitExisting = NA,
  stopIfNoPreRunFit = NA,
  module = "the fit module",
  message = TRUE
)

```

Arguments

doFit logical, length one: NA, TRUE or FALSE.

refitExisting, stopIfNoPreRunFit the deprecated aliases; NA means not set.

module character; module name used in the deprecation message.

message logical; send the deprecation message when an alias is set (once per call).

Value

NA, TRUE or FALSE.

runDEoptim	<i>Run DEoptim</i>
------------	--------------------

Description

Provides a wrapper around [DEoptim::DEoptim](#), setting up the multiple cluster connections. This will only work if ssh keys are preconfigured on all machines (if using multiple machines).

termsInDEoptim() is deprecated: it counted every parameter not in the formula as a "logit" term, so the per-year random effect yearSpreadSD was reported as a third logistic parameter. runDEoptim() now names the parameters with names(lower).

Usage

```

runDEoptim(
  landscape,
  annualDTx1000,
  nonAnnualDTx1000,
  fireBufferedListDT,
  historicalFires,
  itermax,
  initialpop = NULL,
  NP = NULL,
  trace,
  strategy,
  cores = NULL,

```

```

paths,
libPath = .libPaths()[1],
logPath = tempfile(sprintf("runDEoptim_%s_", format(Sys.time(),
"%Y-%m-%d_%H%M%S")), fileext = ".log"),
doObjFunAssertions = getOption("fireSenseUtils.assertions", TRUE),
iterStep = 25,
lower,
upper,
mutuallyExclusive,
formulaToFit,
objFunCoresInternal,
covMinMax = covMinMax,
tests = c("SNLL", "adTest"),
maxFireSpread,
Nreps,
thresh = 550,
.c = 0.5,
DEoptimControl = list(),
.verbose,
visualizedDEoptim = logPath,
.plots = "screen",
plotEvery = 25L,
.plotSize = list(height = 1600, width = 2000),
rep = 1L,
runName = "",
progressFile = NULL,
nCoresNeeded = NULL,
rescoreReps = 10L,
sizeLik = "kde",
sizeLikDf = 5,
weighted = TRUE,
adWeight = "auto",
link = NULL,
escapeSizeHa = NULL,
jumpTries = 0,
jumpMeanDist = 0,
yearAreaWeight = 0,
areaDistWeight = 0,
penaliseRunaways = TRUE,
runawayEdgeFrac = fireSenseRunawayEdgeFrac,
runawayEdgeMin = fireSenseRunawayEdgeMin,
runawayBufferMultiple = NULL,
profileReps = 0L,
simulateMembers = 0L,
penaliseCapHits = NULL,
covCentre = NULL
)

```

```
termsInDEoptim(fireSense_spreadFormula, thresh, numParams)
```

Arguments

landscape	A <code>SpatRaster</code> which has the correct metadata associated with the pixelID and cells of other objects in this function call.
annualDTx1000	A list of <code>data.table</code> objects. Each list element will be from 1 year, and it must be the same length as <code>fireBufferedListDT</code> and <code>historicalFires</code> . All covariates must be integers, and must be 1000x their actual values.
nonAnnualDTx1000	A list of <code>data.table</code> objects. Each list element must be named with a concatenated sequence of names from <code>names(annualDTx1000)</code> , e.g., 1991_1992_1993. It should contain all the years in <code>names(annualDTx1000)</code> . All covariates must be integers, and must be 1000x their actual values.
fireBufferedListDT	A list of <code>data.table</code> objects. It must be same length as <code>annualDTx1000</code> , with same names. Each element is a <code>data.table</code> with columns: <code>buff...TODO: INCOMPLETE</code>
historicalFires	A named list of <code>data.frames</code> (one per year, names matching those of <code>annualDTx1000</code>), each with columns <code>cells</code> (pixel indices of fire ignitions) and <code>size</code> (fire size in pixels). Used as the observed reference against which simulated fires are scored.
itermax	Maximum number of iterations for the <code>DEoptim::DEoptim</code> algorithm. Passed to <code>DEoptim::DEoptim.control</code> .
initialpop	Optional. A matrix or vector specifying the initial population for <code>DEoptim::DEoptim</code> . If <code>NULL</code> , <code>DEoptim::DEoptim</code> generates one. Passed to <code>DEoptim::DEoptim.control</code> .
NP	Optional. The number of population members (individuals) in <code>DEoptim::DEoptim</code> . If <code>NULL</code> , <code>DEoptim::DEoptim</code> sets a default. <code>DEoptim::DEoptim.control</code> .
trace	Integer or Logical. Controls the level of tracing information printed by <code>DEoptim::DEoptim</code> during optimization. Passed to <code>DEoptim::DEoptim.control</code> .
strategy	Integer [1,10]. Defines the <code>DEoptim::DEoptim</code> strategy variant to use. Passed to <code>DEoptim::DEoptim.control</code> .
cores	A numeric (for running on localhost only) or a character vector of machine names (including possibly "localhost"), where the length of the vector indicates how many cores should be used on that machine.
paths	list of paths containing the <code>cachePath</code> to store cache. Should likely be <code>cachePath(sim)</code> . See <code>SpaDES.core::paths</code> .
libPath	A character string indicating an R package library directory. This location must exist on each machine, though the function will make sure it does internally.
logPath	A character string indicating what file to write logs to. This <code>dirname(logPath)</code> must exist on each machine, though the function will make sure it does internally.
doObjFunAssertions	logical indicating whether to do assertions.

iterStep	Integer. Must be less than itermax. This will cause DEoptim::DEoptim to run the itermax iterations in ceiling(itermax / iterStep) steps. At the end of each step, this function will plot, optionally, the parameter histograms (if visualizedEoptim is TRUE)
lower	Numeric vector. Lower bounds for the parameters being optimized. Passed to DEoptim::DEoptim . If named and including yearSpreadSD (which must be last), the per-year random effect (a seasonal departure) of .objfunSpreadFit() (fitYearSpreadSD) is fitted, in the fit and in the re-score.
upper	Numeric vector. Upper bounds for the parameters being optimized. Passed to DEoptim::DEoptim .
mutuallyExclusive	If there are any covariates, e.g., youngAge, that should be considered mutually exclusive, then this can be done here (i.e., "if youngAge is non-zero, should vegPC2 be set to zero"). A named list, where the name of the list element must be a single covariate column name in either annualDTx1000 or nonAnnualDTx1000. The list content should be a "grep" pattern with which to match column names, e.g., "vegPC". The values of all column names that match the grep value will be set to 0, whenever the name of that list element is non-zero. Default is list("youngAge" = list("vegPC")), meaning that all columns with vegPC in their name will be set to zero wherever youngAge is non-zero.
formulaToFit	Passed to DEoptim::DEoptim
objFunCoresInternal	Integer. The number of cores to use for potential parallelization <i>within</i> a single call to the objective function (.objfunSpreadFit()). This is distinct from the parallelization managed by DEoptim::DEoptim across population members.
covMinMax, tests, maxFireSpread, Nreps, .verbose	Passed to .objfunSpreadFit() .
thresh	The threshold for accepting fits; e.g., from mod\$thresh.
.c	Numeric in (0, 1]. DEoptim's c, the speed of crossover adaptation (used by strategy = 6). Passed to DEoptim::DEoptim.control() ; a c in DEoptimControl wins.
DEoptimControl	Named list of further DEoptim::DEoptim.control() settings (for example CR, F, p, reltol), passed through clusters::clusterSetup() to DEoptim. NP is the number of workers the cluster gets.
visualizedEoptim	Logical. If TRUE, then histograms will be made of DEoptim::DEoptim outputs.
.plots	Character string. Specifies the plot destination device (e.g., "screen", "png", "pdf"). Passed to internal plotting functions (likely via SpaDES.core::Plots()). Default "screen".
plotEvery	Integer. Generations between DEoptim progress figures; the final figures are always drawn. Passed to clusters::DEoptimIterative() . It is not part of the fit's cache key, so changing it does not refit. Default 25.
.plotSize	List specifying plot height and width, in pixels.

rep	Integer. An identifier for the replication number of this optimization run. Used in cache tags and plot filenames. Default 1L.
runName	Character string used to label this run. Forwarded to <code>clusters::DEoptimIterative()</code> and used as a suffix for the cache <code>.functionName</code> so that runs with different <code>runName</code> values get distinct cache entries. Default "" (no suffix).
progressFile	Passed to <code>clusters::DEoptimIterative()</code> : NULL (default) lets it choose <code>DEoptimProgress_<runName></code> in <code>visualizedEoptim</code> ; FALSE writes none; or a path. The file has one row per generation and a FINISHED row at the end. It is not part of the cache key.
nCoresNeeded	Integer. How many workers to request for the DEoptim cluster. NULL (the default) takes <code>deoptimPopulationSize()</code> of <code>length(lower): 30 for 4 to 7 parameters, 40 for 8 to 12, 50 for 13 to 15</code> . DEoptim's NP is set to the number of workers the cluster actually gets, so a smaller allocation means a smaller population.
rescoreReps	Integer. After the fit, each member of the final population is evaluated this many more times (full evaluations, no early stop) and the result is attached to the returned object as <code>attr(DE, "finalRescore")</code> , for <code>bestByReplicatedMean()</code> . 0 skips it.
sizeLik, sizeLikDf, weighted, adWeight	Passed to <code>.objfunSpreadFit()</code> , in the fit AND in the final-population re-score, so both evaluate the same objective. The defaults are that function's. Before these existed, every fit used those defaults whatever the caller wanted.
link	Passed to <code>.objfunSpreadFit()</code> in the fit and the re-score: NULL or "logistic3pUpper". Needed because DEoptim may hand the objective an unnamed par.
escapeSizeHa	Passed to <code>.objfunSpreadFit()</code> in the fit and the re-score: the size (ha) a fire must reach to count as escaped. NULL keeps the historical rules.
jumpTries, jumpMeanDist, yearAreaWeight, areaDistWeight	Passed to <code>.objfunSpreadFit()</code> , in the fit and the final-population re-score. Off by default (0).
penaliseRunaways	Passed to <code>.objfunSpreadFit()</code> in the fit and the re-score: TRUE (default) scores a simulated fire that reaches the edge of its own buffer as a runaway, not as a fire of the size it reached. The edge ring of every buffer is computed here, once (<code>addBufferEdge()</code>).
runawayEdgeFrac, runawayEdgeMin	Passed to <code>.objfunSpreadFit()</code> in the fit and the re-score: a replicate is a runaway only if it burns at least <code>max(runawayEdgeMin, ceiling(runawayEdgeFrac * n))</code> of the n edge-ring pixels of its fire (at most n). Defaults <code>fireSenseRunawayEdgeFrac</code> and <code>fireSenseRunawayEdgeMin</code> .
runawayBufferMultiple	Passed to <code>.objfunSpreadFit()</code> in the fit and the re-score: NULL (default), or the multiple of its own fire's buffer size that a runaway counts as in the size-based terms.
profileReps	Integer. If above 0, <code>profileCoefficients()</code> is run around the best member of the re-score, each point evaluated this many times, and attached as <code>attr(DE, "profile")</code> . Needs <code>rescoreReps > 0</code> . Costs about <code>6 * nCoefficients * profileReps</code> evaluations.

simulateMembers	Integer. If above 0, that many best members of the re-score simulate the observed fires without the size cap (<code>simulateFireSizes()</code>), attached as <code>attr(DE, "fitSims")</code> for <code>scoreFireSizes()</code> and <code>linkSaturation()</code> . Needs <code>rescoreReps > 0</code> .
penaliseCapHits	Deprecated and ignored; use <code>penaliseRunaways</code> .
covCentre	Passed to <code>.objfunSpreadFit()</code> in the fit, the re-score and the profile and simulation diagnostics: values subtracted from the rescaled covariates, from <code>spreadCovCentre()</code> when the formula has an intercept. NULL (default) centres nothing, and then the call, and so its cache key, is what it was before this argument existed.
fireSense_spreadFormula	The formula to be submitted to <code>DEoptim::DEoptim()</code> , from e.g., <code>sim\$fireSense_spreadFormula</code> .
numParams	The number of parameters (TODO: improve description)

Value

The result of the `clusters::DEoptimIterative()` call. This is typically a list where each element contains the `DEoptim::DEoptim` object state after a block of `iterStep` iterations. The final element represents the state after `itermax` iterations or upon early stopping.

runELFs	<i>Run the ELF module and return the ELF names or maps</i>
---------	--

Description

Extracting only the "ELF" module, runs it via `SpaDES.core::simInitAndSpades2()` and returns the vector of polygon IDs from `spreadFitPreRun`, or the ELF maps. The run is not cached here: the module caches the national ELF list itself, keyed on its `ELFversion` parameter alone and shared through the cloud cache, so a run only downloads it.

Usage

```
runELFs(
  preRunSetupProject,
  whatOut = c("fittedNamesOnly", "allNames", "maps"),
  urlELFresults = paste0("https://drive.google.com/file/d/",
    "1tkC944mPzR9-y-qCMB5o2cAR_1MoDz4/view?usp=drive_link")
)
```

Arguments

`preRunSetupProject`
list. A SpaDES project object prepared for `SpaDES.core::simInitAndSpades2()`, containing `$modules` and `$paths$modulePath`.

whatOut	string. One of c("fittedNamesOnly", "allNames", "maps"). Partial matching is active (i.e., fit will do fittedNamesOnly). Will default to the first one if more than one supplied. If not fittedNamesOnly nor allNames, then will do maps. fittedNamesOnly will return the vector of the .ELFind names that have SpreadFit`` values in the Cloud Geo Cache. allNameswill return all .ELFind names, even i
urlELFresults	Unused and deprecated; kept so existing calls still work. runELFs() no longer reads this Google Drive file or uploads to it.

Details

The function:

1. Restricts preRunSetupProject\$modules to entries matching "ELFs".
2. Initializes and runs the simulation with SpaDES.core::simInitAndSpades2().
3. Returns sim\$spreadFitPreRun[[fireSenseUtils::polygonIDTxt]], or the names or maps chosen by whatOut.

The function expects preRunSetupProject to be a SpaDES project list with at least:

- \$modules: character vector of module names.
- \$paths\$modulePath: base directory containing module folders.

Value

A vector corresponding to sim\$spreadFitPreRun[[fireSenseUtils::polygonIDTxt]]. Currently, no high arctic ELFs are returned (e.g., Ecoprovinces starting with either 1. or 2.), nor 3.2.1 or 3.2.4, which have no SCANFI species.

Author(s)

Based on the provided code snippet.

See Also

[simInitAndSpades2](#)

Examples

```
## Not run:
# Assuming 'prj' is a valid SpaDES project list:
# prj <- somePreparationFunction(...)
ids <- runELFs(prj)

## End(Not run)
```

sizeRuleLandCover	<i>Path to the land-cover map the size rule uses</i>
-------------------	--

Description

SCANFI v2 land cover for 2020 in Canada LCC class codes, the same map and source as [LandR: :prepInputs_SCANFI_LCC_](#) starts from. The file's name carries its release, so a copy in destinationPath or getOption("reproducible.inputPaths") is used as it is, with no call to Google Drive; otherwise it is downloaded.

Usage

```
sizeRuleLandCover(destinationPath)
```

Arguments

destinationPath	Directory to download into.
-----------------	-----------------------------

Value

Character path of the .tif.

spreadCovCentre	<i>The centre of each covariate of a spread fit: its mean over the data the objective uses</i>
-----------------	--

Description

With an intercept in the model the covariates are centred (covCentre of [.objfunSpreadFit\(\)](#)), so the coefficients describe variation and the intercept the level. The centre is the mean of each covariate of the formula, over every pixel-year in annualDTx1000, after the rescaling and the mutual exclusivity the objective applies and before it centres: the very values it goes on to centre.

Usage

```
spreadCovCentre(
  annualDTx1000,
  nonAnnualDTx1000,
  formulaToFit,
  covMinMax = NULL,
  mutuallyExclusive = list(youngAge = c("class", "nf"))
)
```

Arguments

annualDTx1000, nonAnnualDTx1000, covMinMax, mutuallyExclusive

As in `.objfunSpreadFit()`.

formulaToFit The spread formula; the intercept is not a covariate and has no centre.

Value

NULL when the formula has no intercept (nothing is centred), else a named list, one mean per covariate, to give as covCentre.

spreadDesignCols	<i>The columns of the spread model's design, in the order of its coefficients</i>
------------------	---

Description

The terms of the formula, preceded by spreadInterceptTxt when the formula has an intercept ($\sim 1 + a + b$; the module formulas are $\sim 0 + \dots$ and have none). The intercept is a column of 1s that spreadProbFromIntegerCovs() adds after rescaling and centring, so it is multiplied by its coefficient like every covariate, and paramsSeparate(par, length(spreadDesignCols(f))) finds it as the first covariate coefficient. `spreadCovCols()` drops it.

Usage

```
spreadDesignCols(formulaToFit)
```

```
spreadCovCols(colsToUse)
```

Arguments

formulaToFit The spread formula, a character string or a formula; NULL gives no columns.

colsToUse A character vector from spreadDesignCols().

Value

A character vector.

spreadCovCols(): colsToUse without the intercept, i.e. the columns that are covariates (rescaled, centred, and in covMinMax).

spreadFitAdditionalColNames

Default values for various things

Description

Default values for various things

Usage

spreadFitAdditionalColNames

spreadFitFilenameFor *The shared SpreadFit ledger files*

Description

fireSense_spreadFit writes each fit as a row of an rds file in a Google Drive folder (its spreadFitGoogleDriveFolder); fireSense_ELFS and fireSense_dataPrepFit read them. With spreadFitFilename = "latest" those modules use these functions instead of one named file.

latestSpreadFits() returns, for every polygon, its rows from the most recently modified ledger file that has it. Files are read newest first; with polygonIDs, reading stops once all of them are found. A file already in destinationPath with the same MD5 as on Drive is not downloaded again; one that differs is downloaded again with [reproducible::preProcess\(\)](#), into a temporary folder first.

Usage

```
spreadFitFilenameFor(fireYears)
```

```
spreadFitFileTag
```

```
latestSpreadFits(
  cloudFolderID = spreadFitLedgerFolder(),
  destinationPath,
  polygonIDs = NULL
)
```

Arguments

fireYears The fire years the fit used; only the first and last matter.

cloudFolderID The Google Drive folder (url or id) holding the ledger files; default [spreadFitLedgerFolder\(\)](#).
destinationPath

Local folder for the downloaded files.

polygonIDs Optional. The polygons (ELF ids) wanted; NULL reads every file.

Details

spreadFitFileTag marks the files whose fits use the current model: fuel biomass on the linear scale (`fuelLogToLinear()`), escaped fires starting at 50 ha (`escapeSizeHa`), and the objective with the annual-area and area-distribution terms. The date (2026-09-25) is when this model started, i.e. the model version. Files with an older tag ("`_linearFuel_esc50`": the same model before it was dated; "`_linearFuel`": 1-pixel escape; none: log fuel) hold fits of an earlier model, which `fireSense_spreadPredict` would apply wrongly, so "latest" never reads them. A fit made under a new model needs a new tag, so the old files drop out of "latest".

Value

`spreadFitFilenameFor()`: the file name a fit over `fireYears` is written to, e.g. "`fireSenseParams_1985-2024_linearFu`

`latestSpreadFits()`: the ledger rows, in the form `fireSense_spreadFit` writes them (a `data.frame` with a geometry column), or NULL when no file matches. Attribute "`spreadFitFiles`" names the file each polygon came from.

spreadFitValidationData

Compare a spread fit with the data it was fitted to

Description

A response curve of $\text{logit}(p)$ against a covariate cannot show misfit: its points are the model's own predictions, so they sit on the fitted curve by construction. What can show misfit is the share of pixels that actually burned against the share that burned in the fit's own simulations, binned by each covariate.

Usage

```
spreadFitValidationData(
  par,
  landscape,
  annualDTx1000,
  nonAnnualDTx1000,
  formulaToFit,
  historicalFires,
  fireBufferedListDT,
  covMinMax = NULL,
  mutuallyExclusive = list(youngAge = c("class", "nf")),
  covCentre = NULL,
  Nreps = 10,
  lowerSpreadProb = spreadProbFloor,
  maxFireSpread = spreadProbCeiling,
  link = NULL,
  fitYearSpreadSD = NULL,
  seed = NULL,
```

```

    ...
  )

plotSpreadFitValidation(d, nBins = 10L, covariates = NULL, title = NULL)

plotSpreadFitResponse(d, covariates = NULL, bins = 40L, title = NULL)

```

Arguments

<code>par</code>	The fitted parameters, as given to <code>.objfunSpreadFit()</code> (without <code>hillSlope1</code> and <code>inflectionPoint1</code> , which are fixed at 1; <code>yearSpreadSD</code> last if fitted).
<code>landscape</code>	A <code>SpatRaster</code> with extent, resolution, and projection (<code>crs</code>) used for <code>SpaDES.tools::spread2</code> .
<code>annualDTx1000</code>	A list of <code>data.table</code> class objects. Each list element is data from a single calendar year, and whose name takes the form "yearxxxx", where xxxx is the four-digit year. The columns in the <code>data.table</code> must integers, that are 1000x their actual values as this function will divide by 1000.
<code>nonAnnualDTx1000</code>	Like <code>annualDTx1000</code> , but with where each list element will be used for >1 year. The names of the list elements must be of the form "yearxxxx_yearyyyy_yearzzzz" where xxxx, yyyy, and zzzz represent the four-digit calendar years for which that list element should be used. The columns are variables that are used for more than 1 year.
<code>formulaToFit</code>	Formula, put provided as a character string, not class formula. (if it is provided as a class formula, then it invariably will have an enormous amount of data hidden in the formula environment; this is bad for <code>DEoptim::DEoptim</code>)
<code>historicalFires</code>	A named list of <code>data.frames</code> (one per year, names matching those of <code>annualDTx1000</code>), each with columns <code>cells</code> (pixel indices of ignitions) and <code>size</code> (fire size in pixels). The observed fire record against which the simulation is compared.
<code>fireBufferedListDT</code>	A named list of <code>data.tables</code> (one per year, names matching <code>annualDTx1000</code>), each with columns <code>pixelID</code> , <code>buffer</code> (1 if the pixel is inside a buffered fire footprint, 0 otherwise), and <code>ids</code> (a unique identifier per fire). Restricts the set of pixels used when fitting and scoring spread.
<code>covMinMax</code>	This is a 2 row by multiple column <code>data.frame</code> indicating the minimum and maximum values of the original covariate data values. These will be used to rescale the covariates internally so that they are all between 0 and 1. It is important to not simply rescale internally here because only 1 year is run at a time; all years must be rescaled for a given covariate by the same amount.
<code>mutuallyExclusive</code>	Named list of vectors describing groups of model terms that must not be active together (mutually exclusive in the formula). Default <code>list("youngAge" = c("class", "nf"))</code> .
<code>covCentre</code>	Optional named list or vector of values subtracted from the rescaled covariates of the same name. <code>NULL</code> (default) does no centring. Must be the same at fit and predict. <code>spreadCovCentre()</code> gives the centres to use with an intercept.

Nreps	Simulations per fire; use the fit's Nreps.
lowerSpreadProb	Numeric. Lower bound for spreadProb; if a candidate spreadProb falls at or below this, the call exits early to avoid wasted simulation work. Default spreadProbFloor .
maxFireSpread	A value for spreadProb that is considered impossible to go above. Default spreadProbCeiling , also the upper bound of maxAsymptote. maxAsymptote is the spread-probability ceiling in a typical year. The year random effect (yearSpreadSD) is added on the logit of the final spread probability, after the link, so in a given year p can exceed maxAsymptote or fall below lowerSpreadProb; that is intended. There is no absolute cap, because spreadCpp does not need one; maxAsymptote is bounded because runaway fires are slow to simulate and wasted if the parameters are wrong.
link	Passed to logisticAll() : NULL for the form implied by the logistic parameters, or "logistic3pUpper" for the 3-parameter logistic with Stukel's upper tail, whose fourth logistic parameter is upperTail1. Pass it explicitly when fitting: par may arrive unnamed.
fitYearSpreadSD	If TRUE, the last element of par is the sd of a per-year random effect on the logit of spread probability, a seasonal departure: in each replicate the year draws one $\text{eps} \sim N(0, \text{sd}^2)$ and every pixel of that year spreads with $\text{plogis}(\text{qlogis}(p) + \text{eps})$, so all of the year's fires are bigger or smaller together. Without it the simulated fire sizes were too alike: medians too large and the largest fires too small at once (2026-09-23). $\text{sd} = 0$ is the model without it, exactly (no random numbers are drawn). NULL (default): TRUE when par is named and includes yearSpreadSD. DEoptim passes par unnamed, so runDEoptim() sets it explicitly.
seed	If not NULL, the simulations run with this seed and the session's random number stream is restored afterwards.
...	Further arguments to .objfunSpreadFit() , e.g. escapeSizeHa, jumpTries, jumpMeanDist: give it what the fit was given. The objective's gates on implausible landscapes are lifted (landscape1stQuantileThresh = Inf) unless given here.
d	The output of spreadFitValidationData().
nBins	Number of quantile bins per covariate.
covariates	The covariates to plot; all of the formula's by default.
title	Plot title; a default is used when NULL.
bins	Number of hexagons (or squares, without hexbin) across each panel.

Details

- `spreadFitValidationData()` runs the objective's simulations for the fitted parameters, recording which pixels burned (`.objfunSpreadFit(returnBurned = TRUE)`), and returns one row per spreadable pixel-year in the buffers of the fitted fires.
- `plotSpreadFitValidation()` plots, per covariate, the observed and the simulated burned share by quantile bin of that covariate.

- `plotSpreadFitResponse()` plots the fitted response curves (the model, not the data).

Both plot functions take the data as their first argument and return a ggplot, so they can be given to `SpaDES.core::Plots()` as fn.

Value

`spreadFitValidationData()`: a `data.table`, one row per spreadable pixel-year inside the buffers of the fires the objective fits (`minFireSize`, `escapeSizeHa`): `year` (the name of the fire year), `pixelID`, `observed` (1 if the pixel is inside an observed fire, else 0), `simulated` (the share of the Nreps simulations in which it burned), `p` (its fitted spread probability, without the per-year random effect), and one column per covariate of the formula with its raw value (the stored integer / 1000, before rescaling and mutual exclusivity). `attr(, "spreadFitValidation")` holds what the plots need: the parameters, `covMinMax`, `covCentre`, `lowerSpreadProb`, `link`, `Nreps`, the names of the covariates and of the annual (climate) ones, and `yearsNotSimulated` – years the objective declined to simulate, which have no rows.

`plotSpreadFitValidation()`: a ggplot, one panel per covariate. Pixel-years are binned by quantiles of the covariate: all of them for climate covariates, only those where it is present (above its `covMinMax` minimum, else above 0) for the others, and one bin per value for covariates with two values or fewer. Each bin shows the observed burned share and the mean simulated burned share, at the bin's mean covariate value, with its number of pixel-years along the bottom.

`plotSpreadFitResponse()`: a ggplot, one panel per covariate with more than two values. The line is `logit(p)` the fitted parameters give a "pure stand": that covariate alone, every other covariate absent (0 after rescaling: other fuels, youngAge, non-forest), and climate at its median. Behind it, the pixel-years' own predicted `logit(p)` (only where the covariate is present, except for climate), up to its 99.5th percentile. Those follow the curve by construction: the figure shows the model's response, not how well it fits.

`spreadFitWorkerMemoryGB`

Memory (GB) assumed per DEoptim worker before a fit has a memory record

Description

`runDEoptim()` sets `options(clusters.workerMemoryGB)` to this, unless the user has set it, so `clusters` ($\geq 0.0.75$) caps workers per host by free memory from a fit's first cluster build. Measured 2026-10-09 over 457 spread-fit workers on 15 hosts: peak resident memory median 4.5 GB, 90th percentile about 6.5 GB, maximum 14.1 GB. Once a fit has run a chunk, its own record is used.

Usage

`spreadFitWorkerMemoryGB`

spreadIndicatorCols	<i>Indicator covariates of the spread model</i>
---------------------	---

Description

The covariates that are 0 or 1: youngAge, the non-forest land-cover groups (nflCC_*) and the species-mode treedWetland indicator (not the biomass treedWetland_agb). Their range is always $c(0, 1)$ ([spreadIndicatorRanges\(\)](#)), not the range of the data: in an ELF where one is constant the data's range is zero wide and rescaling divides by zero.

Usage

```
spreadIndicatorCols(covNames)

spreadIndicatorRanges(covNames)
```

Arguments

covNames character vector of covariate column names.

Value

spreadIndicatorCols(): the names among covNames that are indicators. spreadIndicatorRanges(): a named list, $c(0, 1)$ for each of them.

spreadProbBounds	<i>The spread-probability ceiling and floor</i>
------------------	---

Description

One value each, used wherever the spread model needs them. spreadProbCeiling is the upper bound fireSense_spreadFit gives maxAsymptote, and the default maxFireSpread of the objective, the gates and the validation: a fitted maxAsymptote above it draws a warning. How a year's random effect can take a pixel past it is described at maxFireSpread in [.objfunSpreadFit\(\)](#). spreadProbFloor is the default lowerSpreadProb, the link's lower asymptote, in the objective, the gates, the validation and fireSense_spreadPredict.

Usage

```
spreadProbCeiling

spreadProbFloor
```

Format

Numeric scalars.

spreadProbFromIntegerCovs

Convert covariates from their x1000 integer to usable by spread

Description

Convert covariates from their x1000 integer to usable by spread

Usage

```
spreadProbFromIntegerCovs(
  shortAnnDTx1000 = NULL,
  annDTx1000,
  nonAnnualDTx1000,
  indexNonAnnual,
  yr,
  covMinMax = NULL,
  mutuallyExclusive,
  colsToUse,
  doAssertions,
  logisticPars,
  covPars,
  maxFireSpread,
  lowerSpreadProb,
  covCentre = NULL
)
```

Arguments

shortAnnDTx1000

Optional if annDTx1000 and nonAnnualDTx1000 are supplied. Otherwise it must be a data.frame/data.table that has all covariates (≥ 1 forests, ≥ 0 non-forests, climate, youngAge), whose values are 1000x their original values so they can be stored as integers.

annDTx1000

TODO: use description of annualDTx1000 parameter in .objfunSpreadFit. If shortAnnDTx1000 is not supplied, this must be supplied. It must be a data.frame/data.table that has all the annual covariates (e.g., ≥ 1 forest biomass, ≥ 0 non forest binary), whose values are 1000x their original values so they can be stored as integers.

nonAnnualDTx1000

If shortAnnDTx1000 is not supplied, this must be supplied. A named list of data.frames/data.tables that has all the non-annual covariates (e.g., ≥ 1 forest biomass, ≥ 0 non forest binary), whose values are 1000x their original values so they can be stored as integers. These non annual covariates must have a date column that can be assessed against yr (via nonAnnualDTx1000[[whKeep]][annDTx1000, on = "pixelID"])

<code>indexNonAnnual</code>	data.table with 2 columns: <code>ind</code> and <code>date</code> which links the list elements in <code>nonAnnualDTx1000</code> with their date (in case <code>nonAnnualDTx1000</code> is not a named list. This is not necessary if the <code>nonAnnualDTx1000</code> is named with equivalent convention (e.g., character or numeric with year) as <code>yr</code>).
<code>yr</code>	A single character or numeric/integer representing the full year (e.g., 2020) to but used. This year will be extracted from both the <code>annDTx1000</code> and <code>nonAnnualDTx1000</code> if they are supplied
<code>covMinMax</code>	A data.table, one column for each column in <code>shortAnnDTx1000</code> (or the <code>ann</code> and <code>nonAnnual</code> alternatives), where the two rows represent the minimum and maximum values in the original fitting dataset. This MUST be supplied if this is prediction scenario so that the new covariates values are not scaled to themselves. I.e., they must be rescaled compared to the fitted data or else their rescaled values will be incorrect.
<code>mutuallyExclusive</code>	If there are any covariates, e.g., <code>youngAge</code> , that should be considered mutually exclusive, then this can be done here (i.e., "if <code>youngAge</code> is non-zero, should <code>vegPC2</code> be set to zero"). A named list, where the name of the list element must be a single covariate column name in either <code>annualDTx1000</code> or <code>nonAnnualDTx1000</code> . The list content should be a "grep" pattern with which to match column names, e.g., " <code>vegPC</code> ". The values of all column names that match the grep value will be set to 0, whenever the name of that list element is non-zero. Default is <code>list("youngAge" = list("vegPC"))</code> , meaning that all columns with <code>vegPC</code> in their name will be set to zero wherever <code>youngAge</code> is non-zero.
<code>colsToUse</code>	Optional. If this is supplied, it must be a character vector indicating the column names to use in <code>shortAnnDTx1000</code> , i.e., it must include everything, <code>annual</code> or <code>nonAnnual</code> , that will be used. It may start with <code>spreadInterceptTxt</code> (see spreadDesignCols()): that column is made here, as 1s, after rescaling and centring.
<code>doAssertions</code>	Logical. If TRUE, the default, the function will test a few minor things for consistency. This should be set to FALSE for operational situations, as the assertions take some small amount of time.
<code>logisticPars</code>	Numeric vector of logistic parameters. Unless <code>link</code> names the form, its length selects the variant <code>logisticAll</code> dispatches to – except that a vector with an element named <code>upperTail1</code> is always <code>logistic3pUpper</code> , whatever its length.
<code>covPars</code>	Numeric vector of covariate coefficients (same length as <code>ncol(mat)</code>); <code>mat %*% covPars</code> forms the linear predictor.
<code>maxFireSpread</code>	A value for <code>spreadProb</code> that is considered impossible to go above. Default spreadProbCeiling , also the upper bound of <code>maxAsymptote</code> . <code>maxAsymptote</code> is the spread-probability ceiling in a typical year. The year random effect (<code>yearSpreadSD</code>) is added on the logit of the final spread probability, after the link, so in a given year <code>p</code> can exceed <code>maxAsymptote</code> or fall below <code>lowerSpreadProb</code> ; that is intended. There is no absolute cap, because <code>spreadCpp</code> does not need one; <code>maxAsymptote</code> is bounded because runaway fires are slow to simulate and wasted if the parameters are wrong.

lowerSpreadProb	Numeric. Lower bound for spreadProb; if a candidate spreadProb falls at or below this, the call exits early to avoid wasted simulation work. Default spreadProbFloor .
covCentre	Optional named list or vector: covCentre[[col]] is subtracted from column col after rescaling, mutual exclusivity and the range assertion. NULL (default) does no centring. Supply the fitted values at prediction.

Value

This returns the full data.table to be used for fireSense, on the original numeric i.e., real scale, with the x1000 reversed.

spreadProbGates	<i>Would parameter sets pass the objective's spreadProb gates in its first block of years?</i>
-----------------	--

Description

Computes each parameter set's spreadProb on the covariates of the first block (the two years with the most burned area) exactly as [.objfunSpreadFit\(\)](#) does, and applies [spreadProbGateTest\(\)](#), which the objective itself calls, without simulating any fire. A parameter set that fails is one the objective would bail on in the first block. It is cheap enough to screen thousands of draws.

Usage

```
spreadProbGates(
  par,
  annualDTx1000,
  nonAnnualDTx1000,
  historicalFires,
  formulaToFit,
  covMinMax = NULL,
  mutuallyExclusive = list(youngAge = c("class", "nf")),
  covCentre = NULL,
  link = NULL,
  maxFireSpread = spreadProbCeiling,
  lowerSpreadProb = spreadProbFloor,
  lanscape1stQuantileThresh = 0.265,
  minFireSize = 2,
  escapeSizeHa = NULL,
  landscape = NULL,
  years = NULL,
  doAssertions = FALSE
)
```

Arguments

`par` a named numeric vector as `.objfunSpreadFit()` takes (logistic parameters, then the covariate coefficients, then `yearSpreadSD` if fitted), or a list of them.

`annualDTx1000`, `nonAnnualDTx1000`, `historicalFires`, `covMinMax`, `mutuallyExclusive`, `covCentre`, `link`, `maxFireSpread`, `lowerSpreadProb`, `landscape1stQuantileThresh`, `minFireSize`, `escapeSizeHa`, `landscape`
 As in `.objfunSpreadFit()` (`landscape` is needed only with `escapeSizeHa`).

`formulaToFit` character; the spread formula, as in the objective.

`years` character; the years to test. Default: the objective's first block.

`doAssertions` logical; as in the objective.

Value

for one `par`, a list of logicals `pass` (every tested year passes), `burny` and `notSpread` (any tested year fails that check), `medianOK` (every year's median is in range) and the tested years; for a list of parameter sets, a `data.frame` with one row of the same per set.

<code>spreadProbGateTest</code>	<i>The objective's spread-probability gates, on one year's <code>spreadProb</code></i>
---------------------------------	--

Description

`.objfunSpreadFit()` refuses to simulate a year (it scores the `minLik` floor, and the parameter set is "bailed") unless its `spreadProb` passes these checks. The objective calls this function, so the checks cannot differ from `spreadProbGates()`.

Usage

```
spreadProbGateTest(
  spreadProb,
  ceiling,
  lowerSpreadProb,
  maxFireSpread,
  landscape1stQuantileThresh
)
```

Arguments

`spreadProb` numeric; the `spreadProb` of each pixel of the year.

`ceiling` numeric; the logistic's asymptote (its first parameter). Pixels at or above 99% of it (or within 2.5% of `lowerSpreadProb`) are "edge" values and are left out of the spread checks.

`lowerSpreadProb`, `maxFireSpread`
 numeric; the median must be within them.

`landscape1stQuantileThresh`
 numeric; the first quartile of the non-edge values must be below this.

Value

a list: pass; burny (first quartile too high, "Too burny a landscape"); notSpread ((q90 - q10) / median <= 0.025, "Not spread out enough"); medianOK; and the nonEdgeValues and their summary that the objective logs.

stackAndExtract	<i>Prepare covariate table with ignition year, fuel class, climate value, and land cover</i>
-----------------	--

Description

Prepare covariate table with ignition year, fuel class, climate value, and land cover

Usage

```
stackAndExtract(years, fuel, LCC, climate, fires = NULL)
```

Arguments

years	character vector of fire years with FS notation e.g. "year2002"
fuel	raster brick of aggregated fuel classes, or a list of them named by years (one per year, as prepare_FuelCovsCoarseByYear() makes)
LCC	raster brick of aggregated LCC classes, or a list named by years
climate	list of raster layers named by climate variable with raster layer names matching years
fires	list of spatial points representing annual ignitions

Value

a data.frame with cell numbers, ignitions, and covariates for each year

toX1000	<i>Convert numeric values to integers x 1000</i>
---------	--

Description

This simply converts to integers times 1000 so that the values can be saved more quickly to disk, for example.

Usage

```
toX1000(lst, omitCols = "pixelID")
```

youngAgeAtYear	<i>youngAge of pixels at one fire year</i>
----------------	--

Description

Time since disturbance at year is the smaller of the data year's time since disturbance aged to year ($\text{tsd} + (\text{year} - \text{dataYear})$) and the years since the last fire before year ($\text{year} - \text{fireYear}$), taken over **all** fires in `firePixelsByYear`, not only the fires being fitted. A pixel is young when that is at or below `cutoffForYoungAge`. A pixel with NA time since disturbance (non-flammable or unknown) is never young, whether or not a fire covers it. A fire in year itself does not count: fire years are compared with the state before they burn.

Usage

```
youngAgeAtYear(
  tsd,
  dataYear,
  year,
  firePixelsByYear = list(),
  cutoffForYoungAge = fireSenseYoungAgeCutoff,
  pixelID = tsd$pixelID
)
```

Arguments

<code>tsd</code>	data.table with columns <code>pixelID</code> and <code>tsd</code> , the time since disturbance at <code>dataYear</code> (for example from <code>makeTSD()</code>).
<code>dataYear</code>	the year <code>tsd</code> describes.
<code>year</code>	the fire year.
<code>firePixelsByYear</code>	list named by fire year ("2001", or "year2001") of the <code>pixelIDs</code> that burned that year (see <code>firePixelsByYear()</code>). Years with no fire may be absent.
<code>cutoffForYoungAge</code>	Numeric. Default is 15. This is the age below which the pixel is considered "young" → <code>youngAge</code> column will be 1 if <code>age <= 15</code>
<code>pixelID</code>	integer, the pixels to return, in this order. Default: all rows of <code>tsd</code> .

Details

It works on pixel IDs, so the same call serves the buffered pixels of a spread fit and every pixel of the landscape (ignition, validation). Only fires in $(\text{year} - \text{cutoffForYoungAge}) : (\text{year} - 1)$ can make a pixel young, so only those are read.

Value

integer vector, 1 for young and 0 otherwise, one per `pixelID`.

youngAgeExclusiveCols *Columns youngAge is mutually exclusive with*

Description

youngAge is mutually exclusive with every other non-climate spread covariate: wherever it is non-zero, fuel biomass, non-forest land cover (always named nFLCC_*, see fuelClassPrep()) and treedWetland are all zero. fireSense_spreadFit derives this from the covariates it holds apart from climate (its non-annual covariate table); fireSense_spreadPredict has no such table, so it identifies fuel columns itself (their covMinMax range) and calls this for the rest, so both modules end up applying the identical rule.

Usage

```
youngAgeExclusiveCols(covNames, fuelCols = character())
```

Arguments

covNames	character vector of all covariate column names present (excluding pixelID).
fuelCols	character vector, the fuel biomass column names among covNames.

Value

a named list, list(youngAge = <matched column names>), suitable as makeMutuallyExclusive()'s mutuallyExclusiveCols argument.

Index

- * **aggregation**
 - prepare_FuelCovsCoarse, 100
 - prepare_ignitionClimate, 103
- * **caching**
 - prepare_ignitionClimate, 103
 - prepare_LightningData, 105
- * **climate**
 - prepare_ignitionClimate, 103
- * **covariates**
 - prepare_FuelCovsCoarse, 100
- * **data-preparation**
 - prepare_LightningData, 105
- * **fuel**
 - prepare_FuelCovsCoarse, 100
- * **ignition**
 - prepare_ignitionClimate, 103
- * **lightning**
 - prepare_LightningData, 105
- * **raster**
 - prepare_FuelCovsCoarse, 100
 - prepare_ignitionClimate, 103
 - prepare_LightningData, 105
- * **reproducibility**
 - prepare_ignitionClimate, 103
 - prepare_LightningData, 105
- .doDataChecks
 - (chk_duplicatedStartPixels), 24
- .objfunSpreadFit, 6
- .objfunSpreadFit(), 12, 16, 46, 57, 59, 122–124, 126, 127, 130, 131, 133, 136, 137
- abbreviateSpNames, 12
- addBufferEdge (bufferEdge), 16
- addBufferEdge(), 123
- adWeightAuto, 12
- adWeightAuto(), 10
- annualStackToDTx1000, 13
- assertCovariateRange, 14
- assessFuelClasses, 15
- assessFuelClasses(), 75
- base::scale(), 107, 115, 116
- base::setdiff(), 57
- bestByReplicatedMean
 - (rescorePopulation), 117
- bestByReplicatedMean(), 59, 123
- bufferEdge, 16
- bufferEdge(), 10
- bufferToArea, 17
- bufferToAreaRast, 18
- buildCohortBurnHistory, 19
- calcNonForestYoungAge, 20, 54
- calcYoungAge, 22
- castCohortData, 23
- chk_duplicatedStartPixels, 24
- chooseDomSecFuelClasses, 25
- chooseDomSecFuelClasses(), 53
- cleanUpSpreadFirePoints, 26
- climateCovRanges, 26
- climateData::latestHistoricalYear(), 32
- climateRasterToDataTable, 27
- clusters::clusterSetup(), 122
- coefIdentifiability (fitDiagnostics), 58
- cohortsToFuelClasses, 27, 54
- combine_fuel_classes, 29
- combinePlans, 29
- combineSpreadProbSummaries
 - (fitDiagnostics), 58
- covsX1000AndSetDF, 30
- data.table::rbindlist(), 90
- data.table::set(), 33
- dataYearElement (dataYearFor), 31
- dataYearFor, 31
- defaultFireYears, 32
- defaultFireYears(), 79

- DEoptim::DEoptim, [7](#), [9](#), [94](#), [119](#), [121](#), [122](#),
[124](#), [130](#), [139](#)
- DEoptim::DEoptim(), [124](#)
- DEoptim::DEoptim.control, [121](#)
- DEoptim::DEoptim.control(), [122](#)
- deoptimPopulationSize, [32](#)
- deoptimPopulationSize(), [123](#)
- dtReplaceNAwith0, [33](#)
- ELFFireCounts, [33](#)
- ELFFireCounts(), [34–36](#), [39](#), [69](#)
- ELFFireOverlay, [34](#)
- ELFFireOverlay(), [34](#), [40](#), [48](#), [69](#)
- ELFFitStatus, [35](#)
- ELFFitStatus(), [29](#), [38](#), [41](#), [48](#)
- ELFFlammableArea, [36](#)
- ELFFlammableArea(), [55](#)
- elfLabels, [36](#)
- elfLabels(), [79](#)
- ELFMergedName, [37](#)
- ELFMergedName(), [43](#)
- ELFMergePlan, [38](#)
- ELFMergePlan(), [29](#), [40](#), [41](#), [43](#), [44](#), [47](#), [48](#),
[79](#), [88](#), [89](#)
- ELFneighbours, [38](#)
- ELFneighbours(), [38](#), [43](#)
- ELFregionStats, [39](#)
- ELFregionStats(), [34](#), [43](#)
- ELFrunName, [40](#)
- ELFsArctic, [41](#)
- ELFsArctic(), [47](#), [79](#)
- ELFsExcluded, [41](#)
- ELFsInStudyArea, [42](#)
- ELFsizePlan, [43](#)
- ELFsizePlan(), [29](#), [39](#), [47](#)
- ELFsSkipped, [44](#)
- ELFsSkipped(), [79](#)
- ELFtemplateRaster, [44](#)
- emptySpreadCovariates, [45](#)
- EnvStats::demp(), [9](#)
- escapedFires, [46](#)
- escapedFires(), [33–35](#), [69](#)
- escapeSizePixels, [46](#)
- escapeSizePixels(), [8](#)
- escapesTxt (fireSenseUtils-constants),
[57](#)
- fewFireELFs, [47](#)
- fewFireELFs(), [79](#)
- firePixelsByYear, [48](#)
- firePixelsByYear(), [103](#), [140](#)
- fireRecordShapefile, [49](#)
- fireSenseCloudParameters, [49](#)
- fireSenseCloudParameters(), [51](#), [80](#)
- fireSenseCloudParametersMap, [50](#)
- fireSenseCloudParametersMap(), [50](#)
- fireSenseCovariatesCreate, [51](#)
- fireSenseCovariatesCreate(), [25](#), [28](#), [75](#),
[102](#), [103](#)
- fireSenseFlammabilityThreshold
(fireSenseSharedDefaults), [56](#)
- fireSenseForestedLCC
(fireSenseSharedDefaults), [56](#)
- fireSenseFuelClassCol
(fireSenseSharedDefaults), [56](#)
- fireSenseIgAggFactor
(fireSenseSharedDefaults), [56](#)
- fireSenseNonflammableLCC, [36](#), [55](#), [56](#), [82](#)
- fireSenseNonForestCanBeYoungAge
(fireSenseSharedDefaults), [56](#)
- fireSenseRunawayEdgeFrac, [11](#), [123](#)
- fireSenseRunawayEdgeFrac
(fireSenseSharedDefaults), [56](#)
- fireSenseRunawayEdgeMin, [11](#), [123](#)
- fireSenseRunawayEdgeMin
(fireSenseSharedDefaults), [56](#)
- fireSenseSCANFIVersion
(fireSenseSharedDefaults), [56](#)
- fireSenseSharedDefaults, [56](#)
- fireSenseUtils
(fireSenseUtils-package), [5](#)
- fireSenseUtils-constants, [57](#)
- fireSenseUtils-package, [5](#)
- fireSenseYoungAgeCutoff
(fireSenseSharedDefaults), [56](#)
- fitConvergence (fitDiagnostics), [58](#)
- fitDiagnostics, [58](#)
- fitLedgerFolders, [61](#)
- fixedLogisticPars, [62](#)
- fuelClassPrep, [62](#)
- fuelLinearRange, [63](#), [73](#)
- fuelLogToLinear, [63](#)
- fuelLogToLinear(), [73](#), [129](#)
- getFirePoints_NFDB, [64](#)
- getFirePoints_NFDB(), [93](#)
- getFirePoints_NFDB_V2, [65](#)
- getFirePoints_NFDB_V2(), [93](#)

- getFirePolygons, [66](#)
- getFirePolygons(), [19](#)
- harmonizeBufferAndPoints, [67](#)
- harmonizeFireData, [68](#)
- harmonizeFireData(), [69](#)
- harmonizeFireDataDeps, [69](#)
- hindcastObserved, [69](#)
- hindcastObserved(), [70](#), [71](#), [97](#), [98](#)
- hindcastSimulated, [70](#)
- identifiedInIsolation (fitDiagnostics), [58](#)
- ignitionFitAdditionalColNamesTxt
(fireSenseUtils-constants), [57](#)
- ignitionFitFilenameFor, [71](#)
- ignitionFitFilenameFor(), [61](#)
- ignitionFitFileTag
(ignitionFitFilenameFor), [71](#)
- ignitionFitLedgerFolder
(fitLedgerFolders), [61](#)
- ignitionFitLedgerFolder(), [72](#)
- ignitionsTxt
(fireSenseUtils-constants), [57](#)
- igOrEscNames, [72](#)
- isLinearFuelRange, [73](#)
- isLinearFuelRange(), [63](#)
- isYoungAge, [73](#)
- LandR::convert_SCANFI_LCC_codes(), [55](#)
- LandR::prepInputs_NTEMS_LCC_FAO(), [55](#),
[82](#), [84](#)
- LandR::prepInputs_SCANFI_LCC_FAO(), [55](#),
[57](#), [82](#), [84](#), [126](#)
- latestIgnitionFits
(ignitionFitFilenameFor), [71](#)
- latestIgnitionFits(), [61](#)
- latestNBACUrl, [74](#)
- latestNBACUrl(), [66](#)
- latestSpreadFits
(spreadFitFilenameFor), [128](#)
- latestSpreadFits(), [61](#)
- lccFlammableShare, [75](#)
- lccFlammableShare(), [15](#)
- linkSaturation (fitDiagnostics), [58](#)
- linkSaturation(), [10](#), [124](#)
- logistic2p (logistic4p), [75](#)
- logistic3p (logistic4p), [75](#)
- logistic3p(), [14](#)
- logistic3pUpper (logistic4p), [75](#)
- logistic4p, [75](#)
- logistic5p (logistic4p), [75](#)
- logisticAll (logistic4p), [75](#)
- logisticAll(), [10](#), [131](#)
- logisticParamNames, [76](#), [77](#)
- logMinB, [54](#), [77](#)
- logMinB(), [45](#), [63](#), [64](#)
- makeELFs, [78](#)
- makeELFs(), [34](#), [43](#), [47](#), [79](#), [88](#), [89](#)
- makeFireIDs, [80](#)
- makeFireSenseLCC, [81](#)
- makeFireSenseLCC(), [52](#), [55–57](#), [83](#), [84](#)
- makeFireSenseLCCDeps, [83](#)
- makeFireSenseLCCDeps(), [69](#)
- makeGLM, [84](#)
- makeLandcoverDT, [85](#)
- makeLandcoverDT(), [88](#)
- makeLociList, [85](#)
- makeMutuallyExclusive, [54](#), [86](#)
- makeMutuallyExclusive(), [53](#)
- makeRastersFromCD, [87](#)
- makeTSD, [87](#)
- makeTSD(), [48](#), [57](#), [103](#), [140](#)
- mergeELFs, [88](#)
- mergeELFs(), [43](#), [79](#)
- mergePreparedCovs, [89](#)
- multiplier, [92](#)
- multiplier(), [18](#)
- mustFit, [93](#)
- nfdbPointUrl, [93](#)
- nonNFColNamesTxt
(fireSenseUtils-constants), [57](#)
- omitNullArgs, [94](#)
- parallel::makeCluster(), [118](#)
- paramsSeparate, [94](#)
- plotBurnSummary, [95](#)
- plotCumulativeBurns, [96](#)
- plotELFs (makeELFs), [78](#)
- plotHindcastDistributions, [97](#)
- plotHindcastMap, [97](#)
- plotHindcastSizes, [98](#)
- plotHistoricFires, [99](#)
- plotSpreadFitResponse
(spreadFitValidationData), [129](#)

- plotSpreadFitValidation
 - (spreadFitValidationData), 129
- polygonIDTxt
 - (fireSenseUtils-constants), 57
- predictIgnition, 100
- prepare_FuelCovsCoarse, 100
- prepare_FuelCovsCoarse(), 102, 103
- prepare_FuelCovsCoarseByYear, 102
- prepare_FuelCovsCoarseByYear(), 138
- prepare_ignitionClimate, 103
- prepare_LightningData, 105
- prepareCovariatesOuter, 107
- prepInputs, 64, 65
- profileCoefficients (fitDiagnostics), 58
- profileCoefficients(), 123
- putBackIntoRaster, 54, 109

- rasterFireBufferDT, 111
- rasterFireSpreadPoints, 112
- rbetaBetween, 112
- readLightningData, 113
- removeBufferedFiresOutsideRTM, 114
- removeBufferedFiresOutsideRTM(), 16
- reproducable::Cache(), 45, 90, 104, 106–108
- reproducable::prepInputs(), 45, 66, 106
- reproducable::preProcess(), 71, 128
- rescaleCovariates, 115
- rescaleCovariates(), 107, 108
- rescaleKnown2, 116
- rescaleKnown2(), 63
- rescaleVarsByMagnitude, 117
- rescorePopulation, 117
- rescorePopulation(), 59
- resolveDoFit, 118
- resolveDoFit(), 93
- runDEoptim, 119
- runDEoptim(), 32, 57, 59
- runELFs, 124
- runELFs(), 41

- scoreFireSizes (fitDiagnostics), 58
- scoreFireSizes(), 124
- simInitAndSpades2, 125
- simulateFireSizes (fitDiagnostics), 58
- simulateFireSizes(), 124
- sizeRuleLandCover, 126
- SpaDES.core::paths, 121
- SpaDES.core::Plots(), 9, 122, 132

- SpaDES.tools::rastFromDF(), 102
- SpaDES.tools::spread2, 7, 130
- spreadCovCentre, 126
- spreadCovCentre(), 9, 124, 130
- spreadCovCols (spreadDesignCols), 127
- spreadCovCols(), 127
- spreadDesignCols, 127
- spreadDesignCols(), 7, 135
- spreadFitAdditionalColNames, 128
- spreadFitAdditionalColNamesTxt
 - (fireSenseUtils-constants), 57
- spreadFitCovCentreTxt
 - (fireSenseUtils-constants), 57
- spreadFitFilenameFor, 128
- spreadFitFilenameFor(), 61
- spreadFitFileTag
 - (spreadFitFilenameFor), 128
- spreadFitLedgerFolder
 - (fitLedgerFolders), 61
- spreadFitLedgerFolder(), 128
- spreadFitValidationData, 129
- spreadFitValidationData(), 10
- spreadFitWorkerMemoryGB, 132
- spreadIndicatorCols, 133
- spreadIndicatorRanges
 - (spreadIndicatorCols), 133
- spreadIndicatorRanges(), 133
- spreadInterceptTxt
 - (fireSenseUtils-constants), 57
- spreadProbBounds, 133
- spreadProbCeiling, 7, 131, 135
- spreadProbCeiling (spreadProbBounds), 133
- spreadProbFloor, 8, 131, 136
- spreadProbFloor (spreadProbBounds), 133
- spreadProbFromIntegerCovs, 134
- spreadProbGates, 136
- spreadProbGates(), 137
- spreadProbGateTest, 137
- spreadProbGateTest(), 136
- spreadProbSummary (fitDiagnostics), 58
- spreadProbSummary(), 10
- stackAndExtract, 138
- stackAndExtract(), 89, 90
- stats::is.empty.model(), 24
- stats::rbeta, 113
- stats::rbeta(), 112

- termsInDEoptim (runDEoptim), 119

terra::aggregate(), [45](#), [102](#), [104](#), [106](#)
terra::cellFromXY(), [34](#)
terra::cover(), [43](#)
terra::distance(), [43](#)
terra::mosaic(), [43](#)
terra::rast(), [110](#)
terra::res(), [46](#)
terra::setValues(), [110](#)
terra::values(), [110](#)
terra::writeRaster(), [45](#)
toX1000, [138](#)
toX1000(), [27](#)
treedWetlandAgbTxt, [52](#)
treedWetlandAgbTxt
 (fireSenseUtils-constants), [57](#)
treedWetlandTxt, [52](#)
treedWetlandTxt
 (fireSenseUtils-constants), [57](#)

updateStackYearNames, [139](#)

visualizeDE, [139](#)

yearTxt (fireSenseUtils-constants), [57](#)
youngAgeAtYear, [140](#)
youngAgeAtYear(), [22](#), [48](#), [53](#), [102](#)
youngAgeExclusiveCols, [141](#)
youngAgeTxt (fireSenseUtils-constants),
 [57](#)