Package ‘sicegar’

November 30, 2016

Type Package

Title Analysis of Single-Cell Viral Growth Curves

Version 0.1

Date 2015-12-17

Description The sicegar package classifies time course fluorescence data of viral growth. The package categorize time course data into one of four categories, ``ambiguous'', ``no signal'', ``infection'', and ``infection and lysis'' by fitting a series of mathematical models to the data. Biologically relevant parameters associated with each of these models are also reported, allowing for analysis of virus yield, replication rate, and the starting time of replication. The origin of the package name came from ``SIngle CEll Growth Analysis in R''.

URL `https://github.com/wilkelab/sicegar`

Imports dplyr, minpack.lm, fBasics, ggplot2, stats

License GPL-2 | GPL-3

LazyData true

Suggests knitr

VignetteBuilder knitr

BugReports `https://github.com/wilkelab/sicegar/issues`

Collate 'categorize.R' 'mainFunctions.R' 'lineFitFunctions.R' 'sigmoidalFitFunctions.R' 'doublesigmoidalFitFunctions.R' 'normalizationFunction.R' 'sicegar.R' 'dataInputCheck.R' 'numericalReCalculation.R' 'generateFigures.R'

RoxygenNote 5.0.1

NeedsCompilation no

Author M. Umut Caglar [aut, cre],
Claus O. Wilke [aut]

Maintainer M. Umut Caglar `<umut.caglar@gmail.com>`

R topics documented:

`categorize` *Categorize input data by comparing the AIC values of the three fitted models.*

Description

Catagorizes dat using the results of all three fitted models (linear, sigmoidal, and double sigmoidal).

Usage

```
categorize(parameterVectorLinear, parameterVectorSigmoidal,
  parameterVectorDoubleSigmoidal, threshold_line_slope_parameter = 0.01,
  threshold_intensity_interval = 0.1, threshold_difference_AIC = 0,
  threshold_lysis_finalAsymptoteIntensity = 0.75, threshold_AIC = -10)
```

Arguments

`parameterVectorLinear`
: output from lineFitFunction.

`parameterVectorSigmoidal`
: output from sigmoidalFitFunction.

`parameterVectorDoubleSigmoidal`
: output from doublesigmoidalFitFunction.

`threshold_line_slope_parameter`
: minimum for line slope (Default is 0.01).

`threshold_intensity_interval`
: minimum for intensity range (Default is 0.1).

`threshold_difference_AIC`
: choice between sigmoidal and double sigmoidal by using AIC values (Default is 0).

`threshold_lysis_finalAsymptoteIntensity`
: minimum amount of decrease for double sigmoidal (Default is 0.75).

`threshold_AIC`
: maximum AIC values in order to have a meaningful fit (Default is -10).

Value

Function returns one of the three text outputs, "no_signal", "infection", or "infection&lysis".

Examples

```
# Example 1 with double sigmoidal data
time=seq(3,24,0.1)

#simulate intensity data and add noise
noise_parameter=0.2
intensity_noise=runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=doublesigmoidalFitFormula(time,
                                    finalAsymptoteIntensity=.3,
                                    maximum=4,
                                    slope1=1,
                                    midPoint1=7,
                                    slope2=1,
                                    midPointDistance=8)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput = normalizeData(dataInput,dataInputName="batch_01_21_2016_samp007623")


# Fit linear model
linearModel=fitFunction(dataInput=normalizedInput,
                                  model="linear",
                                  n_runs_min=20,
                                  n_runs_max=500,
                                  showDetails=FALSE)

# Fit sigmoidal model
sigmoidalModel=fitFunction(dataInput=normalizedInput,
                                     model="sigmoidal",
                                     n_runs_min=20,
                                     n_runs_max=500,
                                     showDetails=FALSE)

# Fit double sigmoidal model
doubleSigmoidalModel=fitFunction(dataInput=normalizedInput,
                                           model="doublesigmoidal",
                                           n_runs_min=20,
                                           n_runs_max=500,
                                           showDetails=FALSE)
```

```
outputCluster=categorize(parameterVectorLinear=linearModel,
                         parameterVectorSigmoidal=sigmoidalModel,
                         parameterVectorDoubleSigmoidal=doubleSigmoidalModel)


# Example 2 with sigmoidal data
time=seq(3,24,0.1)

#simulate intensity data and add noise
noise_parameter=0.2
intensity_noise=runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=sigmoidalFitFormula(time, maximum=4, slope=1, midPoint=8)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput= normalizeData(dataInput,dataInputName="batch_01_21_2016_samp007623")


# Fit linear model
linearModel=fitFunction(dataInput=normalizedInput,
                                  model="linear",
                                  n_runs_min=20,
                                  n_runs_max=500,
                                  showDetails=FALSE)

# Fit sigmoidal model
sigmoidalModel=fitFunction(dataInput=normalizedInput,
                                     model="sigmoidal",
                                     n_runs_min=20,
                                     n_runs_max=500,
                                     showDetails=FALSE)

# Fit double sigmoidal model
doubleSigmoidalModel=fitFunction(dataInput=normalizedInput,
                                           model="doublesigmoidal",
                                           n_runs_min=20,
                                           n_runs_max=500,
                                           showDetails=FALSE)


outputCluster=categorize(parameterVectorLinear=linearModel,
                         parameterVectorSigmoidal=sigmoidalModel,
                         parameterVectorDoubleSigmoidal=doubleSigmoidalModel)
```

`categorize_nosignal` *Checks for signal in the data.*

Description

Checks if signal is present in the data. Often a high percentage of high through-put data does not contain a signal. Checking if data does not contain signal before doing a sigmoidal or double sigmoidal fit can make analysis of data from high through-put experiments much faster.

Usage

```
categorize_nosignal(parameterVectorLinear,
  threshold_line_slope_parameter = 0.01, threshold_intensity_interval = 0.1)
```

Arguments

`parameterVectorLinear`	is the output of lineFitFunction.
`threshold_line_slope_parameter`	minimum for line slope (Default is 0.01).
`threshold_intensity_interval`	minimum for intensity range (Default is 0.1).

Value

Function returns one of two text outputs "no_signal" or "NOT no_signal".

Examples

```
# Example 1 with double sigmoidal data

time=seq(3,24,0.1)

#simulate intensity data and add noise
noise_parameter=0.2
intensity_noise=runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=doublesigmoidalFitFormula(time,
                                     finalAsymptoteIntensity=.3,
                                     maximum=4,
                                     slope1=1,
                                     midPoint1=7,
                                     slope2=1,
                                     midPointDistance=8)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput = normalizeData(dataInput,dataInputName="batch_01_21_2016_samp007623")


# Fit linear model
linearModel=fitFunction(dataInput=normalizedInput,
                                   model="linear",
                                   n_runs_min=20,
                                   n_runs_max=500,
                                   showDetails=FALSE)
```

```
isThis_nosignal=categorize_nosignal(parameterVectorLinear=linearModel)



# Example 2 with no_signal data

time=seq(3,24,0.1)

#simulate intensity data and add noise
noise_parameter=0.05
intensity_noise=runif(n = length(time),min = 0,max = 1)*noise_parameter*2e-04
intensity=doublesigmoidalFitFormula(time,
                                   finalAsymptoteIntensity=.3,
                                   maximum=2e-04,
                                   slope1=1,
                                   midPoint1=7,
                                   slope2=1,
                                   midPointDistance=8)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizeInput= normalizeData(dataInput,dataInputName="batch_01_21_2016_samp007623")


# Fit linear model
linearModel=fitFunction(dataInput=normalizeInput,
                                 model="linear",
                                 n_runs_min=20,
                                 n_runs_max=500,
                                 showDetails=FALSE)

isThis_nosignal=categorize_nosignal(parameterVectorLinear=linearModel)
```

`dataCheck` *Checks if data is in correct format.*

Description

Checks if the input data is appropirate and converts it into an appropirate form. The input data frame should contain two columns named time and intensity for timeData and intensityData. If the data frame is in a list its name in the list should be $timeIntensityData.

Usage

```
dataCheck(data, showDetails = TRUE)
```

Arguments

`data`	the input data. It can be either a list that contains a data frame in .$timeIntensityData or a data frame by itself.
`showDetails`	if TRUE the function will provide the printout "check done" if everything is OK. Default is FALSE

Examples

```
# Example 1

# generate data frame
time = seq(3,48,0.5)
intensity=runif(length(time), 3.0, 7.5)
dataInput = data.frame(time,intensity)

# Apply dataCheck function
dataOutputVariable = dataCheck(dataInput)

# Example 2

# generate data frame
time = seq(3,48,0.5)
intensity=runif(length(time), 3.0, 7.5)
dataInput = data.frame(time,intensity)

# Normalize Data
dataOutput = normalizeData(dataInput)
dataInput2=dataOutput

# Apply dataCheck function
dataOutputVariable2 = dataCheck(dataInput2)
```

`doublesigmoidalFitFormula` *Double Sigmoidal Formula*

Description

Calculates intensities using the double-sigmoidal model fit and the parameters (maximum, final asymptote intensity, slope1, midpoint1, slope2, and mid point distance).

Usage

```
doublesigmoidalFitFormula(x, finalAsymptoteIntensity, maximum, slope1,
  midPoint1, slope2, midPointDistance)
```

Arguments

`x`	the "time" (time) column of the dataframe
`finalAsymptoteIntensity`	represents the intensity value at infinite time.
`maximum`	the maximum value that the sigmoidal function can reach.
`slope1`	the slope of the sigmoidal function at the steppest point in the exponential phase of the viral production. i.e when the intensity is increasing.
`midPoint1`	the x axis value of the steppest point in the function.
`slope2`	the slope of the sigmoidal function at the steppest point in the lysis phase. i.e when the intensity is decreasing.
`midPointDistance`	the distance between the time of steppest increase and steppest decrease in the intensity data. In other words the distance between the x axis values of arguments of slope1 and slope2.

Value

Returns the predicted intensities for the given time points with the double-sigmoidal fitted parameters for the double sigmoidal fit.

Examples

```
time=seq(3,24,0.1)

#simulate intensity data and add noise
noise_parameter=0.2
intensity_noise=stats::runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=doublesigmoidalFitFormula(time,
                                   finalAsymptoteIntensity=.3,
                                   maximum=4,
                                   slope1=1,
                                   midPoint1=7,
                                   slope2=1,
                                   midPointDistance=8)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput = normalizeData(dataInput)
parameterVector<-doublesigmoidalFitFunction(normalizedInput,tryCounter=2)


#Check the results
if(parameterVector$isThisaFit){
 intensityTheoretical=
       doublesigmoidalFitFormula(
               time,
               finalAsymptoteIntensity=parameterVector$finalAsymptoteIntensity_Estimate,
               maximum=parameterVector$maximum_Estimate,
               slope1=parameterVector$slope1_Estimate,
               midPoint1=parameterVector$midPoint1_Estimate,
               slope2=parameterVector$slope2_Estimate,
               midPointDistance=parameterVector$midPointDistance_Estimate)

 comparisonData=cbind(dataInput,intensityTheoretical)

 require(ggplot2)
 ggplot(comparisonData)+
   geom_point(aes(x=time, y=intensity))+
   geom_line(aes(x=time,y=intensityTheoretical))+
   expand_limits(x = 0, y = 0)}

if(!parameterVector$isThisaFit){print(parameterVector)}
```

doublesigmoidalFitFunction

Double sigmoidal fit function.

Description

The function fits a double sigmoidal curve to given data by using likelihood maximization (LM) and gives the parameters (maximum, final asymptote intensity, slope1, midpoint1, slope2, and mid point distance) describing the double-sigmoidal fit as output. It also provides information about goodness of fit such as AIC, BIC, residual sum of squares, and log likelihood.

Usage

```
doublesigmoidalFitFunction(dataInput, tryCounter,
  startList = list(finalAsymptoteIntensity = 0, maximum = 1, slope1 = 1,
  midPoint1 = 0.3333333, slope2 = 1, midPointDistance = 0.2916667),
  lowerBounds = c(finalAsymptoteIntensity = 0, maximum = 0.3, slope1 = 0.01,
  midPoint1 = -0.5208333, slope2 = 0.01, midPointDistance = 0.04166667),
  upperBounds = c(finalAsymptoteIntensity = 1, maximum = 1.5, slope1 = 180,
  midPoint1 = 1.145833, slope2 = 180, midPointDistance = 0.625),
  min_Factor = 1/2^20, n_iterations = 1000)
```

Arguments

`dataInput`	a data frame composed of two columns. One is for time other is for intensity. The data should be normalized with the normalizeData function first.
`tryCounter`	the number of times the data is fit via maximum likelihood.
`startList`	the initial set of parameters that algorithm tries for the fit. Where the parameters are the 'maximumValue' that represents the maximum value that the function that can take, 'slope1' represents the maximum slope on the normalized y axis at the exponential phase, 'midPoint1' represents the x axis value for the maximum slope in exponential phase, 'slope2' represents the maximum slope in the

	normalized y axis during lysis, 'midPointDistance' represents the x axis distance between the maximum slope in exponential phase and the maximum slope in lysis, 'finalAsymptoteIntensity' represents the intensity value at infinite time as the ratio with respect to maximum value reached, its is bounded between 0 and 1.
`lowerBounds`	the lower bounds for the randomly generated start parameters.
`upperBounds`	the upper bounds for the randomly generated start parameters.
`min_Factor`	defines the minimum step size used by the fitting algorithm.
`n_iterations`	define maximum number of iterations used by the fitting algorithm.

Value

Returns the fitted parameters and goodness of fit metrics.

Examples

```
time=seq(3,24,0.1)

#simulate intensity data and add noise
noise_parameter=0.2
intensity_noise=stats::runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=doublesigmoidalFitFormula(time,
                                   finalAsymptoteIntensity=.3,
                                   maximum=4,
                                   slope1=1,
                                   midPoint1=7,
                                   slope2=1,
                                   midPointDistance=8)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput = normalizeData(dataInput)
parameterVector<-doublesigmoidalFitFunction(normalizedInput,tryCounter=2)


#Check the results
if(parameterVector$isThisaFit){
    intensityTheoretical=
         doublesigmoidalFitFormula(
             time,
             finalAsymptoteIntensity=parameterVector$finalAsymptoteIntensity_Estimate,
             maximum=parameterVector$maximum_Estimate,
             slope1=parameterVector$slope1_Estimate,
             midPoint1=parameterVector$midPoint1_Estimate,
             slope2=parameterVector$slope2_Estimate,
             midPointDistance=parameterVector$midPointDistance_Estimate)

 comparisonData=cbind(dataInput,intensityTheoretical)

 require(ggplot2)
 ggplot(comparisonData)+
   geom_point(aes(x=time, y=intensity))+
   geom_line(aes(x=time,y=intensityTheoretical))+
   expand_limits(x = 0, y = 0)}

if(!parameterVector$isThisaFit){print(parameterVector)}
```

`fitFunction` *fit function.*

Description

Calls the fitting algorithms to fit the data starting from random initial parameters. Multiple attempts at fitting the data are necessary to avoid local minima.

Usage

```
fitFunction(dataInput, dataInputName = NA, model, n_runs_min, n_runs_max,
  showDetails = FALSE, randomParameter = NA, ...)
```

Arguments

`dataInput`	normalized input data that will be fitted transferred into related functions
`dataInputName`	name of data set (Default is 'NA').
`model`	type of fit function that will be used. Can be "linear", "sigmoidal", "double_sigmoidal", or "test".
`n_runs_min`	number of minimum successfull runs returned by the fitting algorithm.
`n_runs_max`	number of maximum number of times the fitting is attempted.
`showDetails`	if TRUE prints details of intermediate steps of individual fits (Default is FALSE).
`randomParameter`	a parameter needed to run the "test" model. Default is 'NA'
`...`	all other arguments that model functions ("exampleFitFunction", "lineFitFunction", "sigmoidalFitFunction", "doublesigmoidalFitFunction") may need

Value

Returns the parameters related with the curve fitted to the input data.

Examples

```
# Example 1 (test function without normalization)
# data sent to algorithm directly as data frame
# a- Generate data
time = seq(3,48,0.5)
intensity=stats::runif(length(time), 3.0, 7.5)
dataInput = data.frame(time,intensity)
# b- generate "random Parameter" for model "test"
randomParameterValue=0.7 # it should be a parameter between 0 and 1
# c- use the function "test"
parameterOutput=fitFunction(dataInput=dataInput,
                            model="test",
                            n_runs_min=5,
                            n_runs_max=15,
                            randomParameter=randomParameterValue)

# Example 2 (test function with normalization)
# data sent to algorithm after normalization
# a- Generate data
time = seq(3,48,0.5)
intensity=stats::runif(length(time), 3.0, 7.5)
dataInput = data.frame(time,intensity)
# b- normalize data
dataOutput = normalizeData(dataInput)
# c- generate "random Parameter" for model "test"
randomParameter=0.7 # it should be a parameter between 0 and 1
# d- use the function "test"
dataInput2=dataOutput
parameterOutput=fitFunction(dataInput=dataInput2,
                            model="test",
                            n_runs_min=5,
                            n_runs_max=15,
                            randomParameter=randomParameterValue)

# Example 3 (linear function without normalization)
# data sent to algorithm directly as data frame
# a- Generate data
time = seq(3,48,0.5)
intensity=stats::runif(length(time), 3.0, 7.5)
dataInput = data.frame(time,intensity)
# b- use the function "linear"
parameterOutput=fitFunction(dataInput=dataInput,
                            model="linear",
                            n_runs_min=5,
                            n_runs_max=15)


# Example 4 (linear function with normalization)
time=seq(3,24,0.5)

#simulate intensity data with noise
noise_parameter=20
```

```
intensity_noise=stats::runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=lineFitFormula(time, slope=4, intersection=-2)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput = normalizeData(dataInput)
parameterVector=fitFunction(dataInput=normalizedInput,
                            model="linear",
                            n_runs_min=5,
                            n_runs_max=15)

#Check the results
if(parameterVector$isThisaFit){
 intensityTheoretical=lineFitFormula(time,
                                     slope=parameterVector$slope_Estimate,
                                     intersection=parameterVector$intersection_Estimate)

 comparisonData=cbind(dataInput,intensityTheoretical)

 print(parameterVector$residual_Sum_of_Squares)
 require(ggplot2)
 ggplot(comparisonData)+
   geom_point(aes(x=time, y=intensity))+
   geom_line(aes(x=time,y=intensityTheoretical))+
   expand_limits(x = 0, y = 0)}

if(!parameterVector$isThisaFit){print(parameterVector)}

# Example 5 (sigmoidal function with normalization)
time=seq(3,24,0.5)

#simulate intensity data and add noise
noise_parameter=2.5
intensity_noise=stats::runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=sigmoidalFitFormula(time, maximum=4, slope=1, midPoint=8)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput = normalizeData(dataInput, dataInputName="batch_01_21_2016_samp007623")
parameterVector=fitFunction(dataInput=normalizedInput,
                            model="sigmoidal",
                            n_runs_min=20,
                            n_runs_max=500)

#Check the results
if(parameterVector$isThisaFit){
 intensityTheoretical=sigmoidalFitFormula(time,
                                          maximum=parameterVector$maximum_Estimate,
                                          slope=parameterVector$slope_Estimate,
                                          midPoint=parameterVector$midPoint_Estimate)

 comparisonData=cbind(dataInput,intensityTheoretical)
 print(parameterVector$residual_Sum_of_Squares)

 require(ggplot2)
 ggplot(comparisonData)+
   geom_point(aes(x=time, y=intensity))+
   geom_line(aes(x=time,y=intensityTheoretical),color="orange")+
   expand_limits(x = 0, y = 0)}



if(!parameterVector$isThisaFit){print(parameterVector)}

# Example 6 (doublesigmoidal function with normalization)
time=seq(3,24,0.1)

#simulate intensity data with noise
noise_parameter=0.2
intensity_noise=stats::runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=doublesigmoidalFitFormula(time,
                                   finalAsymptoteIntensity=.3,
                                   maximum=4,
                                   slope1=1,
                                   midPoint1=7,
                                   slope2=1,
                                   midPointDistance=8)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput = normalizeData(dataInput)
parameterVector=fitFunction(dataInput=normalizedInput,
                           dataInputName="batch_01_21_2016_samp007623",
                           model="doublesigmoidal",
                           n_runs_min=20,
                           n_runs_max=500,
                           showDetails=FALSE)


#Check the results
if(parameterVector$isThisaFit){
 intensityTheoretical=
       doublesigmoidalFitFormula(
               time,
               finalAsymptoteIntensity=parameterVector$finalAsymptoteIntensity_Estimate,
               maximum=parameterVector$maximum_Estimate,
               slope1=parameterVector$slope1_Estimate,
               midPoint1=parameterVector$midPoint1_Estimate,
               slope2=parameterVector$slope2_Estimate,
               midPointDistance=parameterVector$midPointDistance_Estimate)

 comparisonData=cbind(dataInput,intensityTheoretical)

 require(ggplot2)
 ggplot(comparisonData)+
   geom_point(aes(x=time, y=intensity))+
   geom_line(aes(x=time,y=intensityTheoretical),color="orange")+
   expand_limits(x = 0, y = 0)}

if(!parameterVector$isThisaFit){print(parameterVector)}
```

`lineFitFormula` *Line fit formula.*

Description

Calculates intesities for given time points (x) by using line fit model and parameters (slope and intersection).

Usage

```
lineFitFormula(x, slope, intersection)
```

Arguments

`x`	the "time" (time) column of the dataframe.
`slope`	the slope of the line.
`intersection`	the intensity intersection point of the line when time is zero in the time-intensity graph.

Value

Returns the predicted intensities for given time points in the line fit model for given slope and intersection values.

Examples

```
time=seq(3,24,0.5)

#simulate intensity data and add noise
noise_parameter=.2
intensity_noise=stats::runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=lineFitFormula(time, slope=4, intersection=-2)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput = normalizeData(dataInput)
parameterVector<-lineFitFunction(normalizedInput,tryCounter=2)

#Check the results
if(parameterVector$isThisaFit){
 intensityTheoretical=lineFitFormula(time,
                                    slope=parameterVector$slope_Estimate,
                                    intersection=parameterVector$intersection_Estimate)
 comparisonData=cbind(dataInput,intensityTheoretical)

 print(parameterVector$residual_Sum_of_Squares)

 require(ggplot2)
 ggplot(comparisonData)+
   geom_point(aes(x=time, y=intensity))+
   geom_line(aes(x=time,y=intensityTheoretical))+
   expand_limits(x = 0, y = 0)}

if(!parameterVector$isThisaFit){print(parameterVector)}
```

`lineFitFunction` *Linear fit function.*

Description

Fits a linear model to a given data by using likelihood maximization (LM) and gives the parameters (slope and intersection) describing the line as output. It also provides information about the goodness of fit such as AIC, BIC values, residual sum of squares, and log likelihood.

Usage

```
lineFitFunction(dataInput, tryCounter, startList = list(slope = 0,
  intersection = 1), lowerBounds = c(-100, -1000), upperBounds = c(100,
  1000), min_Factor = 1/2^20, n_iterations = 500)
```

Arguments

`dataInput`	a data frame composed of two columns. One is for time other is for intensity. Should be normalized data generated by normalizeData.
`tryCounter`	the number of times the data is fit via maximum likelihood.
`startList`	the initial set of parameters that the fitting algorithm tries, these parameters are the slope and the y intercept.
`lowerBounds`	the lower bounds for the randomly generated start parameters, these parameters are the slope and the y intercept.
`upperBounds`	the upper bounds for the randomly generated start parameters, these parameters are the slope and the y intercept.
`min_Factor`	minimum step size used by the fitting algorithm.
`n_iterations`	maximum number of iterations used by the fitting algorithm.

Value

Returns fitted parameters for lineFit. The slope, intersection, and goodness of fit metrics.

Examples

```
time=seq(3,24,0.5)

#intensity with Noise
noise_parameter=.2
intensity_noise=stats::runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=lineFitFormula(time, slope=4, intersection=-2)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput = normalizeData(dataInput)
parameterVector<-lineFitFunction(normalizedInput,tryCounter=2)

#Check the results
if(parameterVector$isThisaFit){
 intensityTheoretical=lineFitFormula(time,
                                    slope=parameterVector$slope_Estimate,
                                    intersection=parameterVector$intersection_Estimate)

 comparisonData=cbind(dataInput,intensityTheoretical)

 print(parameterVector$residual_Sum_of_Squares)

 require(ggplot2)
 ggplot(comparisonData)+
   geom_point(aes(x=time, y=intensity))+
   geom_line(aes(x=time,y=intensityTheoretical))+
   expand_limits(x = 0, y = 0)}

if(!parameterVector$isThisaFit){print(parameterVector)}
```

`normalizeData` *Normalization of given data*

Description

Maps the given time-intensity data into a recaled frame where time is between [x,1] and intensity is between [0,1].

Usage

```
normalizeData(dataInput, dataInputName = NA)
```

Arguments

`dataInput`	a data frame composed of two columns. One is for time other is for intensity.
`dataInputName`	experiment name (Default is 'NA').

Value

Function returns another data frame, scaling factors and scaling constants for time and intensity. The other data frame includes 2 columns one is for normalized time and the other is for noralized intensity. The whole time is distributed between 0 and 1 and similarly the whole intensity is distributed between 0 and 1. The time and intensity constants and scaling factors are the parameters to transform data from given set to scaled set.

Examples

```
# generateRandomData
time = seq(3,48,0.5)
intensity=runif(length(time), 3.0, 7.5)
dataInput = data.frame(time,intensity)

# Normalize Data
dataOutput = normalizeData(dataInput,dataInputName="batch_01_21_2016_samp007623")
```

`numericalReCalculation` *Calls fitting algorithm with different initial parameters.*

Description

Calls the fitting algorithms and fits the data with random initial parameters.

Usage

```
numericalReCalculation(parameterVector, stepSize = 1e-05)
```

Arguments

`parameterVector`	output of fitFunction or data frame that gives the variables related with double sigmoidal fit.
`stepSize`	step size used by the fitting algorithm.

Value

Returns the parameters related with fitted curve to input data.

Examples

```
time=seq(3,24,0.1)

#simulate intensity data and add noise
noise_parameter=0.2
intensity_noise=runif(n = length(time),min = 0,max = 1)*noise_parameter
```

```
initialParameters=data.frame(dataScalingParameters.timeRatio=24,
                                 finalAsymptoteIntensity_Estimate=.3,
                                 maximum_Estimate=4,
                                 slope1_Estimate=1,
                                 midPoint1_Estimate=7,
                                 slope2_Estimate=1,
                                 midPointDistance_Estimate=8,
                                 model="doublesigmoidal")

initialParameters = numericalReCalculation(initialParameters, stepSize=0.00001)

intensity=
 doublesigmoidalFitFormula(
         time,
         finalAsymptoteIntensity=initialParameters$finalAsymptoteIntensity_Estimate,
         maximum=initialParameters$maximum_Estimate,
         slope1=initialParameters$slope1_Estimate,
         midPoint1=initialParameters$midPoint1_Estimate,
         slope2=initialParameters$slope2_Estimate,
         midPointDistance=initialParameters$midPointDistance_Estimate)


intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizeInput = normalizeData(dataInput)
parameterVector=fitFunction(dataInput=normalizeInput,
                           dataInputName="batch_01_21_2016_samp007623",
                           model="doublesigmoidal",
                           n_runs_min=20,
                           n_runs_max=500,
                           showDetails=FALSE)


dataOutput2 = numericalReCalculation(parameterVector, stepSize=0.00001)
```

printInfectionCurves *Prints infection curves.*

Description

Generates figures using ggplot that shows the input data and the fitted curves.

Usage

```
printInfectionCurves(dataInput, sigmoidalFitVector = NULL,
  doubleSigmoidalFitVector = NULL, showParameterRelatedLines = FALSE)
```

Arguments

`dataInput`	a data frame composed of two columns. One is for time and the other is for intensity. Should be normalized data generated by normalizeData function.
`sigmoidalFitVector`	the output of sigmoidalFitFunction. Default is NULL.
`doubleSigmoidalFitVector`	the output of double sigmoidal fit function. Default is NULL.
`showParameterRelatedLines`	if equal to TRUE, figure will show parameter related lines on the curves. Default is FALSE.

Value

Returns infection curve figures.

Examples

```
time=seq(3,24,0.1)

#simulate intensity data and add noise
noise_parameter=0.2
intensity_noise=runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=doublesigmoidalFitFormula(time,
                                   finalAsymptoteIntensity=.3,
                                   maximum=4,
                                   slope1=1,
                                   midPoint1=7,
                                   slope2=1,
                                   midPointDistance=8)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput = normalizeData(dataInput,dataInputName="batch_01_21_2016_samp007623")

# Do the sigmoidal fit
sigmoidalModel=fitFunction(dataInput=normalizedInput,
                                    model="sigmoidal",
                                    n_runs_min=20,
                                    n_runs_max=500,
                                    showDetails=FALSE)

# Do the double sigmoidal fit
doubleSigmoidalModel=fitFunction(dataInput=normalizedInput,
                                          model="doublesigmoidal",
                                          n_runs_min=20,
                                          n_runs_max=500,
                                          showDetails=FALSE)

doubleSigmoidalModel = numericalReCalculation(doubleSigmoidalModel,
                                                        stepSize=0.00001)
fig01=printInfectionCurves(dataInput=normalizedInput)
```

```
print(fig01)

fig02=printInfectionCurves(dataInput=normalizedInput,
                           sigmoidalFitVector=sigmoidalModel)
print(fig02)

fig03=printInfectionCurves(dataInput=normalizedInput,
                           doubleSigmoidalFitVector=doubleSigmoidalModel)
print(fig03)

fig04=printInfectionCurves(dataInput=normalizedInput,
                           sigmoidalFitVector=sigmoidalModel,
                           doubleSigmoidalFitVector=doubleSigmoidalModel)
print(fig04)

fig05=printInfectionCurves(dataInput=normalizedInput,
                           doubleSigmoidalFitVector=doubleSigmoidalModel,
                           showParameterRelatedLines=TRUE)
print(fig05)

fig06=printInfectionCurves(dataInput=normalizedInput,
                           sigmoidalFitVector=sigmoidalModel,
                           showParameterRelatedLines=TRUE)
print(fig06)
```

`sameSourceDataCheck` *Check is data came from the same source.*

Description

Checks if the provided data and models came from same source by looking to ".dataInputName" columns of the inputs.

Usage

```
sameSourceDataCheck(dataInput, sigmoidalFitVector, doubleSigmoidalFitVector)
```

Arguments

`dataInput`	a data frame composed of two columns. One is for time and the other is for intensity. Should be normalized data generated by normalizeData.
`sigmoidalFitVector`	is the output of sigmoidalFitFunction. Default is NULL.
`doubleSigmoidalFitVector`	is the output of double sigmoidal fit function. Default is NULL.

Value

Returns TRUE if models can from same source, FALSE otherwise.

`sigmoidalFitFormula` *sigmoidalFitFormula*

Description

Calculates intesities for given time points (x) by using sigmoidal fit model and parameters (maximum, slope, and midpoint).

Usage

```
sigmoidalFitFormula(x, maximum, slope, midPoint)
```

Arguments

`x`	the "time" (time) column of the dataframe.
`maximum`	the maximum value that the sigmoidal function can reach.
`slope`	the slope of the sigmoidal function at the steppest point.
`midPoint`	the x axis value of the steppest point in the function.

Value

Returns the predicted intensities for given time points with the given sigmoidal fit parameters.

Examples

```
time=seq(3,24,0.5)

#simulate intensity data and add noise
noise_parameter=0.1
intensity_noise=stats::runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=sigmoidalFitFormula(time, maximum=4, slope=1, midPoint=8)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput = normalizeData(dataInput)
parameterVector<-sigmoidalFitFunction(normalizedInput,tryCounter=2)

#Check the results
if(parameterVector$isThisaFit){
intensityTheoretical=sigmoidalFitFormula(time,
                                        maximum=parameterVector$maximum_Estimate,
                                        slope=parameterVector$slope_Estimate,
                                        midPoint=parameterVector$midPoint_Estimate)
comparisonData=cbind(dataInput,intensityTheoretical)

require(ggplot2)
ggplot(comparisonData)+
 geom_point(aes(x=time, y=intensity))+
 geom_line(aes(x=time,y=intensityTheoretical))+
 expand_limits(x = 0, y = 0)}

if(!parameterVector$isThisaFit){print(parameterVector)}
```

`sigmoidalFitFunction` *Sigmoidal fit function*

Description

Fits a sigmoidal curve to given data by using likelihood maximization (LM) and gives the parameters (maximum, slope and midpoint) describing the sigmoidal fit as output. It also provides information about the goodness of fit such as AIC, BIC, residual sum of squares, and log likelihood.

Usage

```
sigmoidalFitFunction(dataInput, tryCounter, startList = list(maximum = 1,
  slope = 36, midPoint = 0.3333333), lowerBounds = c(maximum = 0.3, slope =
  1e-05, midPoint = 0.3125 - 0.8333333), upperBounds = c(maximum = 1.5, slope
  = 180, midPoint = 0.3125 + 0.8333333), min_Factor = 1/2^20,
  n_iterations = 1000)
```

Arguments

`dataInput`	a data frame composed of two columns. One is for time and the other is for intensity. Should be normalized data generated by normalizeData.
`tryCounter`	the number of times the data is fit using the fitting algorithm.
`startList`	the initial set of parameters that algorithm tries to fit. The parameters are 'maximumValue' that represents the maximum value that the function can take, 'slopeValue' that represents the slope in normalized y axis, and 'midPointValue' that represents the midpoint.
`lowerBounds`	the lower bounds for the randomly generated start parameters.
`upperBounds`	the upper bounds for the randomly generated start parameters.
`min_Factor`	the minimum step size in the iterations used by the fitting algorithm.
`n_iterations`	the maximum number of iterations used by the fitting algorithm.

Value

Returns fitted parameters for the sigmoidal model.

Examples

```
time=seq(3,24,0.5)

#simulate intensity data and add noise
noise_parameter=0.1
intensity_noise=stats::runif(n = length(time),min = 0,max = 1)*noise_parameter
intensity=sigmoidalFitFormula(time, maximum=4, slope=1, midPoint=8)
intensity=intensity+intensity_noise

dataInput=data.frame(intensity=intensity,time=time)
normalizedInput = normalizeData(dataInput)
parameterVector<-sigmoidalFitFunction(normalizedInput,tryCounter=2)

#Check the results
if(parameterVector$isThisaFit){
intensityTheoretical=sigmoidalFitFormula(time,
                                       maximum=parameterVector$maximum_Estimate,
                                       slope=parameterVector$slope_Estimate,
                                       midPoint=parameterVector$midPoint_Estimate)

comparisonData=cbind(dataInput,intensityTheoretical)

require(ggplot2)
ggplot(comparisonData)+
 geom_point(aes(x=time, y=intensity))+
 geom_line(aes(x=time,y=intensityTheoretical))+
 expand_limits(x = 0, y = 0)}

if(!parameterVector$isThisaFit){print(parameterVector)}
```

`unnormalizeData` *Unnormalization of given data*

Description

Maps the given time-intensity data into a recaled frame where time is between [0,1] and similarly intensity is between [0,1].

Usage

```
unnormalizeData(dataInput)
```

Arguments

`dataInput`	a list file composes of two parts First part is the data that will be unnormalized, which is a data frame composed of two columns. One is for time and the other is for intensity Second part is the scaling parameters of the data which is a vector that has three components. The first one of them is related with time and last two of them are related with intensity. The second value represents the min value of the intensity set. First and third values represent the difference between max and min value in the relevant column.

Value

Returns a data frame, scaling factors and scaling constants for time and intensity. The other data frame includes 2 columns one is for normalized time and the other is for noralized intensity. The whole time is distributed between 0 and 1 and similarly the whole intensity is distributed between 0 and 1. The time and intensity constants and scaling factors are the parameters to transform data from given set to scaled set.

Examples

```
# generateRandomData
time = seq(3,48,0.5)
intensity=runif(length(time), 3.0, 7.5)
dataInput = data.frame(time,intensity)
# Normalize Data
dataOutput = normalizeData(dataInput)
dataInput2=dataOutput
# Un Normalize it
dataOutput2 = unnormalizeData(dataInput2)
```