# Package 'ACCLMA'

February 19, 2015

**Type** Package

**Title** ACC & LMA Graph Plotting

**Version** 1.0

**Date** 2010-05-31

**Author** Tal Carmi, Liat Gaziel

**Maintainer** Tal Carmi <talca@bgu.ac.il>

**Description** The main function is plotLMA(sourcefile,header) that takes a data set and plots the appropriate LMA and ACC graphs. If no sourcefile (a string) was passed, a manual data entry window is opened. The header parameter indicates by TRUE/FALSE (false by default) if the source CSV file has a head row or not. The data set should contain only one independent variable (X) and one dependent varialbe (Y) and can contain a weight for each observation

**License** GPL-2

**LazyLoad** yes

**Keywords** ACC,LMA,Absolute Concentration Curve, LOI minus ACC, Regression coefficient robustness

**Repository** CRAN

**Date/Publication** 2012-10-29 13:13:35

**NeedsCompilation** no

## R topics documented:

---

`ACCLMA-package` *Plots ACCvsLOI & LMA graphs*

---

## Description

This package contains a function that imports data from a CSV file or uses manually entered data from the format (X,Y,Weight) and plots the appropriate ACC vs LOI graph and LMA graph.

## Details

| | |
|---|---|
| Package: | ACCLMA |
| Type: | Package |
| Version: | 1.0 |
| Date: | 2010-05-31 |
| License: | GPL-2 |
| LazyLoad: | yes |

The main function is plotLMA(filename,header). If an import from a CSV file is required, enter the full file path a string to the filename parameter and specify using the boolean header parameter if the file contains a header row or not. Leaving this field blank opens a window for manual data entry. The data set is expcted in the format of (X,Y,Weight), if weights are irrelvant, leave the column blank (null values).

## Author(s)

Tal Carmi, Liat Gaziel

Maintainer: Tal Carmi <talca@bgu.ac.il>

## References

~~ Literature or other references for background information ~~

## Examples

```
#PlotlMA("c:/data.csv",TRUE)
```

averageSameXs *Unifies all same X observations*

## Description

The function unifies all the observations with the same X values into one obseravtion with regard to the weights of each observation

## Usage

```
averageSameXs(mat)
```

## Arguments

| | |
|---|---|
| mat | A matrix containing the X,Y,Weight columns after the calcWeights function been used on it |

## Value

Data set with only row for each X value

## Author(s)

Tal Carmi, Liat Gaziel

## Examples

```
d <- c(1,1,3,4)
e <- c(5,6,7,8)
f <- c(1,1,1,1)
mydata <- data.frame(d,e,f)
names(mydata) <- c("X","Y","Weight")
averageSameXs(mydata)

## The function is currently defined as
function (mat)
{
    n <- size(mat)
    temp <- matrix(NA, nrow = n, ncol = 3)
    line <- 1
    i <- 1
    while (i <= n) {
        j <- i
        sumY <- 0
        sumWeights <- 0
        while (j <= n & mat[i, 1] == mat[j, 1]) {
            sumY <- sumY + mat[j, 2] * mat[j, 3]
            sumWeights <- sumWeights + mat[j, 3]
            j <- j + 1
```

```
        }
        temp[line, 1] <- mat[i, 1]
        temp[line, 2] <- sumY/sumWeights
        temp[line, 3] <- sumWeights
        line <- line + 1
        i <- j
    }
    temp <- as.data.frame(temp[1:(line - 1), ])
    names(temp)[1] = "X"
    names(temp)[2] = "Y"
    names(temp)[3] = "Weights"
    return(temp)
  }
```

---

`calcFX` *Calculates the appropriate F(X) value of each observation*

---

## Description

Calculate the F(X) of each obseravtion by summing the F(X) of the previous observation with the weight of the current observation, should be used only after the calcWeights has been used on the source matrix

## Usage

```
calcFX(mat)
```

## Arguments

| | |
|---|---|
| `mat` | A data set that has the X,Y,Weight columns after calcWeights and reduceSameXs has been used on it |

## Value

Return a data set with the F(X) column calculated

## Author(s)

Tal Carmi,Liat Gaziel

## Examples

```
d <- c(1,1,3,4)
e <- c(5,6,7,8)
f <- c(1,1,1,1)
mydata <- data.frame(d,e,f)
names(mydata) <- c("X","Y","Weight")
mydata<-calcWeights(mydata)
mydata<-averageSameXs(mydata)
```

```
calcFX(mydata)

## The function is currently defined as
function (mat)
{
    mat[1, 4] = mat[1, 3]
    n <- size(mat)
    for (i in 2:n) {
        mat[i, 4] = mat[i - 1, 4] + mat[i, 3]
    }
    names(mat)[4] <- "FX"
    return(mat)
  }
```

---

calcFY *Calculates the appropriate FY value of each observation*

---

## Description

Calculates the aggregated mean Y value of each observation by summing the F(Y) value of the previous observation with the Y*Weight of the current observation

## Usage

```
calcFY(mat)
```

## Arguments

| | |
|---|---|
| mat | A matrix containing the X,Y,Weight,FX columns after the function calcWeights,reduceSameXs and calcFX has been used on it |

## Value

A matrix containg the newely added FY column

## Author(s)

Tal Carmi, Liat Gaziel

## Examples

```
d <- c(1,1,3,4)
e <- c(5,6,7,8)
f <- c(1,1,1,1)
mydata <- data.frame(d,e,f)
names(mydata) <- c("X","Y","Weight")
mydata<-calcWeights(mydata)
mydata<-averageSameXs(mydata)
mydata<-calcFX(mydata)
calcFY(mydata)
```

```
## The function is currently defined as
function (mat)
{
    mat[1, 5] = mat[1, 2] * mat[1, 3]
    n <- size(mat)
    for (i in 2:n) {
        mat[i, 5] = mat[i - 1, 5] + mat[i, 2] * mat[i, 3]
    }
    names(mat)[5] <- "FY"
    return(mat)
  }
```

---

calcLMA *Calculates the appropriate LMA values for each observation*

---

## Description

This function receives a matrix containing the X,Y,Weight,FX,FY,LOI columns, after the calcWeights and the reduceSameXs function has been used on it and calculates the LMA column

## Usage

```
calcLMA(mat)
```

## Arguments

| | |
|---|---|
| mat | A matrix containing the X,Y,Weight,FX,FY,LOI columns, after the calcWeights and the reduceSameXs function has been used on it |

## Value

A matrix containing the newely added LMA column

## Author(s)

Tal Carmi, Liat Gaziel

## Examples

```
d <- c(1,1,3,4)
e <- c(5,6,7,8)
f <- c(1,1,1,1)
mydata <- data.frame(d,e,f)
names(mydata) <- c("X","Y","Weight")
mydata<-calcWeights(mydata)
mydata<-averageSameXs(mydata)
mydata<-calcFX(mydata)
mydata<-calcFY(mydata)
mydata<-calcLOI(mydata)
```

```
calcLMA(mydata)

## The function is currently defined as
function (mat)
{
    n <- size(mat)
    for (i in 1:n) {
        mat[i, 7] = mat[i, 6] - mat[i, 5]
    }
    names(mat)[7] = "LMA"
    return(mat)
  }
```

---

`calcLOI` *Calculate the appropriate LOI value for each observation*

---

## Description

This function receives a matrix containing the X,Y,Weight,FX,FY columns, after the calcWeights and the reduceSameXs function has been used on it and calculates the LOI column

## Usage

```
calcLOI(mat)
```

## Arguments

| | |
|---|---|
| `mat` | A matrix containing the X,Y,Weight,FX,FY columns, after the calcWeights and the reduceSameXs function has been used on it |

## Value

A matrix with the newely added LOI column

## Author(s)

Tal Carmi, Liat Gaziel

## Examples

```
d <- c(1,1,3,4)
e <- c(5,6,7,8)
f <- c(1,1,1,1)
mydata <- data.frame(d,e,f)
names(mydata) <- c("X","Y","Weight")
mydata<-calcWeights(mydata)
mydata<-averageSameXs(mydata)
mydata<-calcFX(mydata)
mydata<-calcFY(mydata)
calcLOI(mydata)
```

```
## The function is currently defined as
function (mat)
{
    n <- size(mat)
    for (i in 1:n) {
        mat[i, 6] = mat[i, 4] * mat[n, 5]
    }
    names(mat)[6] = "LOI"
    return(mat)
  }
```

---

`calcWeights` *Calculates the relative weight of each observation*

---

## Description

This function receives a matrix after the averageSameXs function has been used on it and calculates the relative weight of each observation by dividing its weight by the total weights. If at least one observation doesn't have a weight specified the function assumes all the observations are of equal weight

## Usage

```
calcWeights(mat)
```

## Arguments

| | |
|---|---|
| `mat` | a matrix after the averageSameXs function has been used on it |

## Value

A matrix with the updated weights column

## Author(s)

Tal Carmi, Liat Gaziel

## Examples

```
d <- c(1,1,3,4)
e <- c(5,6,7,8)
f <- c(1,1,1,1)
mydata <- data.frame(d,e,f)
names(mydata) <- c("X","Y","Weight")
mydata<-averageSameXs(mydata)
calcWeights(mydata)

## The function is currently defined as
```

```
function (mat)
{
    n <- size(mat)
    zeroWeight <- 0
    sumWeight <- 0
    for (i in 1:n) {
        if (is.null(mat[i, 3]) || is.na(mat[i, 3]) || mat[i,
            3] == 0) {
            zeroWeight <- 1
        }
        sumWeight <- sumWeight + mat[i, 3]
    }
    if (zeroWeight == 1) {
        mat[3] = 1/n
        return(mat)
    }
    for (i in 1:n) {
        mat[i, 3] <- mat[i, 3]/sumWeight
    }
    return(mat)
  }
```

---

`fillCSVData` *Imports data from a CSV file and sort it by the first column*

---

## Description

This function open a CSV, copies its first three columns to a data set and then sort it by the first column, the function assumes the first column is the X values, the second one the Y values and the third one to be the weights column and names these columns accordingly

## Usage

```
fillCSVData(str, head)
```

## Arguments

| | |
|---|---|
| `str` | A string containing the full path of the CSV file to be imported, should be left blank if manual data entry is desired |
| `head` | A boolean variable indicating if the imported CSV file has a header row or not, ignored if str is null |

## Value

A data set sorted by the first column

## Author(s)

Tal Carmi, Liat Gaziel

## Examples

```
#Assuming there is a three column csv in C drive named "data.csv" with no header
#mydata<-fillCSVData("c:/data.csv",FALSE)

## The function is currently defined as
function (str, head)
{
    mat <- read.table(str, header = head, sep = ",")
    names(mat)[1] = "X"
    names(mat)[2] = "Y"
    if ((is.null(mat[1, 3]) && is.null(mat[2, 3])) || (mat[1,
        3] == 0 && mat[2, 3] == 0))
        mat <- calcWeights(mat)
    names(mat)[3] = "Weights"
    mat <- mat[order(mat[1]), ]
    return(mat)
  }
```

---

`fillData` *Opens a window for manual entry of X,Y,weight data set*

---

## Description

This function opens a manual data entry window, saves it as a data set and then sorts it by the X values (the first column)

## Usage

```
fillData()
```

## Value

A sorted by X data set

## Author(s)

Tal Carmi, Liat Gaziel

## Examples

```
#mydata<-fillData()

## The function is currently defined as
function ()
{
    mydata <- data.frame(X = numeric(0), Y = numeric(0), Weight = numeric(0))
    mydata <- edit(mydata)
    mydata <- mydata[order(mydata[1]), ]
    return(mydata)
  }
```

plotGraphs *Plots the ACC vs LOI graph and the LMA graph*

## Description

This function receives a matrix containing the X,Y,Weight,FX,FY,LOI,LMA columns after the calcWeights and the reduceSameXs function has been used on it and plots the appropriate ACC vs LOI graph and the LMA graph, each in a seperate window

## Usage

```
plotGraphs(mat)
```

## Arguments

| | |
|---|---|
| mat | A matrix containing the X,Y,Weight,FX,FY,LOI,LMA columns after the calcWeights and the reduceSameXs function has been used on it |

## Value

none

## Author(s)

Tal Carmi, Liat Gaziel

## Examples

```
d <- c(1,1,3,4)
e <- c(5,6,7,8)
f <- c(1,1,1,1)
mydata <- data.frame(d,e,f)
names(mydata) <- c("X","Y","Weight")
mydata<-calcWeights(mydata)
mydata<-averageSameXs(mydata)
mydata<-calcFX(mydata)
mydata<-calcFY(mydata)
mydata<-calcLOI(mydata)
mydata<-calcLMA(mydata)
plotGraphs(mydata)

## The function is currently defined as
function (mat)
{
    mat[size(mat) + 1, 4] = 0
    mat[size(mat) + 1, 5] = 0
    mat[size(mat) + 1, 6] = 0
    mat[size(mat) + 1, 7] = 0
    mat <- mat[order(mat[4]), ]
```

```
  trans <- t(mat)
  originalPar <- par(no.readonly = TRUE)
  par(lwd = 2)
  par(col = "black")
  plot(trans[4, ], trans[5, ], type = "n", main = "ACC", xlab = "F(x)",
      ylab = "Accumulating Y Mean")
  par(col = "blue")
  lines(trans[4, ], trans[5, ], type = "l")
  par(col = "red")
  lines(trans[4, ], trans[6, ], type = "l")
  legend("bottomright", c("ACC", "LOE"), col = c("blue", "red"),
      lwd = 2, bty = "n", text.col = "black")
  par(originalPar)
  windows()
  par(lwd = 2)
  par(col = "black")
  par(xaxs = "i")
  plot(trans[4, ], trans[7, ], type = "n", main = "LMA", xlab = "F(x)",
      ylab = "LOE minus ACC")
  par(col = "black", lwd = 1)
  lines(c(0, 1), c(0, 0), type = "l")
  par(col = "blue", lwd = 2)
  lines(trans[4, ], trans[7, ], type = "l")
  par(originalPar)
}
```

---

`plotLMA` *Plots the ACC vs LOI and LMA graphs of an imported or manually entered data set*

---

## Description

This is the main function of the package and the one that should be used to plot the ACC vs LOI and LMA graphs. If weights are not relevant leave the column blank (null values)

## Usage

```
plotLMA(str = NULL, header = FALSE)
```

## Arguments

| | |
|---|---|
| `str` | A string containing the full path of the CSV file to be imported, should be left blank if manual data entry is desired |
| `header` | A boolean variable indicating if the imported CSV file has a header row or not, ignored if str is null |

## Value

A matrix containing all the calculated columns that were used in the graphs plotting

### Author(s)

Tal Carmi,Liat Gaziel

### Examples

```
#plotLMA()

## The function is currently defined as
function (str = NULL, header = FALSE)
{
    if (is.null(str)) {
        mat <- fillData()
    }
    else {
        mat <- fillCSVData(str, header)
    }
    mat <- calcWeights(mat)
    mat <- averageSameXs(mat)
    mat <- calcFX(mat)
    mat <- calcFY(mat)
    mat <- calcLOI(mat)
    mat <- calcLMA(mat)
    plotGraphs(mat)
    return(mat)
  }
```

---

## size *Counts the number of rows in a data set*

---

### Description

This function counts the number of rows in a data set by running over the first column and stops on the first blank value in this column it finds

### Usage

```
size(mat)
```

### Arguments

| | |
|---|---|
| mat | Any data set |

### Value

An integer representing the number of rows in the data set

### Author(s)

Tal Carmi, Liat Gaziel

## Examples

```
d <- c(1,1,3,4)
e <- c(5,6,7,8)
f <- c(1,1,1,1)
mydata <- data.frame(d,e,f)
names(mydata) <- c("X","Y","Weight")
size(mydata)

## The function is currently defined as
function (mat)
{
    i <- 1
    while (!is.na(mat[i, 1]) & !is.null(mat[i, 1])) {
        i = i + 1
    }
    return(i - 1)
  }
```

# Index