

Package ‘mseq’

February 14, 2012

Type Package

Title Modeling non-uniformity in short-read rates in RNA-Seq data

Version 1.2

Date 2011-09-7

Depends R(>= 2.5),gbm

Author Jun Li

Maintainer Jun Li <junli07@stanford.edu>

Description This package implements all the methods in the paper “Modeling non-uniformity in short-read rates in RNA-Seq data”. Especially, it implements both the iterative glm procedure for the Poisson linear model and the training procedure of the MART model. The cross-validation for both of the methods is also implemented. Version 1.1 and later also implements the Poisson linear model with dinucleotide composition. Version 1.2 corrected a bug in `expData.R`. This bug leads to mis-labeled lines in `plotCoef.R`, but will not change other parts like training, testing, or cross-validation.

License GPL (>= 2)

LazyData yes

Repository CRAN

Date/Publication 2011-09-08 08:34:02

R topics documented:

mseq-package	2
CVTrainIndexGene	4
expData	5
expData2nt	6
gl_part	7

getDev	8
getNullCount	9
getOriLogPref	9
getPredCount	10
getWeight	11
glmPred	11
glmPred2nt	12
iterGlm	13
iterGlm2nt	13
iterGlmCV	14
iterGlmCV2nt	15
martCV	15
martPred	16
martTrain	17
plotCoef	18
plotCounts	18
setOriOffset	19
updateOffset	20

Index	21
--------------	-----------

mseq-package

Modeling non-uniformity in short-read rates in RNA-Seq data

Description

This package implements all the methods in the paper "Modeling non-uniformity in short-read rates in RNA-Seq data". Especially, it implements both the iterative glm procedure for the Poisson linear model and the training procedure of the MART model. The cross-validation for both of the methods is also implemented. Moreover, in folder `data_top100`, data files of the top 100 genes and surrounding sequences of all the eight sub-datasets are provided. In Version 1.2, these datasets have been moved to <http://www.stanford.edu/~junli07/research.html> to make this R package smaller (mseq 1.1 was among the top 50 largest on CRAN!). In Version 1.1 and later, the Poisson linear model with dinucleotide composition is also implemented.

Details

```

Package:  mseq
Type:    Package
Version:  1.2
Date:    2011-09-7
License:  GPL (version 2 or newer)
LazyData: yes

```

This package mainly includes the following functions:
`expData.R`: Expand the surrounding sequences in the original file.

`plotCounts.R`: Plot the counts in the original data or fitted counts.
`getDev.R`: Calculate the deviance under Poisson model.
`getNullCount.R`: Get the counts of reads for each position under the null hypothesis.
`getPredCount.R`: Get the predicted counts given the predicted preferences.
`iterGlm.R`: Fit the Poisson linear model by iteratively fitting glm and gene expression levels.
`plotCoef.R`: Plot the coefficients of Poisson linear model.
`glmPred.R`: Predict the log preferences using the trained iterative glm model.
`iterGlmCV.R`: Get the cross-validation R squared for iterative glm.
`martTrain.R`: Train the MART model.
`martPred.R`: Get the predicted log sequencing preferences by the trained MART model.
`martCV.R`: Get the cross-validation R squared for MART.
`expData2nt.R`: Expand the surrounding sequences in the original file, dinucleotide composition included.
`iterGlm2nt.R`: Fit the Poisson linear model with dinucleotide composition by iteratively fitting glm and gene expression levels.
`glmPred2nt.R`: Predict the log preferences using the trained iterative glm model with dinucleotide composition.
`iterGlmCV2nt.R`: Get the cross-validation R squared for iterative glm with dinucleotide composition.

Author(s)

Jun Li

Maintainer: Jun Li <junli07@stanford.edu>

References

Li J, Jiang H, Wong WH, Modeling non-uniformity in short-read rates in RNA-Seq data, submitted.

Examples

```

# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3) # here the surrounding sequences is only of length 5. In real datasets, it should be 1
pdf("ori_counts.pdf")
plotCounts(data$count[data$index == 1])
dev.off()

# get the CV R squared for Poisson linear model
R_sq <- iterGlmCV(data)

# train and predict by Poisson linear model
train.data <- data[data$index < 6, ]
test.data <- data[data$index >= 6, ]
train.glm <- iterGlm(train.data)
pdf("coef.pdf")
plotCoef(train.glm, 2, 3)
dev.off()
pred.pref <- exp(glmPred(train.glm, test.data))

```

```

# get predicted counts
pred.count <- getPredCount(test.data, pred.pref)
pdf("glm_fitted_counts.pdf")
plotCounts(pred.count[data$index == 1])
dev.off()

# get the R squared
glm.dev <- getDev(pred.count, test.data$count)
null.count <- getNullCount(test.data)
null.dev <- getDev(null.count, test.data$count)
R_sq <- 1 - glm.dev / null.dev

# To shorten the running time, this example uses small values of interaction.depth and n.trees. For real datasets,
# get the CV R squared for MART
R_sq <- martCV(data, interaction.depth = 2, n.trees = 100)

# train and predict by MART
train.data <- data[data$index < 6, ]
test.data <- data[data$index >= 6, ]
train.mart <- martTrain(train.data, interaction.depth = 2, n.trees = 100)
pred.pref <- exp(martPred(train.mart, test.data, n.trees = 100))

# get predicted counts
pred.count <- getPredCount(test.data, pred.pref)
pdf("mart_fitted_counts.pdf")
plotCounts(pred.count[data$index == 1])
dev.off()

# get the R squared
mart.dev <- getDev(pred.count, test.data$count)
null.count <- getNullCount(test.data)
null.dev <- getDev(null.count, test.data$count)
R_sq <- 1 - mart.dev / null.dev

#### Poisson linear model using dinucleotide composition
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData2nt(g1_part, 2, 1) # here the surrounding sequences is only of length 3. In real datasets, it should

# get the CV R squared for Poisson linear model
R_sq <- iterGlmCV2nt(data)

# train and predict by Poisson linear model
train.data <- data[data$index < 6, ]
test.data <- data[data$index >= 6, ]
train.glm <- iterGlm2nt(train.data)
pred.pref <- exp(glmPred2nt(train.glm, test.data))

```

Description

Get index matrix for the training data in cross-validation. Only for internal use. Not available to users.

Usage

```
CVTrainIndexGene(data, fold, seed)
```

Arguments

data	the data frame generated by expData.R
fold	the fold of cross-validation
seed	random seed

Value

A matrix of TRUE or FALSE. TRUE for training data, and FALSE for testing data. Each row strands for one fold.

expData	<i>expand the surrounding sequences</i>
---------	---

Description

The original data file only contains one sequence. This function will expand for each retained position its surrounding sequence. The output data frame can be directly used by Poisson linear model and MART.

Usage

```
expData(oriData, llen, rlen)
```

Arguments

oriData	the original data frame read directly from the file
llen	the number of nucleotides before the first nucleotide of a read, which we consider as surrounding sequence
rlen	the number of nucleotides in and after the first nucleotide of a read, which we consider as surrounding sequence

Details

Note that we will not check the format of oriData here. Please generate the original data file carefully. Please refer to Readme_format.txt for the details of the format.

Value

a data frame including the counts and surrounding sequences, which can be directly used by Poisson linear model and MART. The number of rows are the number of 0s in `oriData$tag`. The number of columns are `llen + rlen + 2`, with names `index`, `count`, `pMllen`, ..., `pM1`, `p0`, `p1`, ..., `p(rlen-1)`, where `p` means position, `M` means minus.

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3) # In real datasets, the surrounding sequences should be set longer.
```

expData2nt	<i>expand the surrounding sequences, including both the single nucleotide and dinucleotide composition</i>
------------	--

Description

The original data file only contains one sequence. This function will expand for each retained position its surrounding sequence, including both the single nucleotide and dinucleotide composition. The output data frame can be directly used by Poisson linear model and MART.

Usage

```
expData2nt(oriData, llen, rlen)
```

Arguments

<code>oriData</code>	the original data frame read directly from the file
<code>llen</code>	the number of nucleotides before the first nucleotide of a read, which we consider as surrounding sequence
<code>rlen</code>	the number of nucleotides in and after the first nucleotide of a read, which we consider as surrounding sequence

Details

Note that we will not check the format of `oriData` here. Please generate the original data file carefully. Please refer to `Readme_format.txt` for the details of the format.

Value

a data frame including the counts and surrounding sequences, which can be directly used by Poisson linear model and MART. The number of rows are the number of 0s in `oriData$tag`. The number of columns are $2 + (llen + rlen) * 3 + (llen + rlen - 1) * 9$, with names `index`, `count`, `pMllen.A`, `pMllen.C`, `pMllen.G`, ..., `(prlen-1).A`, `(prlen-1).C`, `(prlen-1).G`, `pMllen.pM(llen-1).TA`, ..., `p(rlen-2).p(rlen-1).CC`, where `p` means position, `M` means minus.

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData2nt(g1_part, 2, 1) # In real datasets, the surrounding sequences should be set longer.
```

g1_part	<i>a part of g1.csv</i>
---------	-------------------------

Description

g1.csv in folder data_top100 (In Version 1.2, data_top100 has been moved to <http://www.stanford.edu/~junli07/research.html> to reduce the total size of this package.) stores the counts and sequences of the top 100 genes in the Grimmond EB data. This file only stores a part of g1.csv—the top 10 genes. The reason we only keep a small part is to shorten the calculation time of the example codes. The full top 100 genes from each datasets are provides as separate files in the folder data_top100. Please read Readme_format.txt for details about the data and the required format. This data can be generated by

```
g1 <- read.csv("g1.csv"); g1_part <- g1[g1$index < 11,]
```

Usage

```
data(g1_part)
```

Format

A data frame with 8307 observations on the following 4 variables.

index a numeric vector

tag a numeric vector

seq a factor with levels T A C G

count a numeric vector

Details

index is an index for the gene from where this count comes.

tag is an integer value, 0 means to consider this count, any other value means this count should not be taken into account. In our files, -2 means the UTR part, and -1 means the further 100 bp. The user can use any integer other than 0 to denote the discarded counts.

seq is the nucleotide of this position. Must be capital A C G T. No other characters accepted. No little characters accepted. No missing values accepted. If the number of missing values is small, you can use T (or A G T) for them; this should not change the result significantly.

count is the count of reads starting at this position.

For each gene (or each group of positions that have the same level of expression, like exon or isoform), a distinguished index should be used. Each gene (or group) may include positions in both strand (like data generated by Illumina) or single strand (like data generated by ABi). Within each

gene (or group), the positions should be in the 5 prime to 3 prime order for each strand. There should be no gaps or missing values. So actually, for each gene in Illumina outputs, the data are comprised of two halves. The first half are the data from the forward strand, and the second half are the data from the second strand. For each gene in ABi outputs, there are no such two halves.

For each gene or each half, the nucleotides retained for analysis should be surrounded with long-enough nonretained nucleotides. For example, if you want to consider left 40 bp and right 40 bp as surrounding sequences, then there should be at least 40 bp in both sides of nucleotides retained.

Right formats are very important; otherwise, the program may give unpredictable results. This package itself will not justify the correctness of the format. Please make sure you have done it.

References

Li J, Jiang H, Wong WH, Modeling non-uniformity in short-read rates in RNA-Seq data, submitted.

getDev	<i>Poisson deviance</i>
--------	-------------------------

Description

Calculate the deviance under Poisson model.

Usage

```
getDev(pred_count, real_count)
```

Arguments

pred_count	the fitted or predicted counts
real_count	the real counts in the data

Value

the deviance, a single numeric value

References

Hardin JW, Hilbe JM: Generalized Linear Models and Extensions. 2 edn. College Station, TX: Stata Press; 2007.

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3)

null.count <- getNullCount(data)
null.dev <- getDev(null.count, data$count)
```

getNullCount	<i>counts under null model</i>
--------------	--------------------------------

Description

Get the counts of reads for each position under the null hypothesis (all reads have the same sequencing preferences).

Usage

```
getNullCount(data)
```

Arguments

data the data frame generated by expData.R

Value

a numeric vector stores the counts

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3)

null.count <- getNullCount(data)
null.dev <- getDev(null.count, data$count)
```

getOriLogPref	<i>initialize for mart</i>
---------------	----------------------------

Description

Get the original log preferences, that is, the logarithms of gene expression levels. Here the gene with zero count will be replaced by a small count. Only for internal use. Not available to users.

Usage

```
getOriLogPref(data, small_count = 0.5)
```

Arguments

data the data frame generated by expData.R
small_count the small count that will replace the zero count

Value

a numeric vector of the log preferences

getPredCount	<i>predicted counts</i>
--------------	-------------------------

Description

Get the predicted counts given the predicted preferences.

Usage

```
getPredCount(data, pred_pref)
```

Arguments

data	the data frame generated by expData.R
pred_pref	the predicted preferences

Value

predicted counts, a numeric vector

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3) # In real datasets, surrounding sequences should be set longer.

# train and predict by Poisson linear model
train.data <- data[data$index < 6, ]
test.data <- data[data$index >= 6, ]
train.glm <- iterGlm(train.data)
pred.pref <- exp(glmPred(train.glm, test.data))

pred.count <- getPredCount(test.data, pred.pref)
```

getWeight	<i>weight vector</i>
-----------	----------------------

Description

Get the weight vector for MART. Only for internal use. Not available to users.

Usage

```
getWeight(data)
```

Arguments

data	the data frame generated by expData.R
------	---------------------------------------

Value

a numeric vector of the weights

glmPred	<i>predict using iterative glm</i>
---------	------------------------------------

Description

Predict the log preferences using the trained iterative glm model.

Usage

```
glmPred(train.glm, newdata)
```

Arguments

train.glm	the trained iterative glm model
newdata	the new data

Value

the predicted log preferences. a numeric vector.

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3) # In real datasets, surrounding sequences should be set longer.

# train and predict by Poisson linear model
train.data <- data[data$index < 6, ]
test.data <- data[data$index >= 6, ]
train.glm <- iterGlm(train.data)
pred.pref <- exp(glmPred(train.glm, test.data))
```

glmPred2nt

predict using iterative glm with dinucleotide composition

Description

Predict the log preferences using the trained iterative glm model with dinucleotide composition.

Usage

```
glmPred2nt(train.glm, newdata)
```

Arguments

train.glm	the trained iterative glm model
newdata	the new data

Value

the predicted log preferences. a numeric vector.

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData2nt(g1_part, 2, 1) # In real datasets, surrounding sequences should be set longer.

# train and predict by Poisson linear model
train.data <- data[data$index < 6, ]
test.data <- data[data$index >= 6, ]
train.glm <- iterGlm2nt(train.data)
pred.pref <- exp(glmPred2nt(train.glm, test.data))
```

iterGlm	<i>iterative generalized linear regression</i>
---------	--

Description

Fit the Poisson linear model by iteratively fitting glm and gene expression levels.

Usage

```
iterGlm(data, thrd = 0.01, max_iter = 10)
```

Arguments

data	the data frame generated by expData.R
thrd	the threshold of improvement of R squared
max_iter	maximum number of iterations

Value

an glm structure. Summarize it will list the coefficients and their p-values.

Examples

```
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3)

data.glm <- iterGlm(data)
summary(data.glm)
```

iterGlm2nt	<i>iterative generalized linear regression using dinucleotide composition</i>
------------	---

Description

Fit the Poisson linear model by iteratively fitting glm and gene expression levels.

Usage

```
iterGlm2nt(data, thrd = 0.01, max_iter = 10)
```

Arguments

data	the data frame generated by expData2nt.R
thrd	the threshold of improvement of R squared
max_iter	maximum number of iterations

Value

an glm structure. Summarize it will list the coefficients and their p-values.

Examples

```
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData2nt(g1_part, 2, 1)

data.glm <- iterGlm2nt(data)
summary(data.glm)
```

iterGlmCV	<i>cross-validation for iterative glm</i>
-----------	---

Description

Get the cross-validation R squared for iterative glm.

Usage

```
iterGlmCV(data, fold = 5, seed = 281142, thrd = 0.01, max_iter = 10)
```

Arguments

data	the data frame generated by expData.R
fold	number of folds of CV
seed	seed to generate training indexes
thrd	threshold for R squared increase in iterative glm
max_iter	maximum times of iterations in iterative glm

Value

the CV R squared, a numeric value

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3) # In real datasets, surrounding sequences should be set longer.

# get the CV R squared for Poisson linear model
R_sq <- iterGlmCV(data)
```

iterGlmCV2nt	<i>cross-validation for iterative glm using dinucleotide composition</i>
--------------	--

Description

Get the cross-validation R squared for iterative glm using dinucleotide composition.

Usage

```
iterGlmCV2nt(data, fold = 5, seed = 281142, thrd = 0.01, max_iter = 10)
```

Arguments

data	the data frame generated by expData2nt.R
fold	number of folds of CV
seed	seed to generate training indexes
thrd	threshold for R squared increase in iterative glm
max_iter	maximum times of iterations in iterative glm

Value

the CV R squared, a numeric value

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData2nt(g1_part, 2, 1) # In real datasets, surrounding sequences should be set longer.

# get the CV R squared for Poisson linear model
R_sq <- iterGlmCV2nt(data)
```

martCV	<i>cross-validation for MART</i>
--------	----------------------------------

Description

Get the cross-validation R squared for MART.

Usage

```
martCV(data, fold = 5, seed = 281142, shrinkage = 0.06, interaction.depth = 10, n.trees = 2000, small_cou
```

Arguments

data the data frame generated by expData.R
 fold number of fold for CV
 seed seed to generate training indexes
 shrinkage the shrinkage argument of gbm
 interaction.depth the interaction.depth argument of gbm
 n.trees the n.trees argument of gbm
 small_count the small count which will replace the zero count

Value

the CV R squared, a numeric value

Examples

```

# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3) # In real datasets, surrounding sequences should be set longer.

# To shorten the running time, this example uses small values of interaction.depth and n.trees. For real datasets,
# get the CV R squared for MART
R_sq <- martCV(data, interaction.depth = 2, n.trees = 100)

```

 martPred

predict using MART

Description

Get the predicted log sequencing preferences by the trained MART model, and adjust them so that read TT...T have value 0.

Usage

```
martPred(train.gbm, newdata, n.trees = 2000)
```

Arguments

train.gbm the trained gbm object
 newdata the data that we want to predict
 n.trees number of trees to be considered

Value

a numeric vector of predicted log preferences

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3) # here the surrounding sequences is only of length 5. In real datasets, it should be 1

# train and predict by MART
train.data <- data[data$index < 6, ]
test.data <- data[data$index >= 6, ]
train.mart <- martTrain(train.data, interaction.depth = 2, n.trees = 100)
pred.pref <- exp(martPred(train.mart, test.data, n.trees = 100))
```

martTrain

train using MART

Description

Train the MART model.

Usage

```
martTrain(data, shrinkage = 0.06, interaction.depth = 10, n.trees = 2000, small_count = 0.5)
```

Arguments

data the data frame generated by expData.R
shrinkage the shrinkage argument of gbm
interaction.depth the interaction.depth argument of gbm
n.trees the n.trees argument of gbm
small_count the small count which will replace the zero count

Value

a gbm object

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3)

data.mart <- martTrain(data, interaction.depth = 2, n.trees = 100)
```

plotCoef	<i>plot coefficients of Poisson linear model</i>
----------	--

Description

Plot the coefficients of Poisson linear model.

Usage

```
plotCoef(data.glm, llen, rlen)
```

Arguments

data.glm	the glm model trained by iterGlm
llen	the number of nucleotides before the first nucleotide of a read, which we consider as surrounding sequence
rlen	the number of nucleotides in and after the first nucleotide of a read, which we consider as surrounding sequence

Value

No returned value. Will generate a plot.

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3)

data.glm <- iterGlm(data)
pdf("plot_coef.pdf")
plotCoef(data.glm, 2, 3)
dev.off()
```

plotCounts	<i>plot counts</i>
------------	--------------------

Description

This function creates a bar-plot like image. So it can be used to plot counts or sequencing preferences.

Usage

```
plotCounts(counts)
```

Arguments

counts a sequence of positive values

Value

No returned value. Will generate a plot.

Examples

```
# read and expand the data
data(g1_part) # for real data, please use read.csv, like g1 <- read.csv("g1.csv")
data <- expData(g1_part, 2, 3)

pdf("plot_counts.pdf")
plotCounts(data$count[data$index == 1])
dev.off()
```

setOriOffset

initialize

Description

Get the original offsets for Poisson linear model, that is, the logarithms of gene expression levels. Only for internal use. Not available to users.

Usage

```
setOriOffset(data)
```

Arguments

data the data frame generated by expData.R

Value

a numeric vector of the offsets

updateOffset	<i>update offsets</i>
--------------	-----------------------

Description

Update the offsets for Poisson linear model, that is, the logarithms of gene expression levels when we take sequencing preferences into account. Only for internal use. Not available to users.

Usage

```
updateOffset(data, pred_count, log_expr)
```

Arguments

data	the data frame generated by expData.R
pred_count	the counts fitted by Poisson linear model
log_expr	The log expression levels (offsets) we had before this iteration.

Value

a numeric vector of the offsets

Index

*Topic **datasets**

g1_part, 7

*Topic **models**

CVTrainIndexGene, 5

expData, 5

expData2nt, 6

getDev, 8

getNullCount, 9

getOriLogPref, 9

getPredCount, 10

getWeight, 11

glmPred, 11

glmPred2nt, 12

iterGlm, 13

iterGlm2nt, 13

iterGlmCV, 14

iterGlmCV2nt, 15

martCV, 15

martPred, 16

martTrain, 17

mseq-package, 2

plotCoef, 18

plotCounts, 18

setOriOffset, 19

updateOffset, 20

*Topic **package**

mseq-package, 2

*Topic **regression**

glmPred, 11

glmPred2nt, 12

iterGlm, 13

iterGlm2nt, 13

iterGlmCV, 14

iterGlmCV2nt, 15

mseq-package, 2

*Topic **tree**

martCV, 15

martPred, 16

martTrain, 17

mseq-package, 2

CVTrainIndexGene, 4

expData, 5

expData2nt, 6

g1_part, 7

getDev, 8

getNullCount, 9

getOriLogPref, 9

getPredCount, 10

getWeight, 11

glmPred, 11

glmPred2nt, 12

iterGlm, 13

iterGlm2nt, 13

iterGlmCV, 14

iterGlmCV2nt, 15

martCV, 15

martPred, 16

martTrain, 17

mseq (mseq-package), 2

mseq-package, 2

plotCoef, 18

plotCounts, 18

setOriOffset, 19

updateOffset, 20