Package 'rSHAPE'

July 19, 2019

Type Package

Title Simulated Haploid Asexual Population Evolution

Version 0.3.2

Author Jonathan Dench

Maintainer Jonathan Dench <jdenc017@gmail.com>

Description In silico experimental evolution offers a cost-and-time effective means to test evolutionary hypotheses. Existing evolutionary simulation tools focus on simulations in a limited experimental framework, and tend to report on only the results presumed of interest by the tools designer. The R-package for Simulated Haploid Asexual Population Evolution ('rSHAPE') addresses these concerns by implementing a robust simulation framework that outputs complete population demographic and genomic information for in silico evolving communities. Allowing more than 60 parameters to be specified, 'rSHAPE'
simulates evolution across discrete time-steps for an evolving community of haploid asexual populations with binary state genomes. These settings are for the current state of 'rSHAPE' and future steps will be to increase the breadth of evolutionary conditions permitted. At present, most effort was placed into permitting varied growth models to be simulated (such as constant size, exponential growth, and logistic growth) as well as various fitness landscape models to reflect the evolutionary landscape (e.g.: Additive, House of Cards - Stuart Kauffman and Simon Levin (1987) <doi:10.1016/S0022-5193(87)80029-2>, NK - Stuart A. Kauffman and Edward D. Weinberger (1989) <doi:10.1016/S0022-5193(89)80019-0>, Rough Mount Fuji - Neidhart, Johannes and Szendro, Ivan G and Krug, Joachim (2014) <doi:10.1534/genetics.114.167668>). This package includes numerous functions though users will only need defineSHAPE(), runSHAPE(), shapeExperiment() and summariseExperiment(). All other functions are called by these main functions and are likely only to be on interest for someone wishing to develop 'rSHAPE'. Simulation results will be stored in files which are exported to the directory referenced by the shape_workDir option (defaults to tempdir() but do change this by passing a folderpath argument for workDir when calling defineSHAPE() if you plan to make use of your results beyond your current session). 'rSHAPE' will generate numerous replicate simulations for your defined range of experimental parameters. The experiment will be built under the experimental working directory (i.e.: referenced by the option shape_workDir set using defineSHAPE()) where individual replicate simulation results will be stored as well as processed results which I have made in an effort to facilitate analyses by automating collection and processing of the potentially thousands of files which will be created. On that note, 'rSHAPE' implements a robust and flexible framework with highly detailed output at the cost of computational ef-

ficiency and potentially requiring significant disk space (generally gigabytes but up to tera-
bytes for very large simulation efforts). So, while 'rSHAPE' offers a single frame-
work in which we can simulate evolution and directly compare the impacts of a wide range of pa-
rameters, it is not as quick to run as other in silico simulation tools which focus on a single sce-
nario with limited output. There you have it, 'rSHAPE' offers you a less restrictive in silico evolu-
tionary playground than other tools and I hope you enjoy testing your hypotheses.

License GPL-3

Depends R (>= 3.2)

Imports abind, graphics, sn, VGAM, evd, stats, utils, RSQLite, DBI,
foreach, parallel, doParallel

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2019-07-19 09:40:02 UTC

R **topics documented:**

addDrift	*This is a simple little function used to represent drift by introducing stochasticity to the vector passed by making poisson distribution calls. At present it forces values to integers because I've not been able to implement an appropriate continuous distribution for such calls that works with tested models and expected outcome.*

Description

This is a simple little function used to represent drift by introducing stochasticity to the vector passed by making poisson distribution calls. At present it forces values to integers because I've not been able to implement an appropriate continuous distribution for such calls that works with tested models and expected outcome.

Usage

```
addDrift(func_inVector, func_integerValues = TRUE)
```

Arguments

```
func_inVector
```
> A vector of value to which stochasticity is to be added, integer values will be returned.

```
func_integerValues
```
> Logical toggle if a discrete or continous distribution is to be used for draws. DISABLED - as testing could not identify a continuous distribution which works for obtaining expected results from established models.

Value

A vector of values, with same length as func_inVector

Examples

```
# This adds drift by making draws from the Poisson distribution with a location parameter ba
# the elements to which drift is to be added.
replicate(10,addDrift(c(0.5,1,5,10,14.1)))
```

---

addQuotes	*This is a function to add quotation marks around each element of a character string vector*

Description

This is a function to add quotation marks around each element of a character string vector

Usage

```
addQuotes(funcIn)
```

Arguments

```
funcIn          a vector of character strings which you want padded by quotation marks
```

Value

character vector of length equal to the input

---

adjustBirths	*This function ensures that a vector of values will sum to a given number. It's implemented in certain growth forms (curently:* **constant** *and* **logistic***)*

Description

This function ensures that a vector of values will sum to a given number. It's implemented in certain growth forms (curently: **constant** and **logistic**)

Usage

```
adjustBirths(func_adjVector, func_sumTotal,
  func_roundValues = getOption("shape_track_asWhole"))
```

Arguments

func_adjVector

> Vector of values which must sum to the func_sumTotal.

func_sumTotal

> A single integer value which is to be the target summed value.

func_roundValues

> Logical toggle to control in values must be rounded to integers.

Value

A vector of values adjusted to sum to a single value. These may have been forced to be rounded or could still contain decimals.

Examples

```
# In the event we're enforcing a vector to sum to a particular value, this function will
# force that vector to the sum and adjust proportionally to elements.  You can force values
# to become integers.
adjustBirths(func_adjVector = c(9,70,20), func_sumTotal = 100, func_roundValues = FALSE)
# When rounding, this is stochastic
replicate(10,adjustBirths(func_adjVector = c(9,70,20),
                          func_sumTotal = 100,
                          func_roundValues = TRUE))
# Same idea, different input vectors
adjustBirths(func_adjVector = c(10,75,20), func_sumTotal = 100, func_roundValues = FALSE)
replicate(10,adjustBirths(func_adjVector = c(10,75,20),
                          func_sumTotal = 100,
                          func_roundValues = TRUE))
```

birthFunction	*This function calculates the number of births for the vector of populations which are expected to be passed. The number of parameters which can be passed may be more than the number required to use one of the growth forms.*

Description

This function calculates the number of births for the vector of populations which are expected to be passed. The number of parameters which can be passed may be more than the number required to use one of the growth forms.

Usage

```
birthFunction(func_inSize, func_inFitness, func_bProb, func_sizeStep,
    func_growthForm = c("logistic", "exponential", "constant", "poisson"),
    func_deaths = NULL, func_carryingCapacity = NULL,
    func_basalRate = NULL, func_deathScale = FALSE, func_drift = TRUE,
    func_roundValues = TRUE)
```

Arguments

func_inSize   This is the vector of population sizes within the community

func_inFitness

This is the vector of fitness value for the community

func_bProb   This is the general bith probability defined for this run of SHAPE

func_sizeStep

This is a proportional scalar that will control what proportion of a standard "generation" is simulated for each step within a SHAPE run. NOTE: This parameter is not perfectly validated to run as may be expected with all models. For now, it should be left as a value of "1", but exists for future implementation and testing.

func_growthForm

This is the implemeted growth model to be simulated in this run. Currently this can be one of **"logistic","exponential","constant","poisson"**.

func_deaths   This is the vector of deaths for the genotypes within the community

func_carryingCapacity

This is the maximum community size supported by tge simulated environment.

func_basalRate

This is the basal growth rate, otherwise definable as the number of offspring an individual will produce from a single birth event.

func_deathScale

This is a logical toggle to define if the number of births should be scaled by the number of deaths. The exact interpretation of this varies by growth model, but in general it forces growth to follow rates expected by standard pure birth models while still simulating deaths within the community.

func_drift This is a logical toggle as to whether or not stochasticity is introduced into the deterministic calculations that may be encountered within the growth function. Its exact implementation varies based on the growth model being simulated.

func_roundValues

  This is a logical toggle to define if the number of births and deaths are forced to be tracked as integer values. If TRUE, then any fractional amounts will be stochastically rounded to the nearest integer with a probability of being rounded up equal to the decimal value – ie: 0.32 means 32% chance of being rounded up –

Value

A vector of births with the same length as the vector of population sizes passed.

Imagine you've got an evolving community of three populations where in each time step individuals with # relateive fitness of 1 produce 2 offspring. birthFunction(func_inSize = c(100,100,100), func_inFitness = c(1,2,1.05), func_bProb = 1, func_sizeStep = 1, func_growthForm = "exponential", func_drift = FALSE) # Now with evolutionary drift birthFunction(func_inSize = c(100,100,100), func_inFitness = c(1,2,1.05), func_bProb = 1, func_sizeStep = 1, func_growthForm = "exponential", func_drift = TRUE)

---

buildPedigree	*This is a convenience script to build an named list of empty lists, where the names are based on the genotype IDs being passed.*

Description

This is a convenience script to build an named list of empty lists, where the names are based on the genotype IDs being passed.

Usage

```
buildPedigree(func_focalID)
```

Arguments

func_focalID This should be any vector, that can be interpreted as character, and faithfully represent the genotype IDs of interest for your pedigree.

Value

a named list of empty lists.

Examples

```
# this creates a named list, this trivial function exists for future flexibility and method
buildPedigree(c(1,"zebra","walrus",4))
```

calc_relativeFitness

> *This is a function to calculate the relative fitness for a vector of fitnesses. As a frame of reference it can use either an ancestral fitness value or the mean fitness of the passed vector. If the frame of reference is a value of zero - OR - the func_absDistance is set to TRUE then instead the vector is centered around a value of 1 where negative values will be set to zero.*

Description

This is a function to calculate the relative fitness for a vector of fitnesses. As a frame of reference it can use either an ancestral fitness value or the mean fitness of the passed vector. If the frame of reference is a value of zero - OR - the func_absDistance is set to TRUE then instead the vector is centered around a value of 1 where negative values will be set to zero.

Usage

```
calc_relativeFitness(func_fitVector, func_ancestFit = NULL,
  func_weights = NULL, func_absDistance = (getOption("shape_simModel")
  == "RMF"))
```

Arguments

func_fitVector

> a numeric vector of values to be interpreted as fitnesses

func_ancestFit

> An optional single numeric value to be used as a frame of reference for calculating relative fitness.

func_weights An optional vector of weights to be used for calculating relative fitness as an absolute distance from the mean of the func_fitVector vector.

func_absDistance

> A logical toggle to override if relative fitnesses are to be calculated as the absolute distance from 1. Will be overrode if either the mean of func_fitVector or func_ancestFit are zero.

Value

A vector of relative fitness values of length equal to the input vector.

Examples

```
# This calculates relative fitness values either based on the mean of the community or
# based on an ancestral fitness value.
defineSHAPE()
calc_relativeFitness(c(0.9,1,1.1))
calc_relativeFitness(c(0.9,1,1.1),func_ancestFit = 0)
```

```
calc_relativeFitness(c(0.9,1,1.1),func_ancestFit = 1)
calc_relativeFitness(c(0.95,1,1.1))
```

---

compute_distGrowth *This function is used to calculate the effect size and timing of the next stochastic population disturbance in a SHAPE run.*

Description

This function is used to calculate the effect size and timing of the next stochastic population disturbance in a SHAPE run.

Usage

```
compute_distGrowth(func_distFactor, func_growthType, func_distType,
    func_growthRate, func_popSize, func_focalSize,
    func_manualGenerations = NULL, func_stepDivs)
```

Arguments

func_distFactor

> This is the expected effect size of the disturbance, it should be a named vector with elements **factor, random** which are each used as per the func_distType

func_growthType

> This is the growth model of the SHAPE run

func_distType

> This is the type of disturbance to be simulated. Currently I've implemented **bottleneck, random** options for constant bottlenecks or normally distributed random effect sizes

func_growthRate

> This is the basal growth rate of the SHAPE run

func_popSize This is a vector of the number of individuals in each of the populations

func_focalSize

> This only matters if the growth model is exponential in which case the disturbance is always such that the community size is reduced to the func_focalSize value

func_manualGenerations

> If not NULL, it will be rounded to an integer value and taken as the manually controlled number of generations between disturbances. Otherwise, the disturbance factor and growth rate are used to estimate the number of steps required for a community with relative fitness 1 to rebound.

func_stepDivs

> This is the value that controls what proportion of a standard biological "generation" is simulated in each step of a SHAPE run.

Value

A named vector with three elements describing the simulated reduction factor of populations, the number of individuals lost, and the number of steps estimated until the next disturbance.

Examples

```
# This calculates the information for the next planned stochastic disturbance event.
# Consider a situation where there is a disturbance reducing populations 100 fold,
# and it occurs either in a proscriptive number of steps, or we calculate it based
# on recovery time as per the growth rate and growth model parameters.
compute_distGrowth("bottleneck","exponential","bottleneck",
                   2,1e4,1e2,5,1)
compute_distGrowth("bottleneck","exponential","bottleneck",
                   2,1e4,1e2,NULL,1)
# If growth is constant or Poisson, then disturbances are effectively supressed
compute_distGrowth("bottleneck","poisson","bottleneck",
                   2,1e4,1e2,NULL,1)
```

---

createGenotypes	*This function searches the nearby mutational space of a focal genotype, identifies which genotypes in that space have not yet been identified, and create new database entries for any new genotypes.*

Description

This function searches the nearby mutational space of a focal genotype, identifies which genotypes in that space have not yet been identified, and create new database entries for any new genotypes.

Usage

```
createGenotypes(tmp_focalGenotype, tmp_focalFitness, maxHamming,
  tmp_landModel = "HoC", tmp_sepString = getOption("shape_sepString"),
  tmpDirection = getOption("shape_allow_backMutations"),
  tmp_relativeFitness = getOption("shape_const_relativeFitness"),
  tmp_currNeighbours = NULL, tmp_genCon,
  tmp_tableSplit = getOption("shape_db_splitTables"),
  tmp_maxRows = getOption("shape_maxRows"),
  tmp_genomeLength = getOption("shape_genomeLength"),
  tmp_distAsS = getOption("shape_const_distAsS"), ...)
```

Arguments

```
tmp_focalGenotype
```
This is the focal genotype for which we want to create missing mutational neighbours.

```
tmp_focalFitness
```
This is the fitness value of the tmp_focalGenotype.

maxHamming   The maximum number of sites that could be changed by mutation of the tmp_focalGenotype. NOTE: At present I've not made the code work for anything other than a value of 1. So do not update without updating associated code. where appropriate.

tmp_landModel

            This is the character string that defines the fitness landscape model being simulated in this SHAPE run. At present it can be one of: **Additive, Fixed, HoC, NK, RMF**

tmp_sepString

            This is a character string used to collapse vectors of characters.

tmpDirection   This is a logical which controls if reversions are allowed (ie: if TRUE sites can revert from mutated to WT)

tmp_relativeFitness

            This is a logical which controls if fitness values are to be calculated as relative and no absolute values that would otherwise be calculated via calls to the fitness landscape model.

tmp_currNeighbours

            This is an optinal vector that would define the genotype of all neighbours within the 1 step mutational neighbourhood of the tmp_focalGenotype genotype. If NULL then this vector will be calculated within the function.

tmp_genCon   This is the filepath for the database file that contains the fitness landscape information.

tmp_tableSplit

            This is a logical which controls if the tables which report on all genotypes with X mutations should be forced into a single table or it SHAPE is allowed to split them into multiple tables.

tmp_maxRows   The maximum number of rows allowed in a database table before a new table is created. This has no meaning if tmp_tableSplit is FALSE.

tmp_genomeLength

            The length of the genomes, or number of mutable sites/positions, being simulated.

tmp_distAsS   This arugment is passed through to downstream function, but will control if the stochastic portion of fitness effect will be considered as selection coefficients (meaning subtracting 1 from the initially drawn value).

...   Additional arguments that may get passed to internal functions.

Value

This invisibly returns NULL, this function is to perform work on databases.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

---

create_genotypeFrame

> *This is a convenience function to ensure that we have a standard shaped data.frame. It is used to initiate a new table for the fitness landscape.*

Description

This is a convenience function to ensure that we have a standard shaped data.frame. It is used to initiate a new table for the fitness landscape.

Usage

```
create_genotypeFrame(tmpID, tmpStrings, tmpFitnesses)
```

Arguments

tmpID   A numeric vector of the unqiue identifiers for genotypes

tmpStrings   A vector of the character strings that represent the binary string of genotypes

tmpFitnesses   A vector of the numeric fitness values to be input

Value

A 4 column data frame with column names of genotypeID, binaryString, fitness, isExplored

Examples

```
# This is just a convenience function for outputting vectors in a data.frame with
# standard named columns.
create_genotypeFrame(c(1,10,50),c("1","1_7","6_12"),c(1,0.25,1.57))
```

---

deathFunction   *This allows SHAPE to simulate the death process as a deterministic value, and may be density dependent.*

Description

This allows SHAPE to simulate the death process as a deterministic value, and may be density dependent.

Usage

```
deathFunction(func_inSize, func_inProb = 0, func_roundValues = TRUE,
  func_depDensity = FALSE, func_densityMax = NULL,
  func_densityPower = 4)
```

Arguments

`func_inSize`	This is the vector of population sizes within the community
`func_inProb`	This is the general death probability defined for this run of SHAPE

`func_roundValues`

This is a logical toggle to define if the number of births and deaths are forced to be tracked as integer values. If TRUE, then any fractional amounts will be stochastically rounded to the nearest integer with a probability of being rounded up equal to the decimal value – ie: 0.32 means 32% chance of being rounded up –

`func_depDensity`

This is a logical toggle as to whether or not the calculation is density dependent. If TRUE, then func_densityMax reuqires a value.

`func_densityMax`

This is the community size at which maximum density dependent deaths (ie: 100% of func_inSize) occur.

`func_densityPower`

This is a scaling factor that controls the rate of transition between minimal and maximal values of the density dependent deaths. Higher values mean a steeper transition such that there are fewer deaths until higher densities are reached.

Value

A vector of the number of deaths caluclated for each of the populations represented by the func_inSize vector

Examples

```
# Imagine you've got an evolving community of three populations where in each time step
# 100% of individuals die.
deathFunction(func_inSize = c(100,50,200), func_inProb = 1)
# What if their deaths were scaled based on population density,
# or an environmental carrying capacity?
deathFunction(func_inSize = c(100,50,200), func_inProb = 1,
              func_depDensity = TRUE, func_densityMax = 400)
deathFunction(func_inSize = c(100,50,200), func_inProb = 1,
              func_depDensity = TRUE, func_densityMax = 500)
deathFunction(func_inSize = c(100,50,200), func_inProb = 1,
              func_depDensity = TRUE, func_densityMax = 350)
```

---

defineNeighbours	*The function will identify the binary string of all possible neighbours to a focal genotype. It is important when querrying the fitness landscape.*

Description

The function will identify the binary string of all possible neighbours to a focal genotype. It is important when querrying the fitness landscape.

Usage

```
defineNeighbours(func_tmpGenotype, func_tmpDirection,
  func_maxHamming = getOption("shape_max_numMutations"),
  func_sepString = getOption("shape_sepString"),
  func_genomeLength = getOption("shape_genomeLength"))
```

Arguments

`func_tmpGenotype`

> This is the binary string of the focal genotype for which we want to define possible neighbours.

`func_tmpDirection`

> This is a logical which controls if reversions are allowed (ie: if TRUE sites can revert from mutated to WT)

`func_maxHamming`

> The maximum number of sites that could be changed by mutation of the tmp_focalGenotype. NOTE: At present I've not made the code work for anything other than a value of 1. So do not update without updating associated code, where appropriate.

`func_sepString`

> This is a character string used to collapse vectors of characters.

`func_genomeLength`

> The length of the genomes, or number of mutable sites/positions, being simulated.

Value

Vector of all the genotypes in the neighbouring mutational space accessible within 1 mutation event

Examples

```
# If you had some individuals with a genome length of 10 sites, and an
# individual with no mutations, as well as one with a single mutation at
# position 7, each had a mutant.  This would define the possible one step
# mutational neighbours.  I also allow back mutations
defineNeighbours(c(""), func_tmpDirection = FALSE, func_maxHamming = 1,
                 func_sepString = "_", func_genomeLength = 10)
defineNeighbours(c("7"), func_tmpDirection = FALSE, func_maxHamming = 1,
                 func_sepString = "_", func_genomeLength = 10)
#' # Same idea, but if we allow back-mutations (ie: reversions)
defineNeighbours(c("7"), func_tmpDirection = TRUE, func_maxHamming = 1,
                 func_sepString = "_", func_genomeLength = 10)
```

`defineSHAPE`	*These are some global reference options that SHAPE will use and I consider the defaults. SHAPE parameters can be changed by calling this function and changing values OR by using the accessory SHAPE_parameters script, called in the SHAPE_runBody script. This second approach is considered more practical for building and running experiments.*

Description

These are some global reference options that SHAPE will use and I consider the defaults. SHAPE parameters can be changed by calling this function and changing values OR by using the accessory SHAPE_parameters script, called in the SHAPE_runBody script. This second approach is considered more practical for building and running experiments.

Usage

```
defineSHAPE(shape_allow_backMutations = TRUE,
  shape_collapseString = "__:__", shape_constDist = "exp",
  shape_const_relativeFitness = TRUE,
  shape_const_hoodDepth = "limited",
  shape_const_focal_popValue = 1e+05, shape_const_mutProb = 0.001,
  shape_const_distParameters = 20, shape_const_distAsS = FALSE,
  shape_const_RMF_initiDistance = 5, shape_const_RMF_theta = 0.35,
  shape_const_numInteractions = 4, shape_const_fixedFrame = NULL,
  shape_const_birthProb = 1, shape_const_deathProb = 1,
  shape_const_ancestFitness = 0, shape_const_estProp = 0.001,
  shape_const_hoodThresh = 1000, shape_const_distType = "bottleneck",
  shape_const_growthForm = "logistic", shape_const_growthRate = 2,
  shape_const_growthGenerations = NULL, shape_db_splitTables = TRUE,
  shape_death_byDensity = TRUE, shape_death_densityCorrelation = 4,
  shape_death_densityCap = NULL, shape_envString = "shapeEnvir",
  shape_externalSelfing = FALSE,
  shape_external_stopFile = "someNamed.file", shape_finalDir = NULL,
  shape_genomeLength = 100, shape_includeDrift = TRUE,
  shape_init_distPars = c(factor = 100, random = 1),
  shape_maxReplicates = 30, shape_maxRows = 2.5e+07,
  shape_muts_onlyBirths = FALSE, shape_nextID = 0,
  shape_numGenerations = 100, shape_objectStrings = c(popDemographics =
  "popDemo", repeatability = "evoRepeat"), shape_postDir = NULL,
  shape_recycle_repStart = 1, shape_results_removeSteps = TRUE,
  shape_run_isRecycling = c(Landscape = TRUE, Steps = FALSE, Parameters =
  TRUE, Neighbourhood = FALSE), shape_save_batchBase = "yourJob",
  shape_save_batchSet = 1, shape_save_batchJob = 1,
  shape_scaleGrowth_byDeaths = TRUE, shape_sepString = "_",
  shape_sepLines = "__and__", shape_serverFarm = FALSE,
  shape_simModel = "HoC", shape_size_timeStep = 1,
```

```
shape_stringsAsFactors = FALSE, shape_string_lineDescent = "_->_",
shape_string_tableNames = "numMutations", shape_thisRep = 1,
shape_tmpGenoTable = NULL,
shape_tmp_selfScript = "~/random_nullFile.txt", shape_use_sigFig = 4,
shape_toggle_forceCompletion = FALSE, shape_track_asWhole = FALSE,
shape_track_distSize = NULL, shape_workDir = NULL)
```

Arguments

shape_allow_backMutations

This is a logical toggle controlling if revertant mutants are allowed.

shape_collapseString

This is a string to collapse the progenitor and number of mutants pieces of in-
formation.

shape_constDist

This is a character string to control the distribution used for drawing fitness value
random components.

shape_const_relativeFitness

This is a logical toggle which controls if the absolute fitness values calculated
should be reinterpreted as relative fitness values.

shape_const_hoodDepth

shape_const_hoodDepth This is an object to control which strains we get deep
neighbourhood information for It should be one of **"none","limited","priority","full"**
setting this higher will cost more and more in post analysis runtime.

shape_const_focal_popValue

This is the focal population value which has different meanings based on the
growth model implemented.

shape_const_mutProb

This is the probability of a mutation event - occuring relative to the number of
mutable events - in a standard biological generation.

shape_const_distParameters

This allows a single parameter to be passed for use in the distribution of fitness
fitness effects. NOTE: you are likely going to want to pass multiple values in
which case simply set this value prior to a run's start but after loading the library.

shape_const_distAsS

This is a logical toggle controlling if fitness landscape values calculated should
be interpreted as selection coefficients rather than relative fitness values.

shape_const_RMF_initiDistance

This is the distance of the independent global fitness optima away from the WT
genotype. It matters for the Rough Mount Fuji landscapes.

shape_const_RMF_theta

This is the Rough Mount Fuji value that controls the scalar of the independent
fitness contribution.

shape_const_numInteractions

This is the number of sites which interact with respect to fitness calculations in
models such as the NK.

shape_const_fixedFrame

    This defines the fitness landscape when our model is "Fixed", it must be user defined and be explicit to all genotypes possible.

shape_const_birthProb

    This is the proportion of individuals with fitness == 1 having births events in a standard biological generation.

shape_const_deathProb

    This is the proportion of individuals having a death event in a standard biological generation.

shape_const_ancestFitness

    This is the fitness value of the ancestral genotype.

shape_const_estProp

    This is the value controlling when SHAPE considers a population to be established.

shape_const_hoodThresh

    This is the numeric value controlling when a population is of sufficient size for SHAPE to consider it worth having the genotype's mutational neighbourhood to be stored in a convenience DB for easier access - ie: this can save computational time but will cost diskspace during the run.

shape_const_distType

    This is the type of stochastic disturbance events to be simulated.

shape_const_growthForm

    This is the growth form model to be simulated

shape_const_growthRate

    This is the number of offspring from every division event where 1 would mean replacement, 2 is normal binary fission, etc....

shape_const_growthGenerations

    This is an optional integer value controlling if you want a standard number of time steps between each stochastic disturbance function call. Not defining this means it will be calculated based on other paratmerts defined.

shape_db_splitTables

    This is a logical toggle as to whether or not fitness landscape tables - for genotypes with the same number of mutations - are allowed to be split into sub-tables.

shape_death_byDensity

    This is the logical toggle controlling if deaths are density dependent.

shape_death_densityCorrelation

    This is a positive numeric controlling the rate at which density dependent deaths increase from minimal to maximal effect. Where 1 is linear, > 1 creates an exponential form of curve and values < 1 will create a root function curve.

shape_death_densityCap

    If deaths are density dependent this is the maximal community size for when deaths are 100% expected.

shape_envString

    This is a string used for programatically creating workspace environments for rSHAPE

shape_externalSelfing

This is the logical toggle controlling if replicates are to be handled as individual external calls rather than through the normal internal for loop. It has limited value and was desgined for when you work on compute nodes with limited wall time.

shape_external_stopFile

This is the filename for a file which is used to control self-replciation of SHAPE when selfing is external.

shape_finalDir

This is the directory where file from a remote server's compute node are to be back ported regularly. Only matters under the correct conditions.

shape_genomeLength

This is the length of a simulant's genome, or in other words the number of sites where mutations can occur.

shape_includeDrift

This is a logical toggle as to whether or not we should add stochasticity to the growth function calculations. It is meant to simulate drift in calculations that would otherwise be deterministic.

shape_init_distPars

This is the vector of initial values of the dilution factor and random component of the stochastic disturbance function. It needs to be set with a number and range of values approriate to the distribution to be simulated.

shape_maxReplicates

This is the number of replicates to be run.

shape_maxRows

This is the integer number of rows stored in a single table of the fitness landscape DB. Only matters is tables are aplit/

shape_muts_onlyBirths

This is a logical flag to control if mutants only appear as a result of birth events.

shape_nextID This is the next genotype ID to be assigned for a genotype that get's created.

shape_numGenerations

This is the number of generations to be simulated in the run.

shape_objectStrings

This is a named character vector which are the string prefixes used when pro-gramatically naming objects.

shape_postDir

This is the filepath to the directory where post-analysis results will be stored.

shape_recycle_repStart

This is the first replicate being simulated once a SHAPE call is made.

shape_results_removeSteps

This is a logical flag controlling if the steps log is removed after being processed.

shape_run_isRecycling

This is a named vector of four logicals which control which parts of a run is meant to be recycled between replicates.

shape_save_batchBase

This is a character string for naming your experiment.

shape_save_batchSet

>   This is an integer value for the set of this experiment associated to this job.

shape_save_batchJob

>   This is an integer value for the batch of this experiment associated to this job.

shape_scaleGrowth_byDeaths

>   This is a logical flag that controls if growth is scaled by deaths so that the growth form follows standard expectations.

shape_sepString

>   This is a string character that is used for collpasing vectors of information into a single character string, and subsequently splitting that information back out.

shape_sepLines

>   This is a string character that is used in collapsing multiple elements into a single character string though namely employed in the summariseExperiment function.

shape_serverFarm

>   This is a logical flag of whether or not your simulations are going to be run on a remote server or other situation with compute and host nodes where you might want to handle particularities I experienced and thus accounted for.

shape_simModel

>   This is the fitness landscape model to be simulated.

shape_size_timeStep

>   This is the proportion of a standard biological generation to be simulated in a single time step of a SHAPE run. Values greater than 1 are not guaranteed to work as expected. Negative numbers will cause errors.

shape_stringsAsFactors

>   I don't like strings to be factors and so SHAPE will avoid treating them as so.

shape_string_lineDescent

>   This is a string that will be used to collapse vectors of character strings into a single string It get's used when we are tracking sequential genotypes through the line of descent

shape_string_tableNames

>   This is a string value used as the prefix when naming table in the fitness landscape DB.

shape_thisRep

>   This is the replicate number of the first replicate processed in the called run.

shape_tmpGenoTable

>   This is a temporary object of a table of genotype information that is to be passed along different functions of SHAPE. It's stored as an option since it can be build within a function where it is not returned as an object but then used later. There is little value in setting this manually.

shape_tmp_selfScript

>   This is an optionally defined filepath location for a file that will exist to signal that an externally replicating SHAPE run can stop. This only matters if selfing is external.

shape_use_sigFig

>   This is the number of significant figures that will be kept for processed output.

```
shape_toggle_forceCompletion
```
    This is a logical toggle controlling if a run crashes when it is ended prior to the
    maximum number of replicates being completed.

```
shape_track_asWhole
```
    This is a logical toggle controlling if population sizes must be tracked as integer
    values

```
shape_track_distSize
```
    This is a numeric, the size of a disturbance caused by stochastic events. It is the
    dilution factor or the divisor of the community size. It must be > 1 or is forced
    to that value.

```
shape_workDir
```
    This is the main working directory relative to which your SHAPE experiment
    will be built and run. It defaults to the – tempdir – of R when this value is NULL,
    I strongly recommend

Warning

Please pass a directory filepath to the argument of shape_workDir, rSHAPE will create this so it
needn't exist yet. If you leave it as the default – ie NULL – whatever is created will simply be lost
in the temporary folder of this R sessions' workspace.

Examples

```
# This function builds the basic parameters for a run of SHAPE and I recommend as
# the most convenient wayfor setting your own parameters since this function will
# make appropriate derived settings based on values passed.
# You must at least call it before using runSHAPE() or shapeExperiment().

# You can see there are a lot of parameters for SHAPE
args(defineSHAPE)
# Here are some default values that were just loaded as options
sapply(c("shape_workDir","shape_save_batchJob","shape_save_batchBase", "shape_simModel"),get
# As an exmaple we change your working directory, the ID of the job and the fitness landscap
options(list("shape_workDir" = paste(tempdir(),"~/alternativeFolder/",sep=""),
                "shape_save_batchJob" = 3, "shape_save_batchBase" = "non_default_Experiment"
                "shape_simModel" = "NK"))
sapply(c("shape_workDir","shape_save_batchJob","shape_save_batchBase", "shape_simModel"),get
# NOTE: that manually setting the options will not create a new working directory for rSHAPE
# you would need to do this yourself or could simply pass these arguments through a call
# to defineSHAPE().
```

---

 expGrowth   *This function uses the exponential growth model and can either cal-*
           culated the expected growth for a single time step OR it can work
           backwards to calculated what was the expected starting population
           size prior to a step of exponential growth.

Description

This function uses the exponential growth model and can either calculated the expected growth for a single time step OR it can work backwards to calculated what was the expected starting population size prior to a step of exponential growth.

Usage

```
expGrowth(func_rate, func_step, func_startPop = NULL,
  func_endPop = NULL)
```

Arguments

func_rate   This is the number of offpsring expected to be produced by an individual. When calculating the expected population size after a time step, we force this rate to be no less than 1 since this function has meaning only in the birth function and so we do not want to calculate negative births (which would mean deaths).

func_step   This is a proportional scalar that will control what proportion of a standard "generation" is simulated for each step within a SHAPE run. NOTE: This parameter is not perfectly validated to run as may be expected with all models. For now, it should be left as a value of "1", but exists for future implementation and testing.

func_startPop
              This is the initial population size(s) for which you want to calculate a final size. Leave NULL if trying to calculated the expected initial size from a final population.

func_endPop   This is the final population size(s) for which you want to calculate a initial size. Leave NULL if trying to calculated the expected final size from an initial population.

Value

numeric value

Examples

```
# Exponential growth equation implemented but allowing either the final or initial populatio
# to be calculated based on whethere the initial or final community size is input.
expGrowth(func_rate = 2, func_step = 1,func_startPop = 100)
expGrowth(func_rate = 2, func_step = 1,func_endPop = 200)
expGrowth(func_rate = 2, func_step = 7,func_startPop = 100)
# You cannot set a growth rate less than 1 as this would then simulate deaths which is not
# allowed in this calculation.
expGrowth(func_rate = c(0.9,1,1.1), func_step = 1,func_startPop = 100)
```

```
extractInfo_focalID
```
> *This is a function to extract genotype/lineage specific information.*
> *This info will be mostly through time style of information but will also*
> *include information about it's line of descent, growth pressures pre-*
> *establishment, and population size.*

Description

This is a function to extract genotype/lineage specific information. This info will be mostly through
time style of information but will also include information about it's line of descent, growth pressures pre-establishment, and population size.

Usage

```
extractInfo_focalID(func_focalID, func_estValue, func_stepsCon,
  func_landscapeCon, func_hoodCon, func_refMatrix, func_subNaming,
  func_genomeLength = getOption("shape_genomeLength"),
  func_max_numMutations = getOption("shape_max_numMutations"),
  func_allow_backMutations = getOption("shape_allow_backMutations"),
  func_descentSep = getOption("shape_string_lineDescent"),
  func_hoodExplore = getOption("shape_const_hoodDepth"),
  func_stringSep = getOption("shape_sepString"))
```

Arguments

func_focalID   This is the vector of genotype ID(s) of the focal lineage(s) for which information
                is to be extracted.

func_estValue
                This value is used to define the threshold size required for a population before it
                is considered established.

func_stepsCon
                This is the filepath to an SQLite database storing information for the stepwise
                changes of a SHAPE run.

func_landscapeCon
                This is the filepath to an SQLite database storing information for the complete
                explored and neighbouring fitness landscape of a SHAPE run.

func_hoodCon   This is the filepath to an SQLite database storing information for high priority
                mutational neighbourhood information

func_refMatrix
                Is a matrix of a SHAPE run's population demographics at a step in time. I will
                be querried for information regarding a genotype's number of mutations and
                fitness value. of genotypes, but is not required but is also required

func_subNaming
                This is a logical which controls if the tables which report on all genotypes with
                X mutations should be forced into a single table or it SHAPE is allowed to split
                them into multiple tables.

func_genomeLength

> The number of positions simulated within the individual's genomes.

func_max_numMutations

> The maximum number of mutations that could occur in a single mutation event – CAUTION: This should never be anything other than 1 as per how SHAPE is currently implemented.

func_allow_backMutations

> This is a logical toggle controlling if reversions are allowed – meaning loss of mutations.

func_descentSep

> This is the standard string used to collapse line of descent information.

func_hoodExplore

> This is an object to control which strains we get deep neighbourhood information for It should be one of **"none","limited","priority","full"** setting this higher will cost more and more in post analysis runtime. NOTE: That use of **limited** requires that you pass a func_refMatrix of expected shape (has a "genotypeID" column)!

func_stringSep

> A common string separator used to merge information.

Value

This returns a list object with several pieces of summary information for the focal genotype ID.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

---

extract_popDemographics

> *This is a function that steps forward through time steps of a SHAPE run and extracts population demographic information. This includes Fitness, Number of Lineages, and Transitions between dominant genotypes. Most important it will also return the information related to which lineages will eventually establish in the population, a piece of information that will be critical for downstream lineage specific information extraction.*

Description

This is a function that steps forward through time steps of a SHAPE run and extracts population demographic information. This includes Fitness, Number of Lineages, and Transitions between dominant genotypes. Most important it will also return the information related to which lineages will eventually establish in the population, a piece of information that will be critical for downstream lineage specific information extraction.

Usage

```
extract_popDemographics(func_stepsCon, func_estValue, func_landscapeCon,
    func_hoodCon, func_size_timeStep)
```

Arguments

func_stepsCon

> This is the filepath to an SQLite database storing information for the stepwise changes of a SHAPE run.

func_estValue

> This value is used to define the threshold size required for a population before it is considered established.

func_landscapeCon

> This is the filepath to an SQLite database storing information for the complete explored and neighbouring fitness landscape of a SHAPE run.

func_hoodCon This is the filepath to an SQLite database storing information for high priority mutational neighbourhood information (which is simply a subset of the full mutational landscape).

func_size_timeStep

> This is the proportion of a standard biological generation which is to be simulated in a single time step.

Value

This return a list object that contains various pieces of usefull summary demographic information.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

---

findParent   *This function will look through a pedigree data.frame and recursively*
                        continue building that back through the history of the SHAPE run be-
                        ing processed.

Description

This function will look through a pedigree data.frame and recursively continue building that back through the history of the SHAPE run being processed.

Usage

```
findParent(func_focalGenotype, func_startStep, func_stepMatrix,
    func_progenitorList, func_demoArray, func_pedigreeAll,
    func_lineString = getOption("shape_string_lineDescent"))
```

Arguments

`func_focalGenotype`

a vector of genotype IDs whose lineage you wish to identify.

`func_startStep`

this is the first step in the SHAPE run from which you wish to consider re-tracing the lineage.

`func_stepMatrix`

this is the matrix that represent what happened at each step in the SHAPE run.

`func_progenitorList`

this is a list of the known progenitor(s) for our func_focalGenotypes

`func_demoArray`

this is the whole array of step-wise SHAPE records for population demographics and feeds func_stepMatrix.

`func_pedigreeAll`

this is a data.frame which contains all currently known pedigree information and informs our step-wise focus.

`func_lineString`

this is the string that will be used to collapse the vector of progenitor genotype's into a single charater string. This collapse is done as a convenience for storage and retrieval.

Value

a vector of character strings, each of which is the found lineage of the func_focalGenotypes

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

`find_neededNeighbours`

This function querries if a suite of genotypes exist within the fitness landscape database.

Description

This function querries if a suite of genotypes exist within the fitness landscape database.

Usage

```
find_neededNeighbours(tmp_possibleNeighbours, tmp_focal_numMuts,
  tmp_refTables, maxHamming = getOption("shape_max_numMutations"),
  tmp_tableSplit = getOption("shape_db_splitTables"),
  tmp_genomeLength = getOption("shape_genomeLength"),
  tmpDirection = getOption("shape_allow_backMutations"),
  tmpRange_numMuts = NULL, tmp_genCon)
```

Arguments

tmp_possibleNeighbours

> This is a vector of all possible mutants that we're trying to querry within the fitness landscape database.

tmp_focal_numMuts

> This is the number of mutations in the focal genotype, it controls - along with other parameters - what tables of the fitness landscape database are querried.

tmp_refTables

> This is the a vector of named tables that exist within the fitness landscape. It can not be passed in which case the database at tmp_genCon is querried for this information.

maxHamming   The maximum number of sites that could be changed by mutation of the tmp_focalGenotype.

tmp_tableSplit

> This is a logical which controls if the tables which report on all genotypes with X mutations should be forced into a single table or it SHAPE is allowed to split them into multiple tables.

tmp_genomeLength

> The length of the genomes, or number of mutable sites/positions, being simulated.

tmpDirection   This is a logical which controls if reversions are allowed (ie: if TRUE sites can revert from mutated to WT)

tmpRange_numMuts

> This is the range of number of mutations which a mutant neighbour may posses. If not supplied that will be calculated in line via other parameters passed to the function.

tmp_genCon   This is the filepath for the database file that contains the fitness landscape information.

Value

A vector of the genotypes that need to be created as they've not yet been defined within the fitness landscape.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

---

fitnessDist   *This is the function that will call for draws from distributions.*

Description

This is the function that will call for draws from distributions.

Usage

```
fitnessDist(tmpDraws, tmpDistribution, tmpParameters)
```

Arguments

tmpDraws   This is the number of draws sought from the distribution being called

tmpDistribution

> This is the character string that represents the implemented distribution you want called. It must be one of: **Fixed, Gamma, Uniform, Normal, Chi2, beta, exp, evd, rweibull, frechet, skewNorm**

tmpParameters

> This is the ordered vector of parameters to be passed in order to parameterise the distribution from which you want to draw

Value

A vector of values with length equal to tmpDraws

Examples

```
# This draws from distributions
fitnessDist(10, "Uniform", c(0,1))
fitnessDist(10, "Normal", c(0,1))
fitnessDist(10, "exp", 1)
```

---

fitnessLandscape   *This function will calculate the fitness values for genotypes being newly recorded to the fitness landscape.*

Description

This function will calculate the fitness values for genotypes being newly recorded to the fitness landscape.

Usage

```
fitnessLandscape(tmpGenotypes, tmp_focalFitness, landscapeModel = "HoC",
  tmp_ancestralFitness = getOption("shape_const_ancestFitness"),
  tmp_weightsRMF = getOption("shape_const_RMF_theta"),
  tmp_optimaRMF = getOption("shape_const_RMF_globalOptima"),
  tmp_correlationsNK = getOption("shape_const_NK_interactionMat"),
  tmp_const_numInteractionsNK = getOption("shape_const_numInteractions"),
  tmp_NK_ancestDep = getOption("shape_const_DepbySite_ancestFitness"),
  relativeFitness = TRUE,
  func_genomeLength = getOption("shape_genomeLength"),
  func_distribution = getOption("shape_constDist"),
```

```
func_distParameters = getOption("shape_const_distParameters"),
func_distAsS = getOption("shape_const_distAsS"),
func_sepString = getOption("shape_sepString"))
```

Arguments

tmpGenotypes   This is a vector of the binaryString values that represent the genotype(s) for
which you want to calculate new fitness values.

tmp_focalFitness

This argument has different meaning depending upon the fitness landscape model
being simulated. It can be a vector of fitness values, a matrix, a single value,
etc...

landscapeModel

This is the character string that defines the fitness landscape model being sim-
ulated in this SHAPE run. At present it can be one of: **Additive, Fixed, HoC,
NK, RMF**

tmp_ancestralFitness

This is the fitness value of the pure WT genotype, it does not always have mean-
ing.

tmp_weightsRMF

This is the weighting of the constant/deterministic term calculated in the RMF
fitness landscape equation.

tmp_optimaRMF

This is the binary string genotype of the optimal genotype in the current RMF
fitness landscape. It needn't yet have been yet explored, it is simply the genotype
that will be the deterministic global optimum.

tmp_correlationsNK

This is the matrix of fitness values and interactions between mutational states
for the NK fitness lanscape model

tmp_const_numInteractionsNK

This is the "K" value of the NK fitness landscape value and represents the num-
ber of other sites correlated to the fitness of a focal site.

tmp_NK_ancestDep

This is the fitness value of the WT mutant for an NK fitness landscape, it is
passed as a computational ease so that it needn't be calculated each time this
function is called.

relativeFitness

This is a logical toggle controlling if the fitness values returned should be relative
fitness values

func_genomeLength

This is the genome length of individuals.

func_distribution

This is a character string representing which of the allowed distribution func-
tions can be called for draws of stochastic values when calculating fitness values.
See fitnessDist for those implemented.

func_distParameters

This is a vector of the ordered distribution parameters expected by the distribu-
tion referenced by func_distribution

func_distAsS This is a logical toggle to control in the final returned values should be considered as selection coefficients, which is achieved by subtracting the calculated value by 1.

func_sepString

This is a character string used for collapsing vectors of information, and expanding the collpased information back into a vector of values.

Value

A vector of fitness values to be assgined for each of the newly explored genotypes defined in the vector tmpGenotypes

Note

There is no example as this does not have meaning outisde of a runSHAPE call.

---

growthFunction *This is a wrapper function where the birth and death related parameters of a SHAPE run are passed before the appropriate functions (and their associated methods) are called. This function will be called once per time step of a SHAPE run.*

Description

This is a wrapper function where the birth and death related parameters of a SHAPE run are passed before the appropriate functions (and their associated methods) are called. This function will be called once per time step of a SHAPE run.

Usage

```
growthFunction(func_inSize, func_inFitness, func_bProb, func_dProb,
  func_deathDen_logical = FALSE, func_deathDen_max = NULL,
  func_deathDen_power = 4, func_sizeStep,
  func_growthForm = c("logistic", "exponential", "constant", "poisson"),
  func_carryingCapacity = NULL, func_basalRate = NULL,
  func_deathScale = FALSE, func_drift = TRUE,
  func_roundValues = FALSE, func_inIDs = NULL)
```

Arguments

func_inSize This is the vector of population sizes within the community

func_inFitness

This is the vector of fitness value for the community

func_bProb This is the general bith probability defined for this run of SHAPE

func_dProb This is the general death probability defined for this run of SHAPE

func_deathDen_logical

This is a logical toggle to define if deaths are calculated in a density dependent manner.

func_deathDen_max

This is the community size at which maximum density dependent deaths (ie: 100% of func_inSize) occur.

func_deathDen_power

This is a scaling factor that controls the rate of transition between minimal and maximal values of the density dependent deaths. Higher values mean a steeper transition such that there are fewer deaths until higher densities are reached.

func_sizeStep

This is a proportional scalar that will control what proportion of a standard "generation" is simulated for each step within a SHAPE run. NOTE: This parameter is not perfectly validated to run as may be expected with all models. For now, it should be left as a value of "1", but exists for future implementation and testing.

func_growthForm

This is the implemeted growth model to be simulated in this run. Currently this can be one of **"logistic","exponential","constant","poisson"**.

func_carryingCapacity

This is the maximum community size supported by tge simulated environment.

func_basalRate

This is the basal growth rate, otherwise definable as the number of offspring an individual will produce from a single birth event.

func_deathScale

This is a logical toggle to define if the number of births should be scaled by the number of deaths. The exact interpretation of this varies by growth model, but in general it forces growth to follow rates expected by standard pure birth models while still simulating deaths within the community.

func_drift   This is a logical toggle as to whether or not stochasticity is introduced into the deterministic calculations that may be encountered within the growth function. Its exact implementation varies based on the growth model being simulated.

func_roundValues

This is a logical toggle to define if the number of births and deaths are forced to be tracked as integer values. If TRUE, then any fractional amounts will be stochastically rounded to the nearest integer with a probability of being rounded up equal to the decimal value – ie: 0.32 means 32% chance of being rounded up –

func_inIDs   This is a vector of the genotype IDs passed to this function, its order should be representative of the ordered genotypeIDs passed for func_inSize and func_inFitness.

Value

A 2 column matrix of numeric values with columns "births" and "deaths", and rownames equal to func_inIDs (as.character).

Examples

```
# Imagine you've got an evolving community of three populations where
# in each time step 100% of individuals die and individuals with relateive
# fitness of 1 produce 2 offspring.  This growth function calculates the births
# and deaths of that community.
# First I show you when births are deterministic (proof of implementation):
growthFunction(func_inSize = c(100,100,100), func_inFitness = c(1,2,1.05),
                  func_bProb = 1, func_dProb = 1,
                  func_sizeStep = 1, func_growthForm = "exponential",
                  func_drift = FALSE, func_deathScale = TRUE)
# Now same things but with evolutionary drift thrown in
growthFunction(func_inSize = c(100,100,100), func_inFitness = c(1,2,1.05),
                  func_bProb = 1, func_dProb = 1, func_sizeStep = 1,
                  func_growthForm = "exponential", func_drift = TRUE,
                  func_deathScale = TRUE)
# Now technically the values in the birth column is really the net population
# size and I'd previously set the births to be scaled by deaths but if this were
# not the case you'd get final population sizes of:
growthFunction(func_inSize = c(100,100,100), func_inFitness = c(1,2,1.05),
                  func_bProb = 1, func_dProb = 1, func_sizeStep = 1,
                  func_growthForm = "exponential", func_drift = TRUE,
                  func_deathScale = FALSE)
```

---

logisticGrowth	*This function is simply an implementation of the logistic growth equation where: f(x) = K / (1 + ((K - N_0)/N_0) *exp-k(x-x_0)) ; Where x_0 is an adjustment to the position of the midpoint of the curve's maximum value K = the curves maximum value, k = the steepness of the curve (growth rate), and N_0 is the starting population it includes parameters to change the midpoint as well as change the natural exponent (ie: exp) to some other value. NOTE: This is for continuous growth, and since SHAPE is discrete at present this is an unused function.*

Description

This function is simply an implementation of the logistic growth equation where: $f(x) = K / (1 + ((K - N_0)/N_0) *exp-k(x-x_0))$; Where x_0 is an adjustment to the position of the midpoint of the curve's maximum value K = the curves maximum value, k = the steepness of the curve (growth rate), and N_0 is the starting population it includes parameters to change the midpoint as well as change the natural exponent (ie: exp) to some other value. NOTE: This is for continuous growth, and since SHAPE is discrete at present this is an unused function.

Usage

```
logisticGrowth(func_rate, func_step, func_startPop = NULL,
  func_maxPop = NULL, func_midAdjust = 0,
  func_basalExponent = exp(1))
```

Arguments

`func_rate`	The basal growth rate of individuals in the SHAPE run.
`func_step`	This is the number of steps forward for which you wish to calculate the growth expected.
`func_startPop`	
	The sum of the populations in the evolving community.
`func_maxPop`	The carrying capacity of the enviroment being simulated.
`func_midAdjust`	
	The midpoint which controls the point of inflection for the logistic equation. Beware, change this at your own risk as its impact will varrying based on the population sizes being simulated. Ideally, don't change this value from its default.
`func_basalExponent`	
	This defaults as the natural exponent "e" / "exp". Change it at your own risk.

Value

Returns a single value representing the amount of logistic growth expected by the community

Examples

```
# This calculates logistic growth based on the mathematical continuous time algorithm
logisticGrowth(func_rate = 2, func_step = 1, func_startPop = 1e2, func_maxPop = 1e4)
# It normally takes log2(D) steps for a binary fission population to reach carrying capacity
# where D is max/start, in this case D = 100 and so it should take ~ 6.64 turns
logisticGrowth(func_rate = 2, func_step = c(1,2,3,6,6.64,7), func_startPop = 1e2, func_maxPo
```

`logisticMap`	*This is the discrete time logistic growth function known as the logistic map. It calculates the amount of growth expected in a step of time given by: $N\_t+1 = N\_t + r * (N\_t (K - N\_t)/K)$; where N_t is community size at a time point, r is the per step growth rate, and K is the environmental carrying capacity.*

Description

This is the discrete time logistic growth function known as the logistic map. It calculates the amount of growth expected in a step of time given by: $N\_t+1 = N\_t + r * (N\_t (K - N\_t)/K)$; where N_t is community size at a time point, r is the per step growth rate, and K is the environmental carrying capacity.

Usage

```
logisticMap(func_rate, func_startPop, func_maxPop)
```

Arguments

func_rate   Per time step intrinsic growth rate of individuals

func_startPop

> The initial summed size of the evolving community

func_maxPop   The carrying capacity of the simulated environment

Value

A single value as to the expected summed size of evolving populations in the considered environment.

Examples

```
# This is the discrete time step form of the logistic equation, known as the logistic map.
# It takes a growth rate starting and max possible community size.
stepwise_Size <- 100
for(thisStep in 1:7){
  stepwise_Size <- c(stepwise_Size,
                      logisticMap(2,stepwise_Size[length(stepwise_Size)],1e4))
}
stepwise_Size
# When a population overshoots, it will loose members.
```

---

lossSampling   *This function actually calculates the stochastic loss to populations.*

Description

This function actually calculates the stochastic loss to populations.

Usage

```
lossSampling(func_inPopulation, func_dilutionFactor)
```

Arguments

func_inPopulation

> This is a vector of the number of individuals in the populations within the community.

func_dilutionFactor

> This is expected proportion of the current population sizes that should remain.

Value

A vector of the resultant population sizes remaining.

Examples

```
# A vector of population sizes is randomly sampled to be around the product of size and fact
replicate(5,lossSampling(c(1e4,2e4,3e4),0.01))
```

---

mutationFunction	*This allows SHAPE to simulate the mutation process as a deterministic value. At present, values must be tracked as integer results for reasons of how I am passing to functions which identify what mutant genotype(s) are created.*

Description

This allows SHAPE to simulate the mutation process as a deterministic value. At present, values must be tracked as integer results for reasons of how I am passing to functions which identify what mutant genotype(s) are created.

Usage

```
mutationFunction(func_inSize, func_inProb = 0)
```

Arguments

func_inSize	This is the vector of the population sizes, or perhaps number of births, or sum of both, within the community. Which vector gets passed will depend on which growth form and other parameters are being implemented by SHAPE.
func_inProb	This is the general mutation rate (probability) defined for this run of SHAPE. It is a per individual considered value, by which I mean that each mutant will have a single new mutation (or reversion if allowed - handled elsewhere) and so this probability is based on the vector of individuals passed and any context of if it is a "per generation" value relates to how time steps and birth probabilities are handled in the run.

Value

A vector of the number of mutants produced by each of the populations represented by the func_inSize vector

Examples

```
# The number of mutants generated is forcibly integer but is based
# on the stochastic rounding of the product of the number of potentially
# mutable individuals and their probability of mutation.
mutationFunction(c(10,50,100),func_inProb = 0.3)
replicate(5,mutationFunction(c(10,50,100),func_inProb = 0.35))
```

---

nameEnviron	*This quick little function is a means for me to create the strings of environments and subsequently extract information back out.*

Description

This quick little function is a means for me to create the strings of environments and subsequently extract information back out.

Usage

```
nameEnviron(func_Index, funcSplit = FALSE,
  funcBase = getOption("shape_envString"))
```

Arguments

func_Index	This is the vector of numeric, or otherwise unique ID values for the environments to be created. Or if funcSplit == TRUE, then these are the names to be split.
funcSplit	A logical toggle of whether you are building or splitting the name
funcBase	This is the character string used as a prefix to identify environment objects

Value

A vector of character string of length equal to input.

Examples

```
# Returns a standard named string
test_envNames <- nameEnviron(1:10)
nameEnviron(test_envNames, funcSplit = TRUE)
```

---

nameObject	*This quick little function is a means for me to create the strings of environments and subsequently extract information back out.*

Description

This quick little function is a means for me to create the strings of environments and subsequently extract information back out.

Usage

```
nameObject(func_inString, func_inPrefix, func_splitStr = FALSE)
```

Arguments

`func_inString`

>> This is the vector of numeric, or otherwise unique ID values for the environments to be created. Or if funcSplit == TRUE, then these are the names to be split.

`func_inPrefix`

>> This is the character string used as a prefix to identify environment objects

`func_splitStr`

>> A logical toggle of whether you are building or splitting the name

Value

A vector of character string of length equal to input.

Examples

```
# Returns a standard named string
test_objectNames <- nameObject(1:10, "testObject")
nameObject(test_objectNames, "testObject", func_splitStr = TRUE)
```

| nameTable | *This is a standardising function which allows SHAPE to programatiically name tables for the fitness landscape OR split a named table and extract the embedded information from its naming.* |

Description

This is a standardising function which allows SHAPE to programatiically name tables for the fitness landscape OR split a named table and extract the embedded information from its naming.

Usage

```
nameTable(func_tmpMutations, func_tmpIndex = NULL,
  func_baseString = getOption("shape_string_tableNames"),
  func_sepString = getOption("shape_sepString"),
  func_splitName = FALSE,
  func_subNaming = getOption("shape_db_splitTables"))
```

Arguments

`func_tmpMutations`

>> Integer value(s) for the number of mutations to be expected in mutants stored within the named tables.

`func_tmpIndex`

>> An optinal element that will be used to insert a unique vector ID

`func_baseString`

>> This is the standard prefix character string used in table naming.

```
func_sepString
```
> This is a character string used to collapse vectors of characters.

```
func_splitName
```
> A logical toggle to control if this function is splitting a named table or not. So, FALSE (default) means we're creating a table name whereas TRUE is splitting a named table into it's parts.

```
func_subNaming
```
> This is a logical which controls if the tables which report on all genotypes with X mutations should be forced into a single table or it SHAPE is allowed to split them into multiple tables.

Value

If func_splitName is TRUE, then a vector of table names is returned, it would be best practice to not assume recycling of passed elements and so pass equally lengthed vectors as input. If FALSE, we split the table and return the data detailing the number of mutations which ought to be present for genotypes stored in the named table.

Examples

```
# This creates a table name in a standard way, it can also split table names to extract info
defineSHAPE()
nameTable(2,1,"myTest","_",FALSE,FALSE)
nameTable("myTest_2",func_splitName = TRUE)
```

```
nameTable_neighbourhood
```
> *This is a standardising function which allows SHAPE to programatically name tables for the neighbourhood record OR split a named table and extract the embedded information from its naming.*

Description

This is a standardising function which allows SHAPE to programatiically name tables for the neighbourhood record OR split a named table and extract the embedded information from its naming.

Usage

```
nameTable_neighbourhood(func_Index, funcSplit = FALSE,
  func_sepString = getOption("shape_sepString"))
```

Arguments

```
func_Index    Integer value(s) for the unique genotype ID whose neighbourhood which will be
              recorded by the named table
```

| funcSplit | A logical toggle to control if this function is splitting a named table or not. So, FALSE (default) means we're creating a table name whereas TRUE is splitting a named table into it's parts. |

func_sepString

This is a character string used to collapse vectors of characters.

Value

If funcSplit is TRUE, then a vector of table names is returned. If FALSE, we split the table and return the data detailing the genotype ID whose neighbourhood is being recorded on the named table.

Examples

```
# This creates a table name in a standard way, it can also split table names to extract info
defineSHAPE()
nameTable_neighbourhood(2,FALSE)
nameTable_neighbourhood("Step_2",TRUE)
```

| nameTable_step | *This is a standardising function which allows SHAPE to programati-ically name tables for the step-wise record OR split a named table and extract the embedded information from its naming.* |

Description

This is a standardising function which allows SHAPE to programatiically name tables for the step-wise record OR split a named table and extract the embedded information from its naming.

Usage

```
nameTable_step(func_Index, funcSplit = FALSE,
  func_sepString = getOption("shape_sepString"))
```

Arguments

| func_Index | Integer value(s) for the step of a SHAPE run which will be recorded by this table |
| funcSplit | A logical toggle to control if this function is splitting a named table or not. So, FALSE (default) means we're creating a table name whereas TRUE is splitting a named table into it's parts. |

func_sepString

This is a character string used to collapse vectors of characters.

Value

If funcSplit is TRUE, then a vector of table names is returned. If FALSE, we split the table and return the data detailing the step number being recorded on the named table.

Examples

```
# This creates a table name in a standard way, it can also split table names to extract info
defineSHAPE()
nameTable_step(2,FALSE)
nameTable_step("Step_2",TRUE)
```

---

name_batchString	*This function is used to build or split character string to be used for naming batches of SHAPE runs.*

Description

This function is used to build or split character string to be used for naming batches of SHAPE runs.

Usage

```
name_batchString(funcBase, func_setID = NULL, func_jobID = NULL,
  func_repID = NULL, funcSplit = FALSE,
  func_sepString = getOption("shape_sepString"))
```

Arguments

funcBase	If building names this is the basal string element prefixing the name. If splitting, it is the vector of names to be split.
func_setID	If building names, a vector of the unique set IDs to be named, otherwise a logical of whether or not the batch naming structure includes sets
func_jobID	If building names, a vector of the unique job IDs to be named, otherwise a logical of whether or not the batch naming structure includes jobs
func_repID	If building names, a vector of the unique replicate IDs to be named, otherwise a logical of whether or not the batch naming structure includes replicates
funcSplit	Logical toggle TRUE if splitting names, FALSE to build string characters
func_sepString	This is the standard string separator for the SHAPE run

Value

Either a vector of character strings for the created batch names, or a matrix with the decomposed elements of the split batch name strings

Examples

```
# This simply produces or splits a standard named string.
name_batchString("myTest",1,9,3,FALSE,"_")
name_batchString("myTest_1_9_3",TRUE,TRUE,TRUE,TRUE,"_")
```

---

name_batchSubmit   *This is a function to programatically create R batch submission script names*

Description

This is a function to programatically create R batch submission script names

Usage

```
name_batchSubmit(inVar)
```

Arguments

inVar   This is the vector of character string(s) to be used for naming

Value

A vector of character string of length equal to input.

---

name_bodyScript   *This is a function to programatically create R script names*

Description

This is a function to programatically create R script names

Usage

```
name_bodyScript(inVar)
```

Arguments

inVar   This is the vector of character string(s) to be used for naming

Value

A vector of character string of length equal to input.

Examples

```
# Returns a standard named string
name_bodyScript(c("myJob","otherContent"))
```

```
name_parameterScript
```
This is a function to programatically create R script names

Description

This is a function to programatically create R script names

Usage

```
name_parameterScript(inVar)
```

Arguments

inVar   This is the vector of character string(s) to be used for naming

Value

A vector of character string of length equal to input.

Examples

```
# Returns a standard named string
name_parameterScript(c("myJob","otherContent"))
```

```
name_subScript
```
This is a function to programatically create R batch submission script names

---

Description

This is a function to programatically create R batch submission script names

Usage

```
name_subScript(inVar)
```

Arguments

inVar   This is the vector of character string(s) to be used for naming

Value

A vector of character string of length equal to input.

Examples

```
# Returns a standard named string
name_subScript(c("myJob","otherContent"))
```

| querryEstablished | *This function is used to find which elements of a population matrix are deemed as established. Established is determined by having a number of individuals greater than or equal to a definable proportion of the summed community size.* |

Description

This function is used to find which elements of a population matrix are deemed as established. Established is determined by having a number of individuals greater than or equal to a definable proportion of the summed community size.

Usage

```
querryEstablished(func_inMatrix, func_sizeCol = "popSize",
  func_fitCol = "fitness", func_estProp = 0.01)
```

Arguments

func_inMatrix

> This is a matrix which must contain at least one column named as func_sizeCol which contains the number of individuals in the communities' populations. But it may also be required to include a column func_fitCol if func_estProp is "Desai".

func_sizeCol DO NOT MODIFY - this is the column name that is querried to find population sizes

func_fitCol DO NOT MODIFY - this is the column name that is querried to find population fitness - only important if func_estProp is set to "Desai"

func_estProp If this value is less than 1 - This is the proportion of the current community size which is used to define a population as established it returns the rows of. If this value is greater than 1, it is the minimum number of individuals required before a population is considered as established. Lastly, it can be the character string "Desai", at which point - as per Desai 2007 - a lineage is established once it has 1/s individuals.

Value

A subset form of the input func_inMatrix matrix object containing the populations which are calculated as established.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

---

reportPopulations	*This is a convenience function to ensure that our population demographics are stored in a data frame and exists because R's standard functions can collapse single row frames to named vectors. It requires that all passed vectors be of the same length*

Description

This is a convenience function to ensure that our population demographics are stored in a data frame and exists because R's standard functions can collapse single row frames to named vectors. It requires that all passed vectors be of the same length

Usage

```
reportPopulations(func_numMuts, func_genotypeID, func_popSizes,
  func_fitnesses, func_births, func_deaths, func_mutants, func_progenitor,
  func_reportMat_colnames = getOption("shape_reportMat_colnames"))
```

Arguments

func_numMuts   This is a vector of the number of mutations held within each tracked genotype.

func_genotypeID

This is a vector of the unique genotype ID for each tracked population in the community.

func_popSizes

This is a vector of the number of individuals for each population of genotypes in the community.

func_fitnesses

This is a vector of the fitness for each genotpe being tracked.

func_births   This is a vector of the number of births produced by each population in this time step.

func_deaths   This is a vector of the number of deaths in each population in this time step.

func_mutants This is a vector of the number of mutants produced by each population in this time step.

func_progenitor

This is a vector of character strings expressing any progenitor genotypes which generated a mutant that fed into each genotype's population in this time step.

func_reportMat_colnames

DO NOT MODIFY - This is the vector of character strings to be assigned as the column names.

Value

A data frame with columns named as per func_reportMat_colnames.

Examples

```
# This returns a data.frame with a standard format
defineSHAPE()
reportPopulations(1:3,2:4,c(10,50,100),rep(1,3),
                  rep(0,3),c(10,10,10),c(1,2,0),c("","0_->_1","2"))
```

---

reset_shapeDB   *This is a convenience function to refresh connections to database files.*

Description

This is a convenience function to refresh connections to database files.

Usage

```
reset_shapeDB(func_conName, func_existingCon = NULL,
  func_type = "connect")
```

Arguments

func_conName   The filepath to which an SQLite connection is sought.

func_existingCon

        If any value other than NULL, then any existing connection is first dropped prior
        to attempting to form a connection to the func_conName filepath.

func_type   This should be a character string of either **connect**, in which case a connection is
        made/refreshed to the filepath in func_conName", or any other value will cause
        disconnection

Value

An SQLite connection object to an SQLite database.

Examples

```
# This function can be called to set, resset SQL connections
fileName_testCon <- paste(tempdir(),"/testCon.sqlite",sep="")
testCon <- reset_shapeDB(fileName_testCon)
reset_shapeDB(testCon, func_type = "disconnect")
```

```
retrieve_binaryString
```
This is a function to search our mutational database and then find the binary string of the genotypeID passed. This function is more efficient when the number of mutations for each genotypeID be passed as this helps reduce the tables of the mutational space that are searched. This matters when large genotypes are simulated.

Description

This is a function to search our mutational database and then find the binary string of the genotypeID passed. This function is more efficient when the number of mutations for each genotypeID be passed as this helps reduce the tables of the mutational space that are searched. This matters when large genotypes are simulated.

Usage

```
retrieve_binaryString(func_genotypeID, func_numMuts = NULL,
  func_subNaming, func_landscapeCon)
```

Arguments

`func_genotypeID`
> This is a vector of the unique genotype ID for each tracked population in the community.

`func_numMuts` This is a vector of the number of mutations held within each tracked genotype.

`func_subNaming`
> This is a logical which controls if the tables which report on all genotypes with X mutations should be forced into a single table or it SHAPE is allowed to split them into multiple tables.

`func_landscapeCon`
> This is the filepath to an SQLite database storing information for the complete explored and neighbouring fitness landscape of a SHAPE run.

Value

This returns a vector of character strings that represent the binary strings of the genotypes

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

---

runProcessing   *This is a wrapper function to process a SHAPE run and extract mean-ingful summary information.*

Description

This is a wrapper function to process a SHAPE run and extract meaningful summary information.

Usage

```
runProcessing(func_saveFile, func_subNaming, func_stepsCon,
  func_landscapeCon, func_hoodCon, func_estProp, func_size_timeStep,
  func_processObjects = getOption("shape_processedObjects"),
  func_hoodPriority = getOption("shape_const_hoodDepth"))
```

Arguments

func_saveFile

        This is the filepath where the SHAPE run processed objects are to be saved.

func_subNaming

        This is a logical which controls if the tables which report on all genotypes with X mutations should be forced into a single table or it SHAPE is allowed to split them into multiple tables.

func_stepsCon

        This is the filepath to an SQLite database storing information for the stepwise changes of a SHAPE run.

func_landscapeCon

        This is the filepath to an SQLite database storing information for the complete explored and neighbouring fitness landscape of a SHAPE run.

func_hoodCon   This is the filepath to an SQLite database storing information for high priority mutational neighbourhood information

func_estProp   This value is used to define the threshold size required for a population before it is considered established.

func_size_timeStep

        This is the proportion of a standard biological generation being considered to be within a single time step.

func_processObjects

        This is a vector of character strings which define the names of what objects will be produced and creates a global objects. DO NOT CHANGE THESE VALUES.

func_hoodPriority

        This is an object to control which strains we get deep neighbourhood information for It should be one of **"none","limited","priority","full"** setting this higher will cost more and more in post analysis runtime.

Value

This returns a string vector stating the result of trying to process for the specified filepath.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

---

runReplicate	*This is the function that runs the main body, or meaningful execution, of SHAPE experiments. In other words this is the main work-horse function that calls all the other parts and will execute you simulation run. It has the main parts of: 1. Stochastic Events; 2. Deaths; 3. Births; 4. Mutations; and during mutations this is where the mutational landscape is queried and updated as required. NOTE: Many of its internal operations are controlled by options with the suffix "shape_" and are not explicitly passed as arguments at call to this function.*

Description

This is the function that runs the main body, or meaningful execution, of SHAPE experiments. In other words this is the main work-horse function that calls all the other parts and will execute you simulation run. It has the main parts of: 1. Stochastic Events; 2. Deaths; 3. Births; 4. Mutations; and during mutations this is where the mutational landscape is queried and updated as required. NOTE: Many of its internal operations are controlled by options with the suffix "shape_" and are not explicitly passed as arguments at call to this function.

Usage

```
runReplicate(func_inputFrames, func_currStep, func_stepCounter,
  func_growthModel = getOption("shape_const_growthForm"),
  func_growthRate = getOption("shape_const_growthRate"),
  func_landscapeModel = getOption("shape_simModel"),
  func_fileName_dataBase = getOption("shape_fileName_dataBase"))
```

Arguments

func_inputFrames

> This is a list of data.frames, either 1 or 2 elements, reporting on the last one or two steps in the simulation.

func_currStep

> This is an integer value counting the absolute step in the simulation, its value is never reset.

func_stepCounter

> This is an integer value which is a counter in the most tradititional sense. It's job is to track if it's time for a Stochastic event to trigger and its value is reset at that point.

func_growthModel
> This is the growth model of the SHAPE run, it is passed here as a computational convenience since it is used numerous times in the function

func_growthRate
> This is the growth rate of the SHAPE run, it is passed here as a computational convenience since it is used numerous times in the function

func_landscapeModel
> This is the fitness landscape model of the SHAPE run, it is passed here as a computational convenience since it is used numerous times in the function

func_fileName_dataBase
> This is the filepaths of DBs of the SHAPE run, it is passed here as a computational convenience since it is used numerous times in the function

Value

Returns a new list of 2 data.frames reporting on the state of SHAPE community for the last 2 time steps - ie: the one just run, and the most prior step.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

runSHAPE	*This is the actual running of shape, it will initialise objects and values which are calculated from the parameters that have been set - see the options with the suffix 'shape_'. It will establish the database output files and other initial conditions and then perform replicate simulations as appropriately defined. In essense this is the master wrapper function for all other functions. If you want to test/see SHAPE's default run then simply call this function after loading the library you'll see an experiment built under your root directory. It at least requires that defineSHAPE have been run, else this is going to fail.*

Description

This is the actual running of shape, it will initialise objects and values which are calculated from the parameters that have been set - see the options with the suffix 'shape_'. It will establish the database output files and other initial conditions and then perform replicate simulations as appropriately defined. In essense this is the master wrapper function for all other functions. If you want to test/see SHAPE's default run then simply call this function after loading the library you'll see an experiment built under your root directory. It at least requires that defineSHAPE have been run, else this is going to fail.

Usage

```
runSHAPE(loop_thisRep = getOption("shape_thisRep"),
  workingReplicates = seq(getOption("shape_thisRep"),
  getOption("shape_maxReplicates"), by = 1),
  tmpEnvir_recycleParms = new.env())
```

Arguments

loop_thisRep This is the first replicate value to be simulated in this run, it is standard 1 but can be changed to help with recovery in the middle of a series of replicates.

workingReplicates

This is the maximum replicate number to to simulated in this call. It is meaningfully different from the number of replicates to be run only when loop_thisRep != 1.

tmpEnvir_recycleParms

This is an environment used to temporarily store loaded RData file objects so that parameters from previous runs, that were stored in RData, can be read back in as required.

Examples

```
# First step is to set parameters for the run, this could be done manually but I
# recommend using the defineSHAPE function which has a default setting for all
# possible parameters and will calculate the value of derived/conditional parameters.
defineSHAPE()
# Now you can run the simulations, you should get printout to your stdout.
 runSHAPE()
# Now go and check the SHAPE working directory, which can be found at:
getOption("shape_workDir")
list.files(getOption("shape_workDir"))
# You'll have an experiment folder as well as post-analysis folder
# created each with appropriate output!
```

---

set_const_NK_interactionsMat

This is a function to just return a matrix that defines the sitewise dependencies for an NK fitness landscape. If K == 0 or, this is not an NK simulation, it return NULL

Description

This is a function to just return a matrix that defines the sitewise dependencies for an NK fitness landscape. If K == 0 or, this is not an NK simulation, it return NULL

Usage

```
set_const_NK_interactionsMat(func_simModel = getOption("shape_simModel"),
func_genomeLength = getOption("shape_genomeLength"),
func_numInteractions = getOption("shape_const_numInteractions"))
```

Arguments

`func_simModel`

This is the fitness landscape model being simulated

`func_genomeLength`

This is the number of sites in the genome being simulated

`func_numInteractions`

An integer value defining the number of sites that interact with each other site

Value

Either NULL, or a matrix with K + 1 columns, detailing the sites interacting with a focal site - identified by the row number and the cell values of the columns.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

`set_const_RMF_globalOptima`

This function samples the space of all possible genotypes and then defines one that will be considered as the independent fitness contribution global optima.

Description

This function samples the space of all possible genotypes and then defines one that will be considered as the independent fitness contribution global optima.

Usage

```
set_const_RMF_globalOptima(func_simModel = getOption("shape_simModel"),
  func_genomeLength = getOption("shape_genomeLength"),
  func_initDistance = getOption("shape_const_RMF_initiDistance"),
  func_sepString = getOption("shape_sepString"))
```

Arguments

`func_simModel`

> This is the fitness landscape model being simulated

`func_genomeLength`

> The number of sites in the genome being simulated

`func_initDistance`

> This is the number of mutations found in the global optimal genotype

`func_sepString`

> This is the string collapse separator used in the run

Value

A character string of genome positions at which there ought to be mutations to be optimal

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

`set_DepbySite_ancestFitness`

> *This is a convenience function for setting the dependent fitness values of sites in an NK fitness landscape model. This allows the dependent fitness of sites to be calculated once and then referenced as mutations occur. It makes exploring this style of fitness landscape a bit more computationally friendly - as it generally isn't.*

Description

This is a convenience function for setting the dependent fitness values of sites in an NK fitness landscape model. This allows the dependent fitness of sites to be calculated once and then referenced as mutations occur. It makes exploring this style of fitness landscape a bit more computationally friendly - as it generally isn't.

Usage

```
set_DepbySite_ancestFitness(func_simModel = getOption("shape_simModel"),
  func_const_siteBystate_fitnessMat = getOption("shape_const_siteBystate_fitnessMat
  func_const_NK_interactionMat = getOption("shape_const_NK_interactionMat"))
```

Arguments

`func_simModel`

> This is the fitness landscape model being simulated

`func_const_siteBystate_fitnessMat`

> This is the sitewise independent fitness contributions in the fitness landscape

`func_const_NK_interactionMat`

> This defines the sitewise dependencies based on the K interactions.

Value

Either the dependent sitewise fitness contributions in an NK fitness landscape, or NA.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

---

set_RMF_indWeight   *In a RMF fitness landscape model, there is a weighting value applied to the independent fitness contribution term. This function calculates that value for the run*

Description

In a RMF fitness landscape model, there is a weighting value applied to the independent fitness contribution term. This function calculates that value for the run

Usage

```
set_RMF_indWeight(func_simModel = getOption("shape_simModel"),
  func_numDraws = 1e+08, func_distType = getOption("shape_constDist"),
  func_distParms = getOption("shape_const_distParameters"),
  func_const_RMF_theta = getOption("shape_const_RMF_theta"))
```

Arguments

func_simModel

This is the model of fitness landscape being considered

func_numDraws

This is the number of draws taken from the independent term's distribution so that we can identify the amount of variance in that distribution. It should be a large integer – eg 5e7

func_distType

This is the distribution string reference for this run

func_distParms

These are the parameters for this runs distribution function

func_const_RMF_theta

This is the theta value which is multiplied to the variance in the distribution. The value returned will be a product of this numeric and the variance calulated. From Neidhart 2014 theta is measured as: theta = c / sqrt var random_component and so if we want to calculate "c" we return the product of theta and sqrt of variance in the distribution

Value

A single numeric value, which may be NA if a non Rough Mount Fuji model is being simulated

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

```
set_siteByState_fitnessMat
```
> *This function is designed to establish an initial object which maps the fitness values of genome positions based on the state of that site. At present, this has no meaning if the model of simulation is no NK, Additive, or Fixed. Where the first is Kauffman's NK model and form of calculations, Additive is what that word would make you think for fitness effects of mutations at sites, and Fixed is when user supplied a defined fitness matrix that describes the entire fitness landscape. NOTE: This function should likely be called without supplying any non-default arguments as it will use the shape_ options defined.*

Description

This function is designed to establish an initial object which maps the fitness values of genome positions based on the state of that site. At present, this has no meaning if the model of simulation is no NK, Additive, or Fixed. Where the first is Kauffman's NK model and form of calculations, Additive is what that word would make you think for fitness effects of mutations at sites, and Fixed is when user supplied a defined fitness matrix that describes the entire fitness landscape. NOTE: This function should likely be called without supplying any non-default arguments as it will use the shape_ options defined.

Usage

```
set_siteByState_fitnessMat(func_simModel = getOption("shape_simModel"),
  func_const_fixedFrame = getOption("shape_const_fixedFrame"),
  func_const_siteStates = getOption("shape_const_siteStates"))
```

Arguments

```
func_simModel
```
> This is the fitness landscape model being simulated

```
func_const_fixedFrame
```
> This is a contextual object that described constant fitness effects

```
func_const_siteStates
```
> These are the posibble states for genome sites, at present this ought to be "0" and/or "1"

Value

A contextually meaningful matrix describing fitness effects of mutations/genotypes, where based on the context NULL may be returned.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

---

shapeCombinations *This is a function to take the input parameters and build the parameter combinations*

Description

This is a function to take the input parameters and build the parameter combinations

Usage

```
shapeCombinations(func_inLines, func_comboRef, func_indepRef, func_condRef)
```

Arguments

func_inLines These are the template lines of text to be updated.

func_comboRef

This is the reference identifiers for grouped as pairwise parameter combinations

func_indepRef

This is the reference identifiers for independent parameter values not to be done pairwise

func_condRef This is the reference indetifiers for grouped parameter combinations which are conditional on others.

Value

A table of parameter combinations which represents the combination of experimental parameters for a SHAPE experiment.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

`shapeExperiment`	*This is a function used to read the SHAPE_experimentalDesign type input file and then build a SHAPE experiment by creating all the folder structure, .R and .sh scripts required to programatically run your experiment – excluding post-analysis, that's a you problem.*

Description

This is a function used to read the SHAPE_experimentalDesign type input file and then build a SHAPE experiment by creating all the folder structure, .R and .sh scripts required to programatically run your experiment – excluding post-analysis, that's a you problem.

Usage

```
shapeExperiment(func_filepath_toDesign, func_templateDir,
  func_maxGrouped_perShell = 2, func_filePath_R = NULL,
  func_baseCall = "CMD BATCH",
  func_rArgs = "\"--args shape_thisRep=1 shape_outDir='fake_serverPath/fakeDir/'\""
  func_remoteLocation = "$TMPDISK", func_submitArgs = c(number_ofCores
  = "-c 1", memory = "--mem=8192", jobName = "-J fakeJob", wallTime =
  "-t 14-00:00:00", fileOut = "-o fakeOut"), func_processingCores = 1,
  func_suppressOld_summaryFiles = FALSE)
```

Arguments

`func_filepath_toDesign`

> This is the absolute filepath which points to the SHAPE_experimentalDesign like template you'd like used to identify parameter combinations for building your experiment.

`func_templateDir`

> This is the absolute filepath to a directory on your machine where the SHAPE template scripts/files have been saved. They are used by this function to help build your experiment.

`func_maxGrouped_perShell`

> Integer value defining the maximal number of jobs that an output shell script will try to have run in parallel once executed. This is related to your parallel computing potential.

`func_filePath_R`

> This is the absolute path to the R application on the system where SHAPE would be run via BATCH MODE, its value is applied in shell scripts written for running the experiment. If left NULL then this function will try to use standard R install paths of which I'm aware.

`func_baseCall`

> This is a string element of arguments when calling BATCH MODE if R via shell script.

func_rArgs   This is a character string which represent additional arguments to be passed via
                  shell script BATCH mode call of R. I consider it most practicable to set the
                  replicate and output directory of SHAPE.

func_remoteLocation

                  The filepath of the compute node on a remote server where your job would
                  be run. The default is based on the environment variable value used in CAC's
                  SLURM submission system.

func_submitArgs

                  This is information concerning sheel script lines for automatic submission of
                  jobs to a remote server's submission system. I'm basing this off of the SLURM
                  system of the Center for Advanced Computing Queen's University computing
                  platform. If your system is different you may need to tweak this. Sorry? This
                  should be a vector of arguments passed for job submissions on a remote server
                  The example here would call 1 core with 8 Gb RAM and a wall time of 14 days
                  and an outFile be named You can add more arguments if your server requires
                  this, they'll get used. BUT where the job's name MUST be identified as —
                  fakeJob —- and the output log as — fakeOut —, you can change the argument
                  queues I also assume your remote server will create a local directory on the
                  compute nodes whre your job once submitted, and that there will be the location
                  defined by func_remoteLocation.

func_processingCores

                  This is the number of parallel cores you would like the summairseExperiment()
                  to call when trying to process your experimental output.

func_suppressOld_summaryFiles

                  Logical flag controlling if your summariseExperiment() will delete old output
                  summary files. setting to FALSE (default) is ideal if you could ever expect you
                  might need to restart whereas TRUE becomes practical if you are worried you'd
                  have updated output to process and you want to ensure a fresh processing start.

Value

If no error is encountered, a message will be returned suggesting the build was successful. SHAPE
makes no effort to perform validation of this effort to build the experiment and presumes no fatal
errors is sufficient evidence.

Examples

```
# This function relies on script templates which can be found at:
# 'https://github.com/JDench/SHAPE_library/tree/master/SHAPE_templates'
# Once these have been downloaded you can pass the appropriate filepath values
# to the first two arguments. For this example, I'll assume you've installed
# them to a folder position that is now just under the root of your
# R-environment working directory.
# However, before runing the function we need to parameterise your run of SHAPE,
# here I call the default parameters:
defineSHAPE()
# Now using the default templates we design an experiment folder complete with
# shell scripts to submit our work programatically.
# NOTE: Again, this example assumes you've downloaded the templates and placed
```

```
#          them at the next filepath and directory-path locations
shapeExperiment(func_filepath_toDesign = "~/SHAPE_templates/SHAPE_experimentalDesign.v.1.r",
                            func_templateDir = "~/SHAPE_templates/")
# You should be greeted with a message suggesting your experiment is built.
# You can find the files now at that script's SHAPE workingDirectory.
list.files(getOption("shape_workDir"))
# Voila!  You can go see the spread of variable evolutionary parameters that were
# considered by looking at -- yourJob_parameterCombos.table -- which is a tab
# delimited file.
# Lastly, you may have R installed elsewhere and so want to have that noted while
# your experiment is built because the shell scripts will need to point to the correct place
shapeExperiment(func_filepath_toDesign = "~/SHAPE_templates/SHAPE_experimentalDesign.v.1.r",
                            func_templateDir = "~/SHAPE_templates/",
                            func_filePath_R = "~/your_R_folder/R_app/bin/R")
# Now obviously the above location likely is not where you installed R,
# but ideally you get the point. The difference is in how the shell scripts were written.
```

---

stopError	*This is a convenience wrapper for sending an error and ending the SHAPE run as well as the R environment. It will print a message and then traceback( ) report before pausing and quiting the R session. This exists to help debugging when SHAPE is run in batch-mode.*

Description

This is a convenience wrapper for sending an error and ending the SHAPE run as well as the R environment. It will print a message and then traceback() report before pausing and quiting the R session. This exists to help debugging when SHAPE is run in batch-mode.

Usage

```
stopError(func_message)
```

Arguments

func_message The message to be sent to screen prior to ending the R session.

Note

There is no example as this functions role is to print a message and then quit the R run.

summariseExperiment

This function is a wrapper for getting a summary of the results of an rSHAPE run and/or experiment as a whole. The former is presumed to be of greater use but either is fine as per your needs. This wrapper will cause RData files to be created which contain the summarised experimental details that you can then use more easily for analysis.

Description

This function is a wrapper for getting a summary of the results of an rSHAPE run and/or experiment as a whole. The former is presumed to be of greater use but either is fine as per your needs. This wrapper will cause RData files to be created which contain the summarised experimental details that you can then use more easily for analysis.

Usage

```
summariseExperiment(func_processingTypes = c("fileList", "parameters",
  "popDemographics", "repeatability"), func_numCores = 1,
  func_suppressOld = FALSE)
```

Arguments

func_processingTypes

A vector of character strings which define the type of processing to be performed when callign this experimental analysis wrapper function. At present, the types include: "fileList", "parameters", "popDemographics","repeatability" as per the rSHAPE option - shape_procExp_filenames

func_numCores

Integer number of computer cores to be requested for performing parallel processing of experiment files. It defaults as 1, which effectively means in tandem - ie: not parallel.

func_suppressOld

This is a logical toggle if files which exist in the expected location should be deleted. Default is FALSE and the function will simply not process alraedy processed output. TRUE might be useful as a means to forcibly re-run the summary fresh.

Value

A message detailing if the requested processed files can be found, either affirmative for all or a note when at least one is missing.

Note

There is no example as this cannot work without a complete rSHAPE experiment to be analysed.

summarise_evolRepeatability

> *This function will use output from summarise_experimentFiles and summarise_experimentParameters to help with expectations concerning run output and handling. This will save an RData file which will contain one object: all_popSets, which is a list of relevant control information about I/O and then a series of other RData files which contain the demographics information as a matrix with the mean and standard deviation of demographics for all replicates.*

Description

This function will use output from summarise_experimentFiles and summarise_experimentParameters to help with expectations concerning run output and handling. This will save an RData file which will contain one object: all_popSets, which is a list of relevant control information about I/O and then a series of other RData files which contain the demographics information as a matrix with the mean and standard deviation of demographics for all replicates.

Usage

```
summarise_evolRepeatability(funcSave_jobExpression,
  func_saveFile = getOption("shape_procExp_filenames")["repeatability"],
  func_experimentDir = getOption("shape_workDir"),
  func_saveDir = getOption("shape_postDir"),
  func_refFile = getOption("shape_procExp_filenames")[c("fileList",
  "parameters")], func_workEnvir = new.env(),
  func_objPrefix = "Repeat_",
  func_sepString = getOption("shape_sepString"),
  func_string_line_ofDescent = getOption("shape_string_lineDescent"),
  func_processedPattern = getOption("shape_processedData_filePattern"),
  func_sepLines = getOption("shape_sepLines"))
```

Arguments

funcSave_jobExpression

> This is a string expression that can be used to find elements of the experiment being analysed. It should be some robust unique string or regular expression.

func_saveFile

> This is the filepath and filename (ending in .RData please) to which the results of this step will be saved.

func_experimentDir

> This is the filepath to the root directoy under which all your experimental files can be found.

func_saveDir   This is the directory to which output will be saved.

func_refFile   This is the filepath to the reference file that contains information regarding all the processed files for the rSHAPE experiment.

`func_workEnvir`

> This is an environment used to load files with the load function. It's used to encapsulate the loaded information to a controlled space.

`func_objPrefix`

> This is a character string for programatic naming of objects of this type.

`func_sepString`

> This is rSHAPE's sepString option but here to be passed into foreach

`func_string_line_ofDescent`

> This is rSHAPE's option of similar name to be passed into foreach

`func_processedPattern`

> This is rSHAPE's option of the similar name to be passed into foreach

`func_sepLines`

> This is rSHAPE's option of the similar name passed into foreach

Note

There is no example as this cannot work without a complete rSHAPE experiment to be analysed.

`summarise_experimentFiles`

> *This function will find all initially processed output files from individual replicates and return summary information. That information is saved to an RData file which will contain 3 objects: all_proccessedFiles, all_jobInfo, all_dividedFiles*

Description

This function will find all initially processed output files from individual replicates and return summary information. That information is saved to an RData file which will contain 3 objects: all_proccessedFiles, all_jobInfo, all_dividedFiles

Usage

```
summarise_experimentFiles(func_experimentDir = getOption("shape_workDir"),
func_saveFile = getOption("shape_procExp_filenames")["fileList"],
func_search_filePattern = getOption("shape_processedData_filePattern"),
func_sepString = getOption("shape_sepString"))
```

Arguments

`func_experimentDir`

> This is the filepath to the root directoy under which all your experimental files can be found.

`func_saveFile`

> This is the filepath and filename (ending in .RData please) to which the results of this step will be saved.

`func_search_filePattern`

> This is a string which can be used to search and find the files which relate to the processed output of individual replicates rSHAPE runs.

`func_sepString`

> This is the character string which was used for commonly collapsing elements in the rSHAPE run.

Note

There is no example as this cannot work without a complete rSHAPE experiment to be analysed.

`summarise_experimentParameters`

> *This function will use output from summarise_experimentFiles to locate all parameter files and then report on all those parameters for the jobs that were run. This will save an RData file which will contain one object: all_parmInfo*

Description

This function will use output from summarise_experimentFiles to locate all parameter files and then report on all those parameters for the jobs that were run. This will save an RData file which will contain one object: all_parmInfo

Usage

```
summarise_experimentParameters(func_workEnvir = new.env(),
  func_saveFile = getOption("shape_procExp_filenames")["parameters"],
  func_experimentDir = getOption("shape_workDir"),
  func_refFile = getOption("shape_procExp_filenames")["fileList"])
```

Arguments

`func_workEnvir`

> This is an environment used to load files with the load function. It's used to encapsulate the loaded information to a controlled space.

`func_saveFile`

> This is the filepath and filename (ending in .RData please) to which the results of this step will be saved.

`func_experimentDir`

> This is the filepath to the root directoy under which all your experimental files can be found.

`func_refFile` This is the filepath to the reference file that contains information regarding all the processed files for the rSHAPE experiment.

Note

There is no example as this cannot work without a complete rSHAPE experiment to be analysed.

---

summarise_popDemographics

> *This function will use output from summarise_experimentFiles and*
> *summarise_experimentParameters to help with expectations concern-*
> *ing run output and handling. This will save an RData file which will*
> *contain one object: all_popSets, which is a list of relevant control*
> *information about I/O and then a series of other RData files which*
> *contain the demographics information as a matrix with the mean and*
> *standard deviation of demographics for all replicates.*

Description

This function will use output from summarise_experimentFiles and summarise_experimentParameters to help with expectations concerning run output and handling. This will save an RData file which will contain one object: all_popSets, which is a list of relevant control information about I/O and then a series of other RData files which contain the demographics information as a matrix with the mean and standard deviation of demographics for all replicates.

Usage

```
summarise_popDemographics(funcSave_jobExpression,
  func_saveFile = getOption("shape_procExp_filenames")["popDemographics"],
  func_experimentDir = getOption("shape_workDir"),
  func_saveDir = getOption("shape_postDir"),
  func_refFile = getOption("shape_procExp_filenames")[c("fileList",
  "parameters")], func_workEnvir = new.env(),
  func_objPrefix = "popDemo_")
```

Arguments

funcSave_jobExpression

  This is a string expression that can be used to find elements of the experiment being analysed. It should be some robust unique string or regular expression.

func_saveFile

  This is the filepath and filename (ending in .RData please) to which the results of this step will be saved.

func_experimentDir

  This is the filepath to the root directoy under which all your experimental files can be found.

func_saveDir This is the directory to which output will be saved.

func_refFile This is the filepath to the reference file that contains information regarding all the processed files for the rSHAPE experiment.

func_workEnvir

  This is an environment used to load files with the load function. It's used to encapsulate the loaded information to a controlled space.

func_objPrefix

  This is a character string for programatic naming of objects of this type.

Note

There is no example as this cannot work without a complete rSHAPE experiment to be analysed.

`trimQuotes`	*This is a function to trim a string by removing the first and last character, it's used to trim quotation marks used in the parameter input*

Description

This is a function to trim a string by removing the first and last character, it's used to trim quotation marks used in the parameter input

Usage

```
trimQuotes(funcIn)
```

Arguments

`funcIn`   a vector of character strings which you want trimmed

Value

character vector of length equal to the input

Examples

```
# It removes leading and trailing string positions, use when quotations are known to exist.
trimQuotes(c('"someWords"','otherwords"',"is_changed"))
```

`updateLines`	*This is a function which is used to update lines that are searched and replace in a manner conditional to this script's circumstances The input lines can be a vector of any length, and the search patterns can be a list of any length where each list vector is used together. The values should be a list of information used as replacement info.*

Description

This is a function which is used to update lines that are searched and replace in a manner conditional to this script's circumstances The input lines can be a vector of any length, and the search patterns can be a list of any length where each list vector is used together. The values should be a list of information used as replacement info.

Usage

```
updateLines(func_inLines, func_searchPattern, func_values)
```

Arguments

func_inLines These are the lines that are to be updated before output

func_searchPattern

These are the string-s- to be searched for replacement

func_values These are the values that are to replace the searched strings.

Value

A vector of character strings that has now been updated.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

---

writeParameters *This is a file for updating the post analysis plotting script and creating an updated copy in the experiment's folder*

Description

This is a file for updating the post analysis plotting script and creating an updated copy in the experiment's folder

Usage

```
writeParameters(func_infile, func_inParms, func_inCombos, func_outDir,
  func_bodyScript, func_ExternalStopper,
  func_sepString = getOption("shape_sepString"))
```

Arguments

func_infile This is the filepath location for the template script to be writte in.

func_inParms These are the parameters to be updated in the plotting file

func_inCombos

This is the combination of parameters to be written

func_outDir This is the director filepath to which output should be written.

func_bodyScript

This is a run body of SHAPE script to be read in as template

func_ExternalStopper

This is a file placed externally used as a logical flag that SHAPE should stop trying to seed new replicates to be run.

func_sepString

This is the common string for collapsing information.

Value

A character string indicating that the plotting file-s- have been written

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.

---

write_subScript	*This function is used to programatically take vectors of paramters and write suites of R parameter scripts that will form part of a SHAPE experiment that is being built for running. This is a wrapper for writting out the suite of necessary scripts to form a run.*

Description

This function is used to programatically take vectors of paramters and write suites of R parameter scripts that will form part of a SHAPE experiment that is being built for running. This is a wrapper for writting out the suite of necessary scripts to form a run.

Usage

```
write_subScript(func_subScipt, func_outDir, func_inCombos, func_inParms,
  func_maxJobs, func_appLocation, func_commonArgs, func_submitArgs,
  func_remoteLocation, func_passedArgs,
  func_externalStopper = getOption("shape_external_stopFile"),
  func_sepString = getOption("shape_sepString"))
```

Arguments

func_subScipt
> This is the template script that needs to be replicated

func_outDir   This is the filepath directory where output should be placed

func_inCombos
> This is the combinations of parameters that are to be used in the experiment.

func_inParms # These are additional parameters to be implemented in writing out combinations.

func_maxJobs This is the maximum number of individual R jobs that should be called at once by the shell submission scripts, it can affect both local and remote server calls.

func_appLocation
> This is the filepath for R so that batch mode runs can be called.

func_commonArgs
> These are common arguments important when running the batch mode

func_submitArgs
> These are common arguments important when submitting the batch mode

func_remoteLocation

> This is a remote server location where an experiment built is to be run it affects the filepathing called by submission scripts and the associated batch mode runs performed.

func_passedArgs

> These are arguments passed through this wrapper to inner functions.

func_externalStopper

> This is a file which exists as a flag for stopping SHAPE from trying to create additional replicates.

func_sepString

> This is the common string used to collapse information.

Value

A character string that should indicate the experiment has been created. Otheriwse this has failed.

Note

There is no example as this cannot work outisde of a runSHAPE call, it requires data produced by the simulation experiment.