Package ‘afCEC’

August 15, 2017

Type Package

Title What the package does (short line)

Version 1.0

Date 2015-02-25

Author Your Name

Maintainer Your Name `<your@email.com>`

Description More about what it does (maybe more than one line)

License GPL (>= 2)

Imports Rcpp (>= 0.11.3), RcppArmadillo, rgl

LinkingTo Rcpp, RcppArmadillo, rgl

Depends Rcpp, RcppArmadillo, rgl

R topics documented:

`afCEC-package` *Active Function Cross-Entropy Clustering*

Description

Active Function Cross-Entropy Clustering (afCEC) partitions the n-dimensional data into the clusters by finding the parameters of the mixed generalized multivariate normal distribution, that optimally approximates the scattering of the data in the n-dimensional space, whose density function is of the form:

p_1*N(mi_1,^sigma_1,sigma_1,f_1)+...+p_k*N(mi_k,^sigma_k,sigma_k,f_k)

The above-mentioned generalization is performed by introducing so called "f-adapted gaussian densities" (i.e. the ordinary gaussian densities adapted by the "active function"). Additionally, the Active Function Cross-Entropy Clustering performs the automatic reduction of the unnecessary clusters. It's implemented in the form of a customizable function `afCEC`.

Details

Package:	afCEC
Type:	Package
Version:	0.9.1
Date:	2014-10-06
License:	GPL-3

Author(s)

Krzysztof Byrski

See Also

`afCEC`.

`afCEC` *afCEC*

Description

Performs the Active Function Cross-Entropy Clustering on the data set.

Usage

```
afCEC (
    points,
    maxClusters,
    initialLabels="k-means++",
    cardMin=0.01,
    costThreshold=-0.000001,
    minIterations=10,
    maxIterations=100,
    numberOfStarts=10,
    method="Hartigan",
    values="quadratic",
    interactive=FALSE
)
```

Arguments

`points`	A (#points x n) matrix of data containing n-dimensional points stored row-by-row.
`maxClusters`	Indicates the maximal number of clusters, that afCEC can partition the data into.
`initialLabels`	Initial labelling determining the data membership to the particular clusters. There are 3 options allowed: • "random" - causes the labelling to be randomly generated. • "k-means++" - causes the labelling to be generated using the k-means++ heuristics. • A (#points x numberOfStarts) matrix of type integer containing values in the range: 0...maxClusters - 1. The value x in the row i and column j indicates that in the j-th start, the i-th point will be initially assigned to the (x + 1)-th cluster. The defalt value is "k-means++".
`cardMin`	Value so that if the number of points in particular cluster relatively to the size of the whole data drops bellow, then that cluster gets removed and it's data is assigned to the other clusters. The default value is 0.01.
`costTreshold`	Negative value so that if the difference between the calculated cost in the subsequent iterations is greater than it then the afCEC terminates returning the solution from the terminated iteration as the final solution (in the present start), provided that at least minIterations already passed. The default value is -0.000001;
`minIterations`	Value indicating the minimal number of iterations needed before afCEC can terminate, provided that no error occurred. The default value is 10.
`maxIterations`	Value indicating the maximal number of iterations in one start, that afCEC cannot exceed. The default value is 100.
`numberOfStarts`	Value indicating the number of runs of the algorithm to be performed. The best solution is chosen out of the solutions given in the particular steps, so that it has the lowest value of the cost function among them. The default value is 10.
`method`	Heuristics used to perform the clustering. There are 2 options allowed: • "Lloyd" - indicates that the Lloyd heuristics will be used to perform the clustering. • "Hartigan" - indicates that the Hartigan heuristics will be used to perform the clustering. The default value is "Hartigan".
`values`	Definition of the active function family used to perform the clustering. There are 3 options allowed:

- "quadratic" - indicates that the function family of the form:

  f(x_1,...,x_(n-1))=a_1*x_1^2+...+a_(n-1)*x_(n-1)^2+a_n*x_1+...+a_(2*n-2)*x_(n-1)+a_(2*n-1)*1,

  where n is the dimensionality of the data and a_i are the optimal coefficients determined by afCEC using the linear least square fitting, will be used to perform the clustering.
- "String containing the user-defined formula. See more in `User-defined formulas`.
- A ((m*n) x #points) matrix containing the values of the particular components of the active function on the data set, placed according to the following layout (row by row):

  First m rows:

  [f_1(x_1_2,...,x_1_n), f_1(x_2_2,...,x_2_n), ..., f_1(x_#points_2,...,x_#points_n)]
  [f_2(x_1_2,...,x_1_n), f_2(x_2_2,...,x_2_n), ..., f_2(x_#points_2,...,x_#points_n)]
  ...
  [f_m(x_1_2,...,x_1_n), f_m(x_2_2,...,x_2_n), ..., f_m(x_#points_2,...,x_#points_n)]

  Second m rows:

  [f_1(x_1_1,x_1_3,...,x_1_n), f_1(x_2_1,x_2_3,...,x_2_n), ...]
  [f_2(x_1_1,x_1_3,...,x_1_n), f_2(x_2_1,x_2_3,...,x_2_n), ...]
  ...
  [f_m(x_1_1,x_1_3,...,x_1_n), f_m(x_2_1,x_2_3,...,x_2_n), ...]

  Last m rows:

  [f_1(x_1_1,...,x_1_(n-1)), f_1(x_2_1,...,x_2_(n-1)), ...]
  [f_2(x_1_1,...,x_1_(n-1)), f_2(x_2_1,...,x_2_(n-1)), ...]
  ...
  [f_m(x_1_1,...,x_1_(n-1)), f_m(x_2_1,...,x_2_(n-1)), ...],

  where: n - dimensionality of the data, x_i_j - j-th coordinate of the i-th point of data, m - number of components of the active function.

  In the foregoing case, the active function family consists of the functions of the form:

  f(x_1,...,x_(n-1))=a_1*f_1(x_1,...,x_(n-1))+...+a_m*f_m(x_1,...,x_(n-1)),

  where n is the dimensionality of the data and a_i are the optimal coefficients determined by afCEC using the linear least square fitting.

The default value is "quadratic".

Remarks:

In the case of the third way of defining the function, the returned object doesn't contain the information about the means coordinates corresponding to the active direction. Moreover, the plotting capabilities of the afCEC package are severely impaired in that case. See `plot`.

`interactive`	Indicates if the algorithm runs in the "interactive" mode. The "interactive" mode allows to track the intermediate steps of the afCEC. Instead of one object of the afCEC class, the whole list of such objects is returned, where each item of the list corresponds to the intermediate step of the algorithm. See `value` section for more details. The default value is FALSE.

Value

- Empty list - if clustering failed.
- Object of class afCEC containing the parameters of the best fitted mixed generalized multivariate normal distribution - if argument interactive=FALSE and clustering succeded.
- List of k objects of class afCEC containing the parameters of the best fitted mixed generalized multivariate normal distribution across the k subsequent steps of the algorithm - if argument interactive=TRUE and clustering succeeded.

See Also

`User-defined formulas`
`plot`

Examples

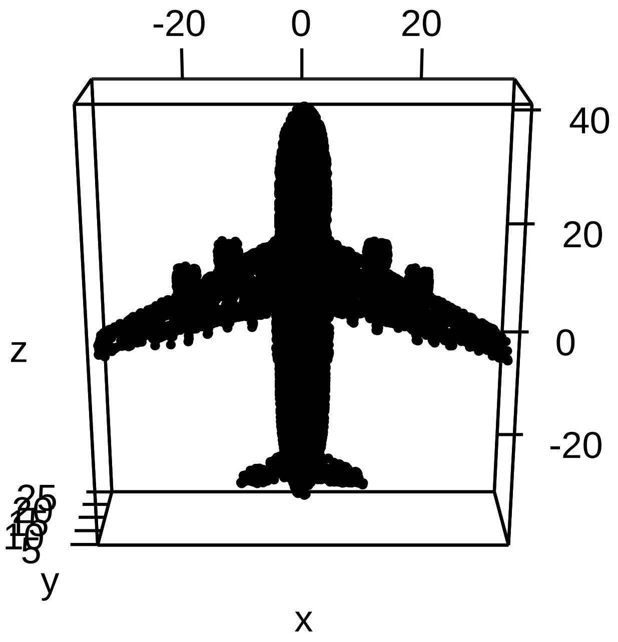
```
# The following three examples demonstrate three equivalent ways of passing the same
# 3D quadratic active function to the afCEC routine:

# 1) Using "quadratic" value (default):

library(afCEC);
data(airplane);
result <- afCEC(airplane, 17);

# what is equivalent to:

library(afCEC);
data(airplane);
result <- afCEC(airplane, 17, values="quadratic");

# 2) Using the string with the user-defined formula:

library(afCEC);
data(airplane);
formula = paste(
    "f:R^2 -> R^5\n",
    "f^1(x) = x(1)^2\n",
    "f^2(x) = x(2)^2\n",
    "f^3(x) = x(1)\n",
    "f^4(x) = x(2)\n",
    "f^5(x) = 1\n",
    sep=""
);
result <- afCEC(airplane, 17, values=formula);

# 3) Using the matrix containing the explicit values of the active function across the
#    subsequent dimensions:
```

```
library(afCEC);
data(airplane);
values = matrix(rep(0, 5*3*dim(airplane)[1]), 5*3, dim(airplane)[1]);
for (i in 1:dim(airplane)[1]) {
    tmp <- airplane[i,2:3];
    for (j in 1:3) {
        values[((j - 1) * 5) + 1, i] <- tmp[1]^2;
        values[((j - 1) * 5) + 2, i] <- tmp[2]^2;
        values[((j - 1) * 5) + 3, i] <- tmp[1];
        values[((j - 1) * 5) + 4, i] <- tmp[2];
        values[((j - 1) * 5) + 5, i] <- 1;
        if (j < 3) tmp[j] <- airplane[i,j];
    }
}
result <- afCEC(airplane, 17, values=values);
```

`airplane` *airplane*

Description

A 10000 x 3 matrix of three dimensional points uniformly distributed on the surface of the airplane model.



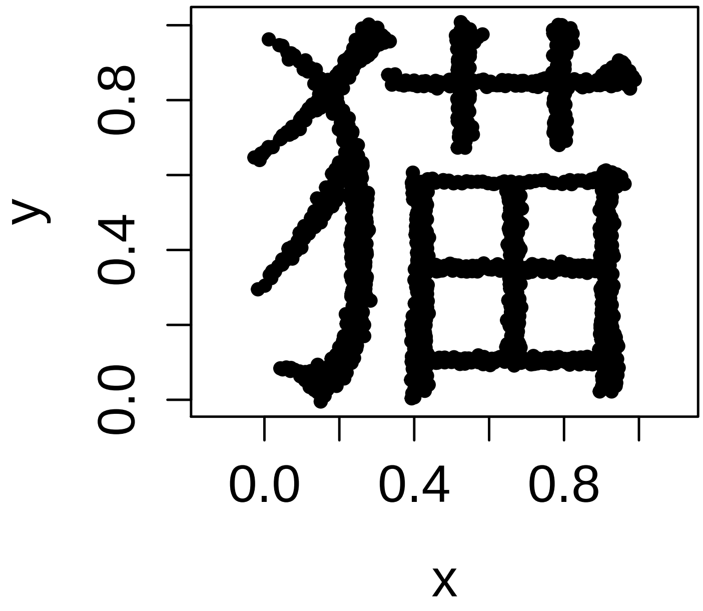
Examples

```
library(afCEC);
data(airplane);
plot3d(airplane);
```

`cat` *cat*

Description

A 1385 x 2 matrix of two dimensional points forming the chinese "cat" letter.



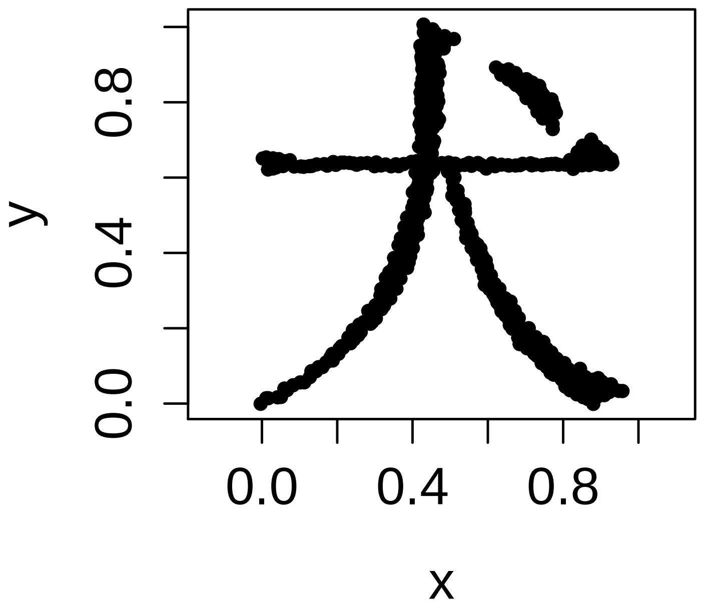
Examples

```
library(afCEC);
data(cat);
plot(cat);
```

`dog` *dog*

Description

A 623 x 2 matrix of two dimensional points forming the chinese "dog" letter.



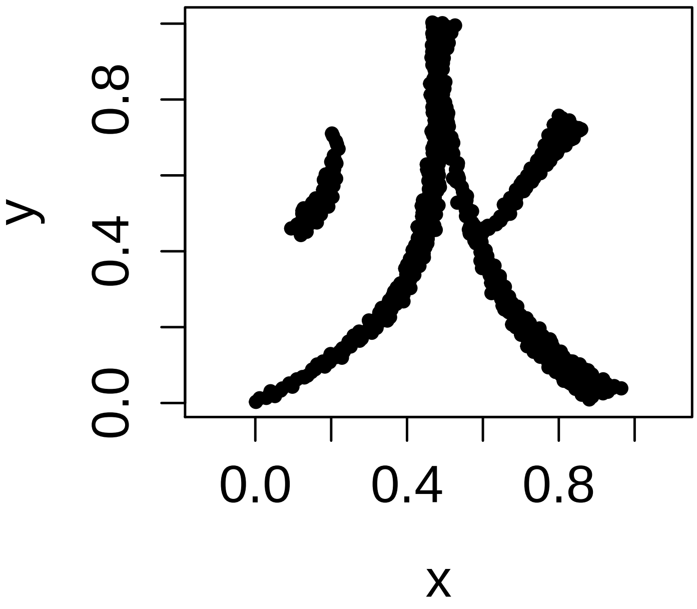
Examples

```
library(afCEC);
data(dog);
plot(dog);
```

`fire` *fire*

Description

A 609 x 2 matrix of two dimensional points forming the chinese "fire" letter.



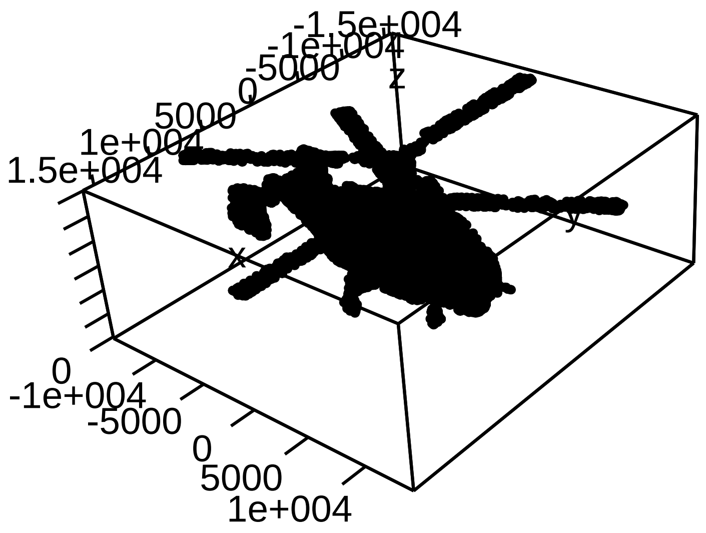
Examples

```
library(afCEC);
data(fire);
plot(fire);
```

`helicopter` *helicopter*

Description

A 10000 x 3 matrix of three dimensional points uniformly distributed on the surface of the helicopter model.



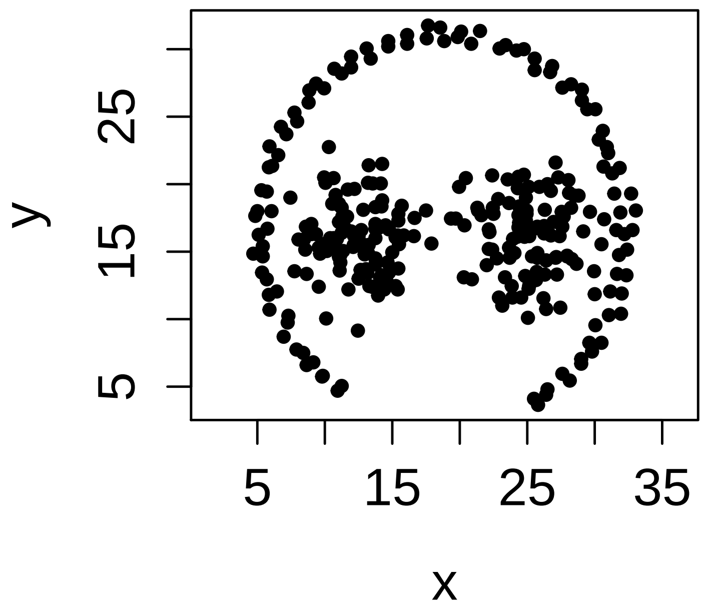
Examples

```
library(afCEC);
data(helicopter);
plot3d(helicopter);
```

`pathbased` *pathbased*

Description

A 300 x 2 matrix of two dimensional points forming the incomplete circle and two filled circles inside.



Examples

```
library(afCEC);
data(pathbased);
plot(pathbased);
```

`plot` *plot*

Description

Plots the chart showing the clustering of the data. Depending on the dimensionality of the data passed to the afCEC function, it draws either 2D or 3D chart. The function doesn't work with the other dimensions.

Remarks:

- The particular items are drawn cluster-by-cluster in the following order:
  - points
  - means
  - curves / surfaces
  - ellipses / ellipsoids of confidence

- The functionality concerning drawing the means (see the `draw_means` argument description for more details), ellipses / ellipsoids of confidence (see the `draw_ellipsoids` argument description for more details) and curves / surfaces (see the `draw_surfaces` argument description for more details) doesn't work if object `x` was retrieved after clustering the data with the active function family defined by the matrix of the active function components values on the data set. In that case, the above-mentioned arguments as well as the other ones related to them (for example `meansSize2D`) are ignored and only data points are drawn on the chart. See afCEC.

Usage

```
plot (
    x,
    draw_points=TRUE,
    draw_means=TRUE,
    draw_ellipsoids=TRUE,
    draw_surfaces=TRUE,
    confidence=0.95,
    grid_resolution=32,

    pointsSize2D=1, pointsColor2D="cluster",
    meansSize2D=2, meansColor2D="black",
    ellipsesHeight2D=1, ellipsesColor2D="black",
    surfacesHeight2D=1, surfacesColor2D="black",
    XLabel2D="X", YLabel2D="Y",

    pointsAlpha3D=1, pointsSize3D=0.01, pointsColor3D="cluster",
    meansAlpha3D=0.5, meansSize3D=0.01, meansColor3D="black",
    ellipsoidsAlpha3D=0.25, ellipsoidsColor3D="cluster",
    surfacesAlpha3D=0.5, surfacesColor3D="cluster",
    XLabel3D="X", YLabel3D="Y", ZLabel3D="Z"
)
```

Arguments

`x`	Object of class afCEC returned from the afCEC function.
`draw_points`	If the value is TRUE, the function draws the points belonging to the particular clusters. The default value is TRUE.
`draw_means`	If the value is TRUE, the function draws the means of the clusters as the big black dots / spheres. The default value is TRUE.
`draw_ellipsoids`	If the value is TRUE, the function draws the curved ellipses / ellipsoids of confidence of the particular clusters. The default value is TRUE.
`draw_surfaces`	If the value is TRUE, the function draws the curves / surfaces corresponding to the subspace spanned by the inactive axes of the particular clusters with respect to the curvilinear coordinate system determined by the active function with 0's in the means of the clusters. The default value is TRUE.
`confidence`	Determines the percentile of data belonging to the particular clusters, the corresponding ellipses / ellipsoids of confidence should contain. For example, if the value 0.5 is specified, then, for each cluster, it's ellipse / ellipsoid of confidence will contain 0.5 * 100 [%] of assigned points. The default value is 0.95.

`grid_resolution`
: Determines the grid resolution using to approximate the curves / surfaces drawn in the plot function. The default value is 32.

`pointsSize2D`
: Determines the size of the points (in pixels) drawn in the plot function. The argument is used only in the case of the 2D data. The default value is 1.

`pointsColor2D`
: Determines the color of the points drawn in the plot function. It can hold the following values:

  - Any color format acceptable by the standard R plot function. In that case the color is fixed across the particular clusters.
  - The "cluster" string indicating, that the plot function will use unique colors for points belonging to the different clusters.

  The argument is used only in the case of the 2D data. The default value is "cluster".

`meansSize2D`
: Determines the size of the means (in pixels) drawn in the plot function. The argument is used only in the case of the 2D data. The default value is 2.

`meansColor2D`
: Determines the color of the means drawn in the plot function. It can hold the following values:

  - Any color format acceptable by the standard R plot function. In that case the color is fixed across the particular clusters.
  - The "cluster" string indicating, that the plot function will use unique colors for means of the different clusters.

  The argument is used only in the case of the 2D data. The default value is "black".

`ellipsesHeight2D`
: Determines the thickness of the ellipses (in pixels) drawn in the plot function. The argument is used only in the case of the 2D data. The default value is 1.

`ellipsesColor2D`
: Determines the color of the ellipses drawn in the plot function. It can hold the following values:

  - Any color format acceptable by the standard R plot function. In that case the color is fixed across the particular clusters.
  - The "cluster" string indicating, that the plot function will use unique colors for ellipses of confidence of the different clusters.

  The argument is used only in the case of the 2D data. The default value is "black".

`surfacesHeight2D`
: Determines the thickness of the curves (in pixels) drawn in the plot function (see the `draw_surfaces` argument description for more details). The argument is used only in the case of the 2D data. The default value is 1.

`surfacesColor2D`
: Determines the color of the curves drawn in the plot function. It can hold the following values:

  - Any color format acceptable by the standard R plot function. In that case the color is fixed across the particular clusters.
  - The "cluster" string indicating, that the plot function will use unique colors for curves (see the `draw_surfaces` argument description for more details) belonging the different clusters.

	The argument is used only in the case of the 2D data. The default value is "black".
`XLabel2D`	Determines the label on the X axis of the chart drawn by the plot function. The argument is used only in the case of the 2D data. The default value is "X".
`YLabel2D`	Determines the label on the Y axis of the chart drawn by the plot function. The argument is used only in the case of the 2D data. The default value is "Y".

Value

None.

See Also

`afCEC`

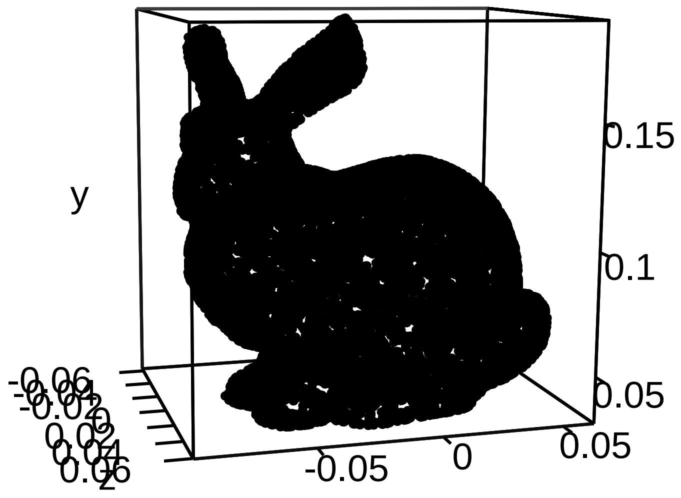
Examples

```
library(afCEC);
data(airplane);
result <- afCEC(airplane, 17);
plot(result);
```

`rabbit` *rabbit*

Description

A 10000 x 3 matrix of three dimensional points uniformly distributed on the surface of the rabbit model.



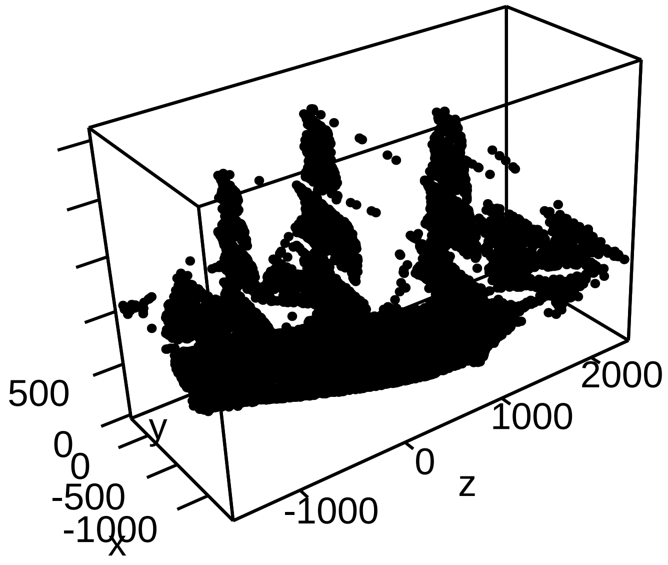
Examples

```
library(afCEC);
data(rabbit);
plot3d(rabbit);
```

`ship` *ship*

Description

A 10000 x 3 matrix of three dimensional points uniformly distributed on the surface of the ship model.



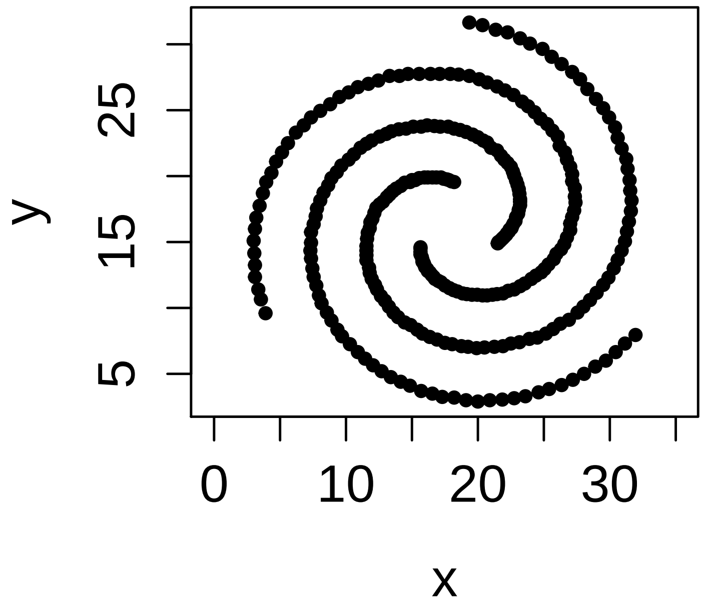
Examples

```
library(afCEC);
data(ship);
plot3d(ship);
```

`spiral` *spiral*

Description

A 312 x 2 matrix of two dimensional points forming the spiral.



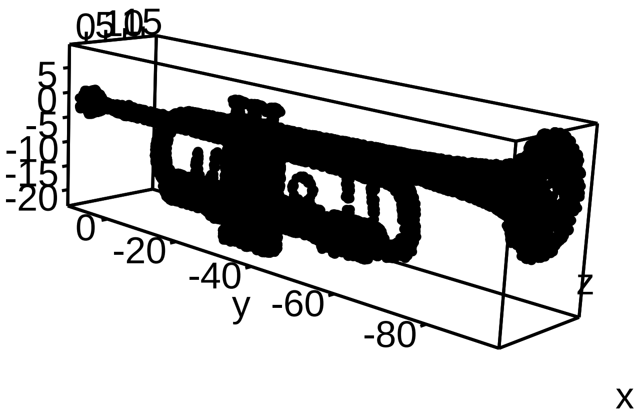
Examples

```
library(afCEC);
data(spiral);
plot(spiral);
```

`trumpet` *trumpet*

Description

A 10000 x 3 matrix of three dimensional points uniformly distributed on the surface of the trumpet model.



Examples

```
library(afCEC);
data(trumpet);
plot3d(trumpet);
```

`UserDefinedFormulas` *User-defined formulas*

Operators:

List of the supported operators (from highest to the lowest priority):

- Arithmetic ! (factorial);
- Arithmetic ^ (power);
- Arithmetic unary +, - ;
  Logical ! (not);
- Arithmetic *, /, % (modulo division);
- Arithmetic +, - ;
- Relational <=, <, >, >= ;
- Relational ==, != ;
- Logical && (and);
- Logical || (or);
- Ternary conditional a ? b : c ;

Remarks:

- ! Computes the factorial of the argument of type integer. If the argument is not an integer value, it's rounded to the nearest one;
- The relational operators produce the expression of the boolean type;
- The logical operators can be used only with expressions of type boolean. Otherwise the exception is thrown and the `afCEC` function returns with errors;
- The boolean expressions can be used as arguments to the arithmetic operators. In that case they are implicitly converted to the 1 value (if the expression evaluates to TRUE) and 0 respectively, otherwise.

  Example:

  The following expressions defined in three different ways are equivalent:

  ((x<=-1)*((x+1)^2+1))+((x>=1)*((x-1)^2+1))+(((x>-1)&&(x<1))*1)

  x<=-1?(x+1)^2+1:x>=1?(x-1)^2+1:1

  if(x<=-1,(x+1)^2+1,if(x>=1,(x-1)^2+1,1))

- The same applies for passing the boolean expressions as the function's real value arguments;

Built-in constants:

- e - Base of the natural logarithm;
- pi - The pi value;

Built-in simple functions:

- abs(x) - Computes the absolute value of x;
- acos(x) - Computes the inverse cosine of x and returns the value expressed in radians;
- acosh(x) - Computes the inverse hyperbolic cosine of x and returns the value expressed in radians;
- acot(x) - Computes the inverse cotangent of x and returns the value expressed in radians;
- acoth(x) - Computes the inverse hyperbolic cotangent of x and returns the value expressed in radians;
- asin(x) - Computes the inverse sine of x and returns the value expressed in radians;
- asinh(x) - Computes the inverse hyperbolic sine of x and returns the value expressed in radians;
- atan(x) - Computes the inverse tangent of x and returns the value expressed in radians;
- atanh(x) - Computes the inverse hyperbolic tangent of x and returns the value expressed in radians;
- binom(r,k) - Computes the r choose k, where r is an arbitrary real number and k is the integer. If k is not an integer value, it's rounded to the nearest one;
- ceil(x) - Computes the ceiling function of x;
- clamp(x,a,b) - Clamps x to the range [a,b];
- cos(x) - Computes the cosine of x expressed in radians;
- cosh(x) - Computes the hyperbolic cosine of x expressed in radians;

- cot(x) - Computes the cotangent of tx expressed in radians;
- coth(x) - Computes the hyperbolic cotangent of x expressed in radians;
- exp(x) - Computes the exponent of x;
- floor(x) - Computes the floor function of x;
- frac(x) - Computes the fractional part of x;
- isfinite(x) - Returns logical TRUE if x is the finite value. Otherwise, the logical FALSE is returned;
- ln(x) - Computes the natural logarithm of x;
- log(x) - Computes the 10-base logarithm of x;
- log(a,x) - Computes the a-base logarithm of x;
- round(x) - Rounds x to the nearest integer value;
- sgn(x) - Computes the signum function of x;

  If x < 0, sgn(x) returns -1;
  If x = 0, sgn(x) returns 0;
  If x > 0, sgn(x) returns 1;

- sin(x) - Computes the sine of x expressed in radians;
- sinh(x) - Computes the hyperbolic sine of x expressed in radians;
- sqrt(x) - Computes the square root of x;
- tan(x) - Computes the tangent of x expressed in radians;
- tanh(x) - Computes the hyperbolic tangent of x expressed in radians;
- trunc(x) - Truncates x (i.e., rounds x towards zero);

Built-in complex functions:

- if(<condition>, <expression_if_true>, <expression_if_false>) - Returns the expression determined by the <expression_if_true> argument, if the <condition> expression evaluates to TRUE. Otherwise, <expression_if_false> is returned. <condition> expression must have the boolean value. Otherwise the exception is thrown and the `afCEC` function returns with errors.

  Example:

  if(x<=-1,(x+1)^2+1,if(x>=1,(x-1)^2+1,1))

- max(x1, x2, ...) - Returns the maximum of the arguments. If no argument is specified (i.e., we have max()), the function returns the neutral element of the max operator (i.e., -INFINITY);
- max(<index_variable>, <low_range>, <high_range>, <expression>) - Returns the maximum value of the expression determined by the <expression> argument depending on the index variable <index_variable> in the range [<low_range>, <high_range>]. <index_variable> is the indentifier of the index variable, while <low_range> and <high_range> are the integer values determining the range boundaries. If <low_range> or <high_range> are not integer values, they are rounded to the nearest ones. In the case where the interval [<low_range>, <high_range>] is degenerate (i.e., <high_range> < <low_range>), the function returns the neutral element of the max operator (i.e., -INFINITY);

  Example:

max(i,0,10,max(j,0,10,sin(i^j)))

- min(x1, x2, ...) - Returns the minimum of the arguments. If no argument is specified (i.e., we have min()), the function returns the neutral element of the min operator (i.e., +INFINITY);
- min(<index_variable>, <low_range>, <high_range>, <expression>) - Returns the minimum value of the expression determined by the <expression> argument depending on the index variable <index_variable> in the range [<low_range>, <high_range>]. <index_variable> is the indentifier of the index variable, while <low_range> and <high_range> are the integer values determining the range boundaries. If <low_range> or <high_range> are not integer values, they are rounded to the nearest ones. In the case where the interval [<low_range>, <high_range>] is degenerate (i.e., <high_range> < <low_range>), the function returns the neutral element of the min operator (i.e., +INFINITY);

  Example:

  min(i,0,10,min(j,0,10,sin(i^j)))

- prod(<index_variable>, <low_range>, <high_range>, <expression>) - Returns the product of the expression determined by the <expression> argument depending on the index variable <index_variable> over the range [<low_range>, <high_range>]. <index_variable> is the indentifier of the index variable, while <low_range> and <high_range> are the integer values determining the range boundaries. If <low_range> or <high_range> are not integer values, they are rounded to the nearest ones. In the case where the interval [<low_range>, <high_range>] is degenerate (i.e., <high_range> < <low_range>), the function returns the neutral element of the * operator (i.e., 1);

  Example:

  prod(i,0,10,prod(j,0,10,sin(i^j)))

- sum(<index_variable>, <low_range>, <high_range>, <expression>) - Returns the sum of the expression determined by the <expression> argument depending on the index variable <index_variable> over the range [<low_range>, <high_range>]. <index_variable> is the indentifier of the index variable, while <low_range> and <high_range> are the integer values determining the range boundaries. If <low_range> or <high_range> are not integer values, they are rounded to the nearest ones. In the case where the interval [<low_range>, <high_range>] is degenerate (i.e., <high_range> < <low_range>), the function returns the neutral element of the + operator (i.e., 0);

  Example:

  sum(i,0,10,sum(j,0,10,sin(i^j)))

Declaring functions:

The function declaration consists of the function signature and the function formula placed after the signature.

Function signature:

The function signature is the expression of the form:

<funcion_name> : <domain_1> x ... x <domain_n> -> <counterdomain>,

where:

<function_name> - Is the identifier of the function;

<domain_i> - Specifies the domain of the i-th argument of the function. It can take the following values:

- R - Indicates that the function argument will be the real value scalar;
- R^<integer_type_constant_expression> - Indicates that the function argument will be the real fixed size vector with number of elements equal to the integer value constant expression. If the expression is not of the type integer, it's rounded to the nearest integer value. The resulting number of elements must be the natural number > 0. Otherwise the exception is thrown and the `afCEC` function returns with errors.
- R^<identifier> - Indicates that the function argument will be the real vector with variable number of arguments accesible (inside the function formula) by the variable <indentifier>;
- Z - Indicates that the function argument will be the integer value scalar;
- Z^<integer_type_constant_expression> - Indicates that the function argument will be the integer fixed size vector with number of elements equal to the integer value constant expression. If the expression is not of the type integer, it's rounded to the nearest integer value. The resulting number of elements must be the natural number > 0. Otherwise the exception is thrown and the `afCEC` function returns with errors.
- Z^<identifier> - Indicates that the function argument will be the integer vector with variable number of arguments accesible (inside the function formula) by the variable <indentifier>;

<counterdomain> - Specifies the counterdomain of the function. It can take the following values:

- R - Indicates that the counterdomain will be the real value scalar;
- R^<integer_type_expression> - Indicates that the counterdomain will be the real vector with the number of elements given by the integer value expression depending on the variables indicating the size of the particular variable size vector arguments. If the expression is not of the type integer, it's rounded to the nearest integer value. The resulting number of elements must be the natural number > 0. Otherwise the exception is thrown and the `afCEC` function returns with errors.

  Example:

  f:R^n -> R^2*n+1

- Z - Indicates that the counterdomain will be the integer value scalar;
- Z^<integer_type_expression> - Indicates that the counterdomain will be the integer vector with the number of elements given by the integer value expression depending on the variables indicating the size of the particular variable size vector arguments. If the expression is not of the type integer, it's rounded to the nearest integer value. The resulting number of elements must be the natural number > 0. Otherwise the exception is thrown and the `afCEC` function returns with errors.

  Example:

  f:R^n -> Z^2*n+1

Remarks:

- The arguments passed to the function are strictly checked with the function signature for the size constraints, i.e., if the vector with more than one element is passed for the scalar argument (or vice-versa) or the vector with different number of elements than those declared in the signature in the case of the fixed size vector is passed as the fixed size vector argument, the exception is thrown and the `afCEC` function returns with errors;
- As it was implicitly stated above, one can pass the scalar argument if the fixed size vector with one element or the variable size vector is required. The same applies to passing the vectors containing exactly one element as the scalar arguments;
- Conversions between the R and Z domain are allowed and performed automatically. In the case of the R to Z conversion, the rounding to the nearest integer value is used;
- In the case of the variable size vectors, the same identifier may be used for more than one argument. In that case, the constraint on the size of that vectors is imposed in the sense that if there exist two or more vectors in the group with the same identifier that have different number of elements, then the exception is thrown and the `afCEC` function returns with errors;

  Example:

  f:R^n x R^m x R^n x R^m -> R

  In that case, the first and third and the second and fourth vector must have exactly the same number of elements. Otherwise, the exception is thrown and the `afCEC` function returns with errors;
- In the case of the fixed size vector arguments declarations using the integer type constant expressions, the globally visible constants passed to the `afCEC` function via the `params` argument are also available to use in the expression. Similarly, the globally visible constants are also available in the integer type expressions in the declaration of the counterdomain;

  Example:

  Suppose, that the globally visible constant d indicating the degree of the multivariate polynomial was passed to the function `afCEC`. The declaration of that polynomial may look like as follows:

  f:R^n -> R^n^d

Index