DGEobj: An S3 Object to Capture DGE Workflows

John Thompson (john.thompson@bms.com)

2018-03-21

Contents

1 Introduction **1**

2 DGEobj Structure **2**
- 2.1 Base Types . . . 2
- 2.2 DGEobj Nomenclature: Items, Types and Base Types . . . 2
- 2.3 Unique Items . . . 2
- 2.4 Levels . . . 3
- 2.5 Parentage . . . 3
- 2.6 Original Data . . . 3

3 Working with DGE objects **3**
- 3.1 Create a DGEobj . . . 3
  - 3.1.1 Initializing a DGEobj from dataframes . . . 3
  - 3.1.2 Converting to/from RSE or ES data formats . . . 4
- 3.2 Adding Items to a DGEobj . . . 5
- 3.3 Batch addition of multiple items . . . 5
- 3.4 DGEobj Length and Dimensions . . . 5
- 3.5 Subsetting a DEobj . . . 6
- 3.6 Inventory of a DGEobj . . . 6
- 3.7 Adding a new data type . . . 6
- 3.8 Accessing data in a DGEobj . . . 7
  - 3.8.1 Retrieve a single item . . . 7
  - 3.8.2 Retrive multiple items . . . 7
- 3.9 Ancillary Functions . . . 7
  - 3.9.1 Function showTypes . . . 7
  - 3.9.2 Function baseType . . . 7
  - 3.9.3 Function getAttribute and getAttributes . . . 7
  - 3.9.4 Function getItemAttributes . . . 8
  - 3.9.5 Function rmItem . . . 8
  - 3.9.6 Function setAttributes . . . 8
  - 3.9.7 Function showMeta . . . 8
  - 3.9.8 Functions under development . . . 8

4 Questions, bug reports, feature requests **9**

1 Introduction

The DGEobj package implements an S3 class data object, the DGEobj, that represents an extension of the capabilties of the RangedSummarizedExperiment(RSE) originally developed by Martin Morgan et al. Both the DGEobj and the RSE object capture the raw data for a differential gene expression analysis, namely a counts matrix along with associated gene and sample annotation. Additionally, the DGEobj extends this concept to support capture of downstream data objects the capture the workflow of and analysis project. The availability of a structured data object like the DGEobj, that catpure an entire workflow has multiple advantages. Sharing DGE results with other colleagues is simplified because the entire analysis is encapsulated within the DGEobj and the recipient of data in this format can examine details of the analysis based on the annotation built into the DGE object. Further the DGEobj structure enables programatic inspection of analysis results and thus enables automation of higher level integrative analyses across multiple projects.

The RSE object that inspired the DGEobj can capture as many assays as desired. An “assay” is defined as any matrix with n genes (rows) and m samples (columns). A limitation of the RSE object however, is that the RSE can capture only 1 instance of row data (typically gene annotation) and 1 instance of column data (sample annotation with one row for every column of data in the assay slot). This limits the RSE in terms of it’s ability to hold downstream data objects because many of those objects meet the definition of row data also (e.g. DGEList, Fit object, topTable output). Other types of data (e.g. design matrices, sample QC) meet the definition of column data.
Thus, the DGEobj was modeled after the RSE object, but extended to accommodate multiple row and column data types. The DGEobj is thus uniquely suited to capturing the entire workflow of a DGE analysis.

2 DGEobj Structure

2.1 Base Types

The DGEobj supports four distinct data types, that we refer to as “base types”:

- assay data: dataframes or matrices of data with n rows (genes or transcripts) and m columns (samples)
- rowData: a dataframe with n rows typically containing information about each gene with as many columns as needed (gene ID, gene symbols, chromosome information, etc). Other types of rowData include the edgeR DGEList object, design matricies and Fit objects
- colData: a dataframe with m rows, that is one row for each sample column in the assay slot
- metaData: anything that doesn’t fit in one of the other slots

Fundamentally, the base type defines how an item should be subsetted.

2.2 DGEobj Nomenclature: Items, Types and Base Types

We accommodate multiple instances of a base type by defining data types and items. Each data type is assigned a base type (e.g. geneData, granges and fit are all “types” of “basetype”" = rowData).

You can have multiple instances of each type (with some exceptions). We therefore describe each instance of a “type” as an “item” and each item must have a user-defined item name and the item name must be unique within a DGEobj.

2.3 Unique Items

The intent of the DGEobj is that it captures the workflow and analysis results of a single dataset. As such, certain items that constitute the raw data are defined as unique and only one instance of these datatypes are allowed.

Three items, counts, design (expt factors and sample information) and geneData (or isoformData, or exonData) are defined as unique. If appropriate chromosome location data (Chr, Start, End, Strand) are supplied in the geneData, then a granges item is also created upon initializing a DGEobj.

If RNA-Seq data is coming from Xpress, then it will also have an effective length matrix. This goes into an assay slot and is also unique.

2.4 Levels

Three levels are predefined for DGEobj objects: gene, isoform or exon. A DGEobj may contain only one of these levels. Thus you would create a separate DGEobj for isoform data if you were performing a transcript analysis. For the sake of simplicity, throughout this document I will refer to geneData which will exist in a gene level DGEobj. You can mentally substitute isoform or exon for gene if you are working with that type of data.

2.5 Parentage

An analysis can become multi-threaded. For example, multiple models can be built from one dataset. Two features of the DGEobj serve to manage multi-threaded analyses. Only one instance of these data types are allowed. Unique types are the "counts", "design" (sample data) and "geneData". The DGEobj can contain multiple instances of other data types. To document the workflow and unambiguosly define the relationships between data items in a multi-threaded analysis, the concept of parent is invoked. Each data item carries a parent attribute that holds the name of the parent data item. In this way, for example, a topTable item can be linked to the contrast fit that produced it.

2.6 Original Data

In the course of an analysis you may subset the DGEobj to remove undetected genes or remove outlier samples. However, for some purposes you may need to return to the orignal un-subsetted data. For this reason, a copy of the original data is also stored in a metadata slot. Metadata slots are carried along without subsetting so the original data may always be retrieved from the metadata slots. The item names of the original data in the metadata slot have a "_orig" suffix.

3 Working with DGE objects

You can create a DGEobj from a set of dataframes containing the counts, geneData and design information (Sample annotation). If you have an Expression Set (ES) or RSE object, you can also recast it as a DGEobj. Note that when recasting an ES or RSE object only the counts, geneData and design data are transferred.

3.1 Create a DGEobj

3.1.1 Initializing a DGEobj from dataframes

Three dataframes are required to initialize a new DGEobj. Import your counts, geneData and design data as dataframes. Then use the initDGEobj function to create a DGEobj. You must also specify the "level" of your data (one of "gene", "isoform" or "exon").

The counts can be a matrix or dataframe with geneIDs as rownames.
The rowData should be a dataframe also wit geneIDs as rownames.
The colData should be a dataframe with colnames(counts) as the rownames.

Adding custom attributes (customAttr argument) are, strictly speaking, optional but **highly recommended**. The reusability of your DGEobj data is only as good as the annotation. Attributes are supplied as named list of name/value pairs with the names entirely up to the end user. It is incumbent upon the user to define and use a consistent set of names for the annotation they supply.

```
# initialize a DGEobj
myDgeObj <- initDGEobj(counts = MyCounts,
                        rowData = MyGeneAnnotation,
                        colData = MyDesign,
                        level = "gene",
                        customAttr = list (PID = "20171025-0001",
                            XpressID = "12345",
                            Genome = "Mouse.B38",
                            GeneModel = "Ensembl.R89")
                      )
```

It can be tedious to add annotation attrributes via the initDGEobj function, so, to encourage extensive annotation, a more convenient method is supplied. We use Omicsoft ArrayStudio to process RNA-Seq data and we collect annotation on the sample set during the process of loading data into Omicsoft. The annotateDGEobj function will read the Omicsoft registration file and capture the same set of sample set annotation that was collected for the Omicsoft project.

```
myDgeObj <- annotateDGEobj(myDgeObj, regfile,
        keys = list("ID", "BMS_PID", "Title",
                    "Description", "Keywords", "Business Unit",
                    "Functional Area", "Disease",
                    "Vendor", "PlatformType", "Technology",
                    "LibraryPrep", "Organism", "Tissue",
                    "AlignmentReference", "GeneModel", "TBio_Owner",
                    "TA_Owner", "LoadData, "ReadLength", "ReadType",
                    "Pipeline", "AlignmentAlgorithm", "ScriptID")
             )
```

This function attaches the key/value pairs found in regfile and returns the DGEobj with the annotation attached.

The keys argument is optional unless you want to change the default list of keys but included here to show the defaults. You can also turn the keys argument off by assigning a NULL value (keys=NULL) in which case it will add any key value pair it finds.

Operationally, this function reads a set of key/value pairs from a text file and attaches them to the DGEobj. Although designed to capture annotation from an Omicsoft registration file, the format is simple enough that you can easily create such an annotation file. Each key/value pair should be on a line by itself with the key and value separated by an equals sign. Lines without key value pairs will be ignored (it looks for an = sign).

3.1.2 Converting to/from RSE or ES data formats

You may receive data in ES or RSE format or need to reformat data into one of those formats for use with other functions. We therefore provide a function to facilitate the convertion to/from RSE format. The SummarizedExperiment package already provides a method to convert from ES to RSE.

```
#convert ES to RSE
library(SummarizedExperiment)
MyRSE <- as(MyES, "RangedSummarizedExperiment")

#convert RSE to DGEobj
MyDgeObj <- convertRSE(MyRSE, "DGEobj")

#convert RSE to either RSE or ES
MyRSE <- convertDGEobj(MyDgeObj, "RSE")
MyES <- convertDGEobj(MyDgeObj, "ES")
```

3.2 Adding Items to a DGEobj

As you progress through an analysis, you'll want to capture certain data objects in the DGEobj.

To add one item to a DGEobj:

```
myDgeObj <- addItem(myDgeObj, item = MyDGEList,
                              itemName = "MyDGEList",
                              itemType = "DGEList",
                              parent = "counts",
                              funArgs = match.call())
```

The item is the actual data object to add.

The itemName is the user-defined name for that object.

The itemType is the predefined type. To see a list of predefine types use the function **showTypes(MyDgeObj)**.

The parent argument is particularly important for a multi-threaded analysis. If you try more than one fit or apply more than one normalization technique, the parent argument maintains the thread and unambiguously defines the workflow. The value assigned to parent should be the item name of the parent data object.

Passing match.call() to funArgs captures the function arguments from the currently running function and serves to document both the function call and the arguments used to create the data object that is being captured. You can also pass a user-defined text string to the funArgs argument.

There is also an overwrite argument to the addItem function. By default addItem will refuse to add an itemname that already exists. Add the overwrite=TRUE argument if you really mean to update an object that already exists in the DGEobj.

3.3 Batch addition of multiple items

You can also add multiple data objects to a DGEobj in one function call using the addItems function. A typical use case for addItems is you've performed a several different contrasts from a single fit and have multiple topTable dataframes to be added to the DGEobj.

```
MyDgeObj <- addItems(MyDgeObj,
                         itemList = MyItems,
                         itemTypes = MyTypes,
                         parents = MyParents,
                         itemAttr = MyAttributes)
```

itemList is a list of the data objects to add.

itemTypes is a list of the types for each item in itemList.

parents is a list of the parent item names for each item in itemList.

itemAttr is an optional named list of attributes that will be added to every item on the itemList. Note that the DGEobj has attributes and each item within the DGEobj can have its own attributes. Here the attributes are being added to the individual items.

3.4 DGEobj Length and Dimensions

The length of a DGEobj refers to the number of data items in the DGEobj.

```
length(MyDgeObj)
```

The dimension reported for a DGEobj is the dimensions of the assays contained in the DGEobj. That is, the row dimension is the number of genes contained in the object and the column dimension is the number of samples contained in the object.

```
dim(MyDgeObj)
```

3.5 Subsetting a DEobj

Coordinates contained in square brackets can be used to subset a DGEobj, the same way that a dataframe or matrix can be subsetted. The subsetting function uses the base type to define how each data items is handled during subsetting. MetaData items are carried along unchanged.

You subset a DGEobj with square brackets the same way you would subset a matrix.

```
#subset to the first 100 genes
MyDGEobj <- MyDGEobj[1:100,]

#subset to the first 10 samples
MyDGEobj <- MyDGEobj[,1:10]

#susbest genes and samples
MyDGEobj <- MyDGEobj[1:100, 1:10]
```

You may also use boolean vectors to subset dimensions of a DGEobj.

3.6 Inventory of a DGEobj

The inventory function prints a table of the data items present in a DGEobj.
The output includes the item name, type, basettype, parent, class and date created.
if the verbose=TRUE argument is added, a funArgs column is also included.

```
inventory(DGEobj, verbose=TRUE)
```

If you just need the item names of data stored in the DGEobj uses the names function.

```
names(MyDgeObj)
```

3.7 Adding a new data type

A set of data types has already been defined based on data types I encounter in my typical edgeR/Voom workflow. Certainly, I haven't thought of everything that someone might want to capture. Therefore, new types can be defined on the fly.

```
#see predefined data types
showTypes(MyDgeObj)

#add a new colData datatype called "sampleQC"
MyDgeObj <- newType(MyDgeObj,
                    itemType="sampleQC",
                    baseType="col",
                    uniqueItem=FALSE)
```

3.8 Accessing data in a DGEobj

You'll likely need to access various components of the DGEobj as you progress through you workflow. Therea are several ways to extract one or more components from the DGEobj for use in your analysis.

3.8.1 Retrieve a single item

The getItem function allows you to retrieve any item in a DGEobj by referencing it's name. Use the inventory function (described above) to check the contents of the DGEobj.

```
MyCounts <- getItem(MyDgeObj, "counts")
```

3.8.2 Retrive multiple items

There are several ways to retrieve multiple items as a list or data objects.

You can supply a list of item names to retreive specific items. The items requested are returned in a list.

```
MyItems <- getItems(MyDgeObj, list("counts", "geneData"))
```

If all the items you wish to retrieve are of the same type, you can use the getType function to retreive them as a list. For example, you might want to retrieve a set of contrast results present as topTable output.

```
MyContrasts <- getType(MyDgeObj, "topTable")
```

Similarly, you can retrieve all items of the same basetype as a list.

```
MyRowData <- getBaseType(MyDgeObj, "row")
```

Finally, should you need to retrieve all the items in a DGEobj as a simple list, you can recast the DGEobj as a simple list.

```
MyList <- as.list(MyDgeObj)
```

3.9 Ancillary Functions

3.9.1 Function showTypes

Shows all the pre-defined types in a DGEobj. This shows all types whether or not the DGEobj contains data associated with a given type.

```
showTypes(MyDgeObj)
```

3.9.2 Function baseType

Returns the baseType for a given Type

```
baseType(MyDgeObj, "DGEList")
```

3.9.3 Function getAttribute and getAttributes

These are for retrieving attributes from an individual item. So far, the only place where I've used attributes attached to an item (as opposed to the DGEobj itself) is the design matrix. In my workflow I attach the formula as an attribute of the design matrix.

```
getAttribute(MyDgeObj$designMatrix, "formula")
```

If an item had multiple attributes, you could retrieve the list of attributes with getAttributes. getAttributes by default excluded common attributes including dim and dimnames which already have base functions for accessing.

3.9.4 Function getItemAttributes

This function will retrieve a requested attribute from all items in a dgeobj, returned as a list. So far I haven't been assigning any common attributes to all items.

```
getItemAttribute(MyDgeObj, "attributeName")
```

3.9.5 Function rmItem

For the most part, you'll be collecting and adding new object to a DGEobj in the course of an analysis. The rmItem function is added for the case where you might need to delete a particular item.

To delete an item from a DGEobj:

```
MyDgeObj <- rmItem(MyDgeObj, "itemName")
```

3.9.6 Function setAttributes

You can add attributes to an item with the addItem function. This function provides an alternate mechanism to add attributes to an item which would typically be done before adding the item to a DGEobj.

```
MyItem <- setAttributes(MyItem, list(date=date(), analyst="JRT"))
```

3.9.7 Function showMeta

This function is intended to return any project oriented attributes assigned to the DGEobj using function annotateDGEobj. It returns the attributes as a two column dataframe of Attribute/Value pairs.

```
MyAnnotation <- showMeta(MyDgeObj)
```

3.9.8 Functions under development

The main DGEobj functions described above are relatively stable at this point. Other functions in the package are either utility functions used by other functions and not really intended for end user use or functions under active development.

One group of functions in this latter category (active development) is a set of functions designed to query a collection of DGEobj objects and then extract all contrast data and assemble the contrasts into a contrast database. These are under active development, may change substantially with future versions and will likely be migrated to a separate package. These include:

- indexDGEobj
- indexType
- buildContrastDB

4 Questions, bug reports, feature requests

Contact John Thompson: john.thompson@bms.com