Package 'dsmodels'

July 22, 2017

Type Package

Title A Language to Facilitate Simulation and Visualization of Two-Dimensional Dynamical Systems

Version 1.1.0

Description An expressive language to facilitate simulation and visualization of two-dimensional dynamical systems. The basic elements of the language are a model wrapping around a function(x,y) which outputs a list(x = xprime, y = yprime), and a range. The language supports three types of visual objects: visualizations, features, and backgrounds. Visualizations, including dots and arrows, depict the behavior of the dynamical system over the entire range. Features display user-defined curves and points, and their images under the system. Backgrounds define and color regions of interest, such as basins of attraction. The language can also approximate attractors and their basins through simulation.

License GPL-2 | file LICENSE

Depends R (>= 3.1.0)

LazyData TRUE

RoxygenNote 6.0.1

Collate 'dsproto.R' 'axeslabel.R' 'dsarrows.R' 'dscurve.R' 'dsdots.R' 'dsmodel.R' 'dspoint.R' 'dsrange.R' 'dsregion.R' 'dssimulation.R' 'features.R' 'modelConstruction.R'

Imports shape, graphics, grDevices, latex2exp, pryr

BugReports https://github.com/Trinity-Automata-Research/dsmodels/issues

URL https://github.com/Trinity-Automata-Research/dsmodels

NeedsCompilation no

Author Charles Stein [aut, cre], Seth Fogarty [aut]

Maintainer Charles Stein <cstein1@trinity.edu>

Repository CRAN

Date/Publication 2017-07-22 06:37:56 UTC

R topics documented:

`dsarrows` *Add a visualization of the system using arrows*

Description

The visualization displays the movement of a uniform array of points under the function defined by the model as arrows. The base of the arrow is placed on each discretized point. The tip of the head of the arrow points in the direction of the image of that point. In order to use this visualization, a discretization parameter must be set either on the range or as a parameter.

Usage

```
dsarrows(scale = 0.9, col = "blue", angle = 30, type = "simple",
  length = NULL, iters = 1, head.length = 0.2, discretize = NULL,
  behind = TRUE, crop = FALSE, ...)
```

Arguments

`scale`	Changes the size of the arrow to a user-specified scale relative to the discretization parameter.
`col`	Colors the arrows.
`angle`	Specifies the angle of the head of the arrow. Passed directly into `shape`'s `Arrows` function.
`type`	Determines type of arrow. Accepted values are identical to acceptable values of arr.type in the "shape" library.
`length`	A non-scaled length of arrow tail. Causes `scale` to be ignored.
`iters`	Allows user to point the head of each arrow towards the result of a specified number of iterations of the function.
`head.length`	Changes the size of the arrowhead. Passed directly to `Arrows` as arr.length. When the range is large, a smaller value produces reasonable results.

discretize	Overrides the discretization parameter defined in the range.
behind	Forces the arrows to be a background object for the purposes of layering.
crop	If crop==TRUE then arrows whose image falls outside the range are not displayed.
...	Further graphical parameters passed to Arrows

Details

The arrows are scaled according to the discretization. If the discretization is too fine (small), the arrows may seem crowded. It is suggested to try a coarser (larger) discretization size before modifying the size of the arrows.

The most common way to invoke dsarrows is to simply add model + dsarrows(), where model is a variable corresponding with the dsmodel class.

See Also

dsdots

Examples

```
library(dsmodels)

fun <- function(X,Y) {
  list(
    X/exp(Y),
    Y/exp(X)
  )
}
model <- dsmodel(fun, title = "Blue Arrows")
range <- dsrange(x = -2:3, y = -2:3, discretize = .5)
model + range + dsarrows()

dsmodel(fun, title = "Spaced Purple Arrows") +
 dsrange(x = -2:3, y = -2:3, discretize = .5) +
 dsarrows(discretize = 1, col = "magenta")
```

---

dscurve *Parametric curves or a graph of functions*

Description

This function takes a description of a curve and creates an object displaying the curve, and optionally it's behavior throughout iterations of the system. Functions can be provided as expressions of x, for graphing curves, or t, for parametric curves. The curve is defined either by the graph of a single function or a pair of parametric equations. By default, rendered with the lines function.

Usage

```
dscurve(fun, yfun = NULL, col = "black", image = NULL, lwd = 3,
  n = NULL, iters = 0, crop = TRUE, tstart = 0, tend = 1,
  discretize = FALSE, xlim = NULL, ...)
```

Arguments

fun	A function. If yfun is provided, this is the x-equation of the parametric equations. If not, the function's graph is rendered. See sections describing graphs and parameteric equations for more info.
yfun	The y-equation of the parameteric equations. See sections describing parametric equations for more info.
col	The color of the original curve, as a string.
image	A single color as a string, or a vector of colors as a string. See details for more information.
lwd	Line width expressed as a double. Only used if discretize is not set.
n	The number of points that will be calculated. Defaults to the dsrange's renderCount. n is used to interact with discretize.
iters	Determines the number of iterations of the function when making a color gradient. Use col = color1, image = color2, iters = n to create a gradient of colors between color1 and color2. See details for more information.
crop	If crop==TRUE, the original curve and all iterations are cropped to the range.
tstart	Only used for parametric curves. The minimum input for both functions. Default 0.
tend	Only used for parametric curves. The maximum input for the functions. Default 1.
discretize	Set discretize=TRUE to display the calculated points, instead of connecting them as a curve: the curve is displayed with points instead of lines.
xlim	Only used for the graph of a function. Determines the range of x values for which the function is plotted. Defaults to the x limits of the model's dsrange.
...	Further graphical parameters passed to lines or points.

The graph of a function

If the parameter yfun is not provided, then dscurve contains the curve of points (x,fun(x)). The inputs to fun are n points between the maximum dsrange's x limits, but can be overwritten with the xlim parameter. fun can either be any function of a single parameter, or an expression with exactly x as the free variable.

Parametric equations

If the parameter fun and yfun are both provided, dscurve contains the parametric curve described by the functions. The function is calculated at n points ranging from tmin to tmax. fun and yfun can either be any function of a single parameter, or an expression with exactly t as the free variable.

Images of curves

The dscurve object begins with an initial curve. Images of the curve may be displayed in three ways. If the image parameter is a single color and iters is not set, then dscurve will calculate and display the image of the curve under the model's function in that color.

If the image parameter is a vector of k colors, then dscurve calculates and displays k successive images of the curve using those colors. The string "NA" may be used to avoid displaying an iteration.

If the image parameter is a single color and iters is defined, then iters successive images are displayed, using a gradient between col and image.

In most cases, rather than specifying col and image separately, they may be combined into a single vector.

See Also

dspoint

Examples

```
library(dsmodels)

fun <- function(X,Y) {
  list(
    X/exp(Y),
    Y/exp(X)
  )
}

model <- dsmodel(fun, title = "Points on a One-Dimensional Curve")
range <- dsrange(-2:2,-2:2, discretize = 0.5)

# Add the graph of a function and its image in blue.
graphcrv <- dscurve(function(x) x^2,
                    col = "orange",
                    image = "blue",
                    discretize = TRUE,
                    xlim = c(-2,2))
model + range + graphcrv
# Add the graph of expression of x.
model + dscurve(x^2+1, col="yellow")

# Create a parametric curve with image iterations red then green.
paramcrv <- dscurve(function(t) t^2, function(t) t,
                    image = c("red", "green"),
                    tstart = -2, tend = 2)
dsmodel(fun, "A Parametric Curve and Iterations of that Curve") +
  dsrange(-2:2, -2:2, discretize = 0.5) +
# A parametic curve defined by expressions of t.
  paramcrv + dscurve(4*t-2,4*t-2,col="blue")
```

`dsdots` *Adds a visualization of the system using dots*

Description

The visualization displays a uniform array of points and their images under the function defined by the model as dots. Multiple iterations of the function may be visualized. In order to use this visualization, a discretization parameter must be provided in either the range or to the `dsdots` object.

Usage

```
dsdots(col = "black", image = "", iters = 1, discretize = NULL,
  crop = TRUE, size = 0.37, behind = TRUE, ...)
```

Arguments

`col`	A string specifying the color of the initial discretized points.
`image`	Sets the color of the final image of the discretized points. See details.
`iters`	Determines the number of iterations of the function when making a color gradient. Use `col = color1, image = color2, iters = n` to create a gradient of colors between color1 and color2. See details for more information.
`discretize`	Overrides the discretization parameter defined in the range.
`crop`	If `crop==TRUE`, remove points found outside of the range from being rendered and iterated upon.
`size`	Determines the display size of each dot.
`behind`	Sets the dots as a background object for layering purposes.
`...`	Extra graphical parameters

Details

The `col` parameter defines the color of the initial, discretized, points. There are three modes of operation. If the `image` parameter parameter is a single color, and `iters` is not set, then the image of each point is displayed as a dot of color `image`.

Alternately, `image` can be set as a vector of colors (for example: `image = c("red", "NA", "green")`). In this case the function is applied iteratively to the points a number of times equal to the length of the vector. The initial points are displayed using `col`. Each iteration is displayed by the corresponding color. `"NA"` may be used to not display that iteration. to specify the number of iterations of the function to apply.

Finally, if iters is set to a numeric value greater than 1, a color gradient is used to display iterations. In this case both `col` and `image` should be single colors. The function is applied `iters` time, with each iteration being colored along the gradient from `col` to `image`.

In most cases, rather than specifying `col` and `image` separately, they may be combined into a single vector.

See Also

dsarrows

dspoint

Examples

```
library(dsmodels)

fun <- function(X,Y) {
  list(
    X/exp(Y),
    Y/exp(X)
  )
}

model <- dsmodel(fun, title = "View of the Discretized Field")
range <- dsrange(-2:2,-2:2, discretize = 0.09)

# To view the discretized field, simply add dsdots() to your model
model + range + dsdots()

# To view a gradient with a certain amount of iterations,
# specify the image and the amount of iterations in the image
# and iters parameters, respectively.
dsmodel(fun, title = "Gradient of Iterations from Blue to Red") +
   dsrange(-2:2,-2:2, discretize = 0.09) +
   dsdots(col = "blue", image = "red", iters = 3)

# Set color to "NA" if you wish for the specified iteration to not
# appear in the image
dsmodel(fun, title = "Display Only the Third Iteration") +
   dsrange(-2:2,-2:2, discretize = 0.09) +
   dsdots(col = "NA", image = c("NA","blue"), size = 1)
```

---

dsmodel *Defines a model object encapsulating a dynamical system*

Description

To begin, define a two dimensional function that outputs a two-dimensional list. For instance: `fun <- function(x,y) list(x,y)`. Then `dsmodel(fun)` will initialize the model. **Make sure that the function defining your model indeed outputs a `list`.** The model is used to hold the data for graphics. To display a desired graphic, add the corresponding feature of type "dsproto" to your model.

Usage

```
dsmodel(fun, title = "", display = TRUE)
```

Arguments

fun	Function with two inputs and two outputs which defines the dynamical system. The output should be a list, preferably with field names x and y.
title	A string title for the graph. Text can be input in the form of pseudo-LaTeX code within quotes. See TeX for more details.
display	If set to FALSE, the model will be drawn only when the user calls `MODELNAME`$display(). Otherwise, the model will be drawn with every dsmodels object added after the dsrange is added.

Details

Models are constructed incrementally using the + operator to add features to the existing dsmodel object. A dsrange must be one of the objects added to a model.

Methods

dsmodel objects support the following methods, which may be helpful for advanced users.

dsmodel$points(format="list", filter="all") returns a list of the points in the model.

filter Valid values are "all", "fixed", "attractor", and "sim". "sim" returns only points generated by simattractors.

format If "list", return a list with x, y, col, and ind fields holding the coordinates, color, and index. Other formats are "objects", returning a vector of dspoint objects, and "pairs", returning a list of pairs of coordinates.

dsmodel$display() forces the model to re-render the plot from scratch. Primarily useful if display=TRUE was set.

dsmodel$basins() returns a list of which fixed points have a basin. This requires simbasins() to have been composed with the model, and is primarily useful when testing if a dynamical system is globally stable. In that case, the method will return a list of length 1. The list will contain the indices of the fixed points, as given in dsmodel$points(format="list", filter="attractor"). An index of 0 means that some points never moved within epsilon of an attractor.

dsmodel$sim.is.stable() attempts to determine if the system is stable by simulation. If no attractors have been composed with the model, simattractors() is composed with defaults. If simbasins has not been composed with the model, it is be composed with defaults. If every point is drawn to a single attractor, the system has been deemed stable. Note that boundary points on the range will not be tested.

See Also

dsrange

Examples

```
library(dsmodels)

fun <- function(X,Y) {
  list(
```

```
    x = X/exp(Y),
    y = Y/exp(X)
  )
}
# Add dsRange to see the actual range.
model <- dsmodel(fun)

dsmodel(function(x,y) {
  list(
    x = x^2,
    y = x/(y+1)
  )
}, title = "Another function showing $f(x)=x^{\\alpha}$!")
```

---

dspoint *Individual points and their images*

Description

pnt and dspoint are the same function. This function takes a single point and creates an object displaying the point, and optionally it's behavior throughout iterations of the system.

Usage

```
dspoint(x, y, label = "", pch = 21, size = 2, col = "blue",
  regionCol = NULL, image = "", offset = NULL, display = TRUE,
  fixed = FALSE, iters = 0, attractor = FALSE, crop = TRUE,
  artificial = FALSE, ...)
```

Arguments

x	The x-coordinate of the point.
y	The y-coordinate of the point.
label	A string label. Text can be input in the form of pseudo-LaTeX code within quotes. See TeX for more details. Text will appear above the dot by default. Please see the offset parameter to adjust.
pch	Plotting 'character' or symbol to use, default is 21 (filled circle). See help(pch) for details.
size	Determines the size of the point.
col	A string color for the point. Use "NA" or "" to hide the point. See also display.
regionCol	An alternate color used to define the color of the region for simbasins(). Defaults to col or col[1].
image	A single color as a string, or a vector of colors as a string. See details for more information.
offset	This will offset the label. Enter as c(x, y). Defaults to an automatic scale dependent on the dsrange's y axis size.

display	Set display = FALSE to hide the dot, but still add to your system. Mostly useful for simbasins().
fixed	A flag to declare a fixed point. The image of any fixed point is should be the original point.
iters	Determines the number of iterations of the function when making a color gradient. Use col = color1, image = color2, iters = n to create a gradient of colors between color1 and color2. See details for more information.
attractor	A flag to delcare a point as an attractor: a fixed point for the function that other points converge to. Used in simbasins().
crop	If crop==TRUE points outside of defined range will be cropped, and no further images will be calculated.
artificial	For internal use.
...	Extra graphical parameters to be sent through points

Images of the point

The dspoint object begins with an initial point. Images of the point may be displayed in three ways.

If the image parameter is a single color and iters is not set, then dspoint will calculate and display the image of the point under the model's function in that color.

If the image parameter is a vector of k colors, then dspoint calculates and displays k successive images of the point using those colors. The string "NA" may be used to avoid displaying an iteration.

If the image parameter is a single color and iters is defined, then iters successive images are displayed, using a gradient between col and image.

In most cases, rather than specifying col and image separately, they may be combined into a single vector.

Examples

```
library(dsmodels)

fun <- function(X,Y) {
  list(
    X/exp(Y),
    Y/exp(X)
  )
}

model <- dsmodel(fun, title = "A Single Point")
model + dsrange(3,3, discretize = .09) +
dspoint(1,2, col = "magenta")

dsmodel(fun, title = "A Point and a Label") +
 dsrange(3,3, discretize = .09) +
 dspoint(2.2, 2.1, label = "$x^{\\alpha}$", col = "green")

dsmodel(fun, title = "A Point and Iterations of that Point") +
```

```
 dsrange(3,3, discretize = .09) +
dspoint(1,1, col = "red", image = c("orange","yellow"))

dsmodel(fun, title = "Iterations of a Point over a Color Gradient") +
 dsrange(3,3, discretize = .09) +
dspoint(0.2, 0.5, image = "pink", iters = 3, col = "grey")
```

---

dsrange *Range of inputs for a model*

Description

dsrange creates a discrete or continuous range for the model to be computed over. Points that fall outside the range will be discarded in all features.

Usage

```
dsrange(x, y, discretize = 0, renderCount = 101, axes = TRUE,
  frame.plot = TRUE, ...)
```

Arguments

x	Specifies the minimum and maximum for the x axis. If only one value is specified, it is used as the maximum, and the minimum will default to 0. If a collection of values are provided, the minimum and maximum are used as the range.
y	Specifies the minimum and maximum for the y axis. If only one value is specified, it is used as the maximum, and the minimum will default to 0. If a collection of values are provided, the minimum and maximum are used as the range.
discretize	If a value is provided, the field is discretized into an array of points. The value specifies the distance between each point. This becomes the default when displaying dsarrows or dsdots. The number of points in the field is defined by:
renderCount	The number of points that a curve will be computed at when being displayed. Default 101.
axes	If FALSE, the axes will not be drawn. Defaults to TRUE.
frame.plot	If FALSE, the frame of the plot will not be drawn. Defaults to TRUE.

$$(xmax - xmin + 1)(ymax - ymin + 1)/discretize.$$

...	Further fields for the dsrange object.

Details

You may either specify a numeric x and y, in which case 0 is the lower bound and that value is the upper bound; or a range of values, in which case the min and the max of the range will be used. To specify a range from min to max, use either c(min,max) or min:max.

See Also

dsmodel

dsarrows

dsdots

Examples

```
fun <- function(X,Y) {
  list(
    X/exp(Y),
    Y/exp(X)
  )
}
model <- dsmodel(fun, title = "A range with no features!")
#Since no features are added, only the area and title are displayed.
model + dsrange(3, 3, discretize = .09)
```

---

dsregion *Colored polygonal region*

Description

Friendly function to create a polygon corresponding with given values of the polygon's corners. The polygon is then colored automatically.

Usage

```
dsregion(..., col = "yellow", border = NA, behind = TRUE)

dspolygon(x, y, col = "yellow", border = NA, behind = TRUE)
```

Arguments

...	Takes points which will act as corners. See example and details for usage.
col	The color of the polygon
border	The color of the border of the polygon.
behind	Forces the polygon to be a background object for the purposes of layering.
x	A numeric vector containing the x-values of each corner.
y	A collection (for example: c(1,2,3)) of points on the y-axis which correspond to the corners of the polygon. Each member of the y parameter corresponds with a member of the x parameter with the same index.

Calling dspolygon Correctly

dspolygon takes the x and y points similar to the default polygon function. The x parameter takes a numeric vector containing the x-values of each corner. The y parameter takes a numeric vector containing the y-values of each corner. The x and y coordinates of the corners of the polygon will be the pairs made from the x and y parameters with equal indices.

Calling dsregion Correctly

The ... parameter in dsregion can take multiple dspoints, pnts, or simply vectors each containing two points c(x,y). See the examples if clarification is needed.

See Also

dspoint

simattractors

simbasins

Examples

```
library(dsmodels)

fun <- function(X,Y) {
  list(
    X/exp(Y),
    Y/exp(X)
  )
}
model <- dsmodel(fun, title = "Regions!")
range <- dsrange(3, 3, discretize = .09)
model+range

# dspolygon usage
model + dspolygon(x = c(-.05,3,3),
                  y = c(0,0,3),
                  col = "yellow")
a <- dspoint(3,3)
b <- dspoint(2.5,3)
c <- dspoint(2,2)

# Different inputs for dsregion shown below

model + dsregion(a, b, c, col = "orange") +
dsregion(pnt(0,0),pnt(1,0),pnt(1,1),pnt(0,1), col = "green") +
dsregion(c(1,1),c(2,1),c(2,2),c(1,2), c(1.8,1.5), col = "magenta")
```

`simattractors` *Determine the attractors of a model through simulation*

Description

Attempts to determine the attractors of a model. The space is discretized into initial points, and repeated iteration of the model's function is used to approximate the attractors. It is possible that non-attractor fixpoints will be found by accident. The function is iterated until the points move less than `tolerance` (default `sqrt(.Machine$double.eps)`) between iterations. The color of each point is drawn from the `col` parameter. If the number of points exceeds the size of col, or col is not defined, then reasonable defaults are used instead. The attractors are `dspoints` that are added to the model.

Usage

```
simattractors(discretize = NULL, xlim = NULL, ylim = NULL, stride = 8,
  iters = 1e+18, epsilon = NULL, tolerance = sqrt(.Machine$double.eps),
  cols = NULL)
```

Arguments

`discretize`	The space between initial points. If not set, the discretization of the range is used. May be set separately from the discretization of the range without overwriting.
`xlim`	The range of x values to search for attractors. Defaults to the limits of the range.
`ylim`	The range of y values to search for attractors. Defaults to the limits of the range.
`stride`	The number of times the function is applied before movement is checked: in essence finding the attractors for `f^stride`. For non-periodic dynamical systems, this is merely an efficiency concern. For systems with periodic attractors with a period that is a factor of that is a factor of `stride`, this may identify each of the points in the orbit. A warning will be issued if a point is only stable in `f^stride`, and not in `f`. Default 8.
`iters`	The maximum number of iterations to use. Points that still move greater than `tolerance` will result in a warning and will be discarded. Can be disabled by setting to `Inf` or 0. Default `1e+18`.
`epsilon`	The distance at which two points are considered to be the same attractor. Defaults to `discretize^2`.
`tolerance`	A, usually smaller, distance at which a point is considered to have stopped moving. Defaults to `sqrt(.Machine$double.eps)`.
`cols`	The colors of the attractors. If insufficient not provided, reasonable defaults are used. Generally (but not always) proceeds left to right, then bottom to top.

See Also

dspoint

dsregion

dspolygon

simbasins

Examples

```
model <- dsmodel(function(X0,Y0) {
list(X0*exp(2.6-X0-6.45/(1+4.5*X0)),
     Y0*exp(2.6-Y0-0.15*X0-6.25/(1+4.5*Y0)))
})

model + dsrange(5,5,0.09) + simattractors(discretize=0.02)
```

---

simbasins *Find basins of attraction by simulation*

Description

Attempts to determine which areas of the range are drawn to which attractors by simulation. The attractors must be added to the model before `simbasins()` is added. The space is discretized into squares, and repeated iteration of the model's function is used to determine which attractor the middle of each square tends towards. The square is then given the color of that attractor. The model is assumed to be well behaved in that every point will eventually move within `epsilon` of an attractor. **There is absolutely no guarantee that all basins will be captured by this approach**, even with a fine-grained discretization. It is possible to blur boundaries, crop basins, or miss entire regions.

Usage

```
simbasins(discretize = NULL, xlim = NULL, ylim = NULL, iters = NULL,
  epsilon = NULL, behind = TRUE, tolerance = sqrt(.Machine$double.eps),
  stride = 8, cols = NULL, missingCol = "NA", ...)
```

Arguments

`discretize`	The size of each square. If not set, the discretization of the range is used. May be set separately from the discretization of the range without overwriting.
`xlim`	The range of x values to calculate regions over. Defaults to the limits of the range.
`ylim`	The range of y values to calculate regions over. Defaults to the limits of the range.
`iters`	If not set, each point will be iterated indvidually. If set as a number, exactly that many iterations will be used. If set as `Inf`, will iterate until points stabilize (see `tolerance`). See details.

epsilon	The distance at which a point is considered to have reached an attractor. Defaults to discretize^2. Not used if iters has a numeric value.
behind	Forces this item to be a background object for the purposes of layering
tolerance	The distance distance at which a point is considered to have stopped moving. Defaults to sqrt(.Machine$double.eps).
stride	The number of times the function is applied before movement is checked: in essence finding the basins for f^stride. For non-periodic dynamical systems, this is merely an efficiency concern. For points that move to a periodic attractor with a period that is a factor of stride, this may color the basins by their parity or rank. Only used when iters has a non-numeric value. Defaults to 8.
cols	The colors to use for the various regions. The colors will be used in the order the attractors were added to the model.
missingCol	The color given to points that stop outside of epsilon of an attractor. Defaults to "NA".
...	Extra graphical parameters for image.

Details

All attractors should be dspoints with the attractor flag set to TRUE, and should already be composed with the model. Attractors may have display=FALSE set to avoid displaying the attractor itself. Their color (or region color, if defined) will be used as the color of the region. If there are no points with the attractor flag set, then all points are used as possible attractors. This is not recommended.

If iters is not set, or is set to NULL, then each point will be individually iterated until within epsilon distance of an attractor, or until it moves less than tolerance between iterations. Points that stop moving further than epsilon of an attractor, are colored missingCol, default "NA".

If iters is given a numeric value, each point is iterated exactly iters times, and tolerance will have no effect. If the final image is within epsilon of an attractor, then the square is colored appropriately. If not, then the point is colored missingCol, default "NA". This will take bounded time, but may give a poorer result.

If iters is given an infinite value, the points are iterated until they move less than tolerance distance. An attractor is chosen only if it falls within epsilon distance, otherwise the point is colored missingCol.

The image function is used to display the results.

See Also

dspoint

dsregion

dspolygon

simattractors

Examples

```
library(dsmodels)

model <- dsmodel(function(X0,Y0) {
list(X0*exp(2.6-X0-6.45/(1+4.5*X0)),
     Y0*exp(2.6-Y0-0.15*X0-6.25/(1+4.5*Y0)))
})
model+dsrange(0:3,0:3,discretize = .08)+
      dspoint(1.9358, 1.5059, attractor=TRUE, col="green",  label = "K12")+
      dspoint(1.9358, 0,      attractor=TRUE, col="magenta",label = "K1")+
      dspoint(0,      1.9649, attractor=TRUE, col="orange", label = "K2")+
      dspoint(0,      0,      attractor=TRUE, col="blue",   display=FALSE)+
      dspoint(0.4419, 0.4416, col="yellow", label="A11")+
      simbasins(discretize=0.05)
```

---

xlabel *Create a label on the x-axis*

Description

Labels the x-axis

Usage

```
xlabel(label = "", line = 3, ...)
```

Arguments

label	The title of the axis, to be displayed on image. Text can be input in the form of pseudo-LaTeX code within quotes. See TeX for more details.
line	The distance from the axis the text will be displayed.
...	Extra parameters. These parameters are fed into `mtext()`.

See Also

TeX

Examples

```
 library(dsmodels)
 fun <- function(X,Y) {
 list(
     X/exp(Y),
     Y/exp(X)
  )
}

model <- dsmodel(fun, title = "Cool Function!")
```

```
model + dsrange(-2:3,-2:3, discretize = .09) +
 xlabel("X-Axis shows $\\alpha$!") +
 ylabel("Y-Axis shows $\\beta$!")
```

---

ylabel *Create a label on the y-axis*

Description

Labels the y-axis

Usage

```
ylabel(label = "", line = 3, ...)
```

Arguments

label	The title of the axis, to be displayed on image. Text can be input in the form of pseudo-LaTeX code within quotes. See TeX for more details.
line	The distance from the axis the text will be displayed.
...	Extra parameters. These parameters are fed into mtext().

See Also

TeX

Examples

```
 library(dsmodels)
 fun <- function(X,Y) {
 list(
   X/exp(Y),
   Y/exp(X)
 )
}
model <- dsmodel(fun, title = "Cool Function!")
model + dsrange(-2:3,-2:3, discretize = .09) +
 xlabel("$X$-Axis shows $\\alpha$!") +
 ylabel("$Y$-Axis shows $\\beta$!")
```

Index