An introduction to ascend - Analysis of Single Cell Expression, Normalisation and Differential expression

Anne Senabouth

2017-11-15

Before you begin

System Requirements

Datasets produced by Single Cell RNAseq (scRNAseq) experiments are very large, ranging from a few hundred to a million cells. The number of cells affect the amount of computational resources required to process the dataset – therefore, you need to determine if you have enough computational power and time to complete the analysis. ascend can comfortably analyse datasets of up to 10,000 cells on a single machine with 8GB of RAM and a quad-core CPU. Larger datasets should be run on a High Performance Cluster (HPC).

We have tested this package on datasets ranging from 100 to 20,000 cells. Generally, increasing the number of CPUs will decrease the processing time of functions, while larger datasets require more RAM.

Configuring BiocParallel

This package makes extensive use of BiocParallel, enabling ascend to make the most of your computer's hardware. As each system is different, BiocParallel needs to be configured by the user. Here are some example configurations.

Unix/Linux/MacOS (Single Machine)

```
library(BiocParallel)
ncores <- parallel::detectCores() - 1
register(MulticoreParam(workers = ncores, progressbar=TRUE), default = TRUE)
```

Windows (Single Machine - Quad-core system)

```
library(BiocParallel)
workers <- 3 # Number of cores on your machine - 1
register(SnowParam(workers = workers, type = "SOCK", progressbar = TRUE), default = TRUE)
```

Installation

Required packages

ascend requires the following packages:

CRAN

- devtools
- Matrix
- ggplot2
- data.table

- dplyr
- reshape2
- Rtsne
- limSolve
- ggbeeswarm
- dynamicTreeCut
- dendextend
- RColorBrewer

Bioconductor

- Biobase
- BiocParallel
- scater
- scran
- DESeq

Please install these packages before installing ascend. You do not need to load these packages when you work with ascend unless you are using functions directly from these packages (eg. BiocParallel).

Installing and loading ascend

Please use `devtools` to load the development version of `ascend`.

```
devtools::load_all("~/CodeRepositories/ascend")
```

```
## Loading ascend
```

You can also use devtools' `install_github` function to install the package, and then load it as normal.

```
devtools::install_github("IMB-Computational-Genomics-Lab/ascend")
library(ascend)
```

Preparing Data for ascend

This package has been tested with scRNAseq data generated by Chromium and DropSeq/FlowSeq. Data generated by other systems can be used with this package, provided the input is in the form of an expression matrix. The following section will describe how the data should be formatted.

Expression Matrix

The main source of input is an expression matrix, or a gene-barcode matrix containing transcript counts. They are usually produced at the end of scRNAseq processing pipelines such as Cell Ranger and DropSeq.

In an expression matrix, each row represents a gene and each column represents a cell. The names of rows and columns will subsequently be named accordingly.

`ascend` is able to use any row and column names in the expression matrix, provided they abide by the following criteria:

1. Names should not repeat. If you have a list with repeats, you can make the names unique by using R's 'make.unique' function.
2. You should be able to identify which genes you would like to select as controls. This is why gene symbols or ENSEMBL transcript IDs should be used.
3. Cells from different batches, samples or sequencing runs should be given a numeric identifier at the end. eg. BARCODE-1, BARCODE-2, BARCODE-3.

Combining expression matrices from different batches

You can concatenate multiple expression matrices with the function *JoinMatrices*. Expression matrices generated with this method should then be normalised with the *NormaliseBatches* function. For pipelines such as Chromium's Cell Ranger, the native aggregation function should be used as it takes into account additional information such as molecule information.

Cell Information

Cell Information is a data frame containing cell identifiers, their associated batch identifier and additional information. ascend will automatically generate batch information for an expression matrix if none are provided. However, it will make the assumption that there is only one batch of cells in the expression matrix.

The Cell Information data frame should be structured as follows:

cell_barcode	batch
Cell1-1	1
Cell2-1	1
...	...
...	...
Cellx-x	x

Column 1 should hold the cell identifiers, and column 2 should hold batch information.

Gene Information

The Gene Information slot holds a data frame that contains the gene identifiers used in the expression matrix, in addition to their corresponding identifiers in other systems. ascend will also automatically generate batch information based on the expression matrix if none are provided. The Gene Information data frame can also hold additional information about genes.

The Gene Information data frame should be structured as follows:

gene_identifier1	gene_identifier2
GENE1	IDENTIFIER1
GENE2	IDENTIFIER2
...	...
...	...
GENEX	IDENTIFIERX

Controls

You must provide a list of gene identifiers linked to controls in order to use ascend's filtering functions. These are generally mitochondrial and ribosomal genes. Spike-ins are also used as controls if they were included in the study.

Controls should be organised into a named list, and identifiers used should be present in the expression matrix.

ascend Expression and Metadata Set - EMSet

Structure

An `ascend` Expression and Metadata Set (EMSet) is a S4 object that stores information generated and used by the `ascend` package. This object contains 8 slots which are as follows:

- **ExpressionMatrix**: Transcript counts stored as a sparse matrix, where rows are transcript/gene identifiers and columns are invididual cells.
- **GeneInformation**: A data frame containing information a set of gene identifiers, such as gene symbols or ENSEMBL transcript identifiers. This data frame also holds information on controls and any information provided by the user.
- **CellInformation**: A data frame containing each cell identifier, its associated batch/sample and additional information such as conditions.
- **Controls**: A named list featuring gene identifiers to use as controls. These gene identifiers must match the identifiers used in the expression matrix.
- **PCA**
  - PCA: PCA matrix generated by the `RunPCA` function.
  - PCAPercentVariance: Percent variance of each component. Used by the plot function `PlotPCAVariance`.
- **Clusters**
  - DistanceMatrix: Distance matrix generated from a PCA-reduced or normal expression matrix. Populated by `RunCORE`.
  - Hclust: Hclust object generated from the distance matrix. Populated by `RunCORE`.
  - PutativeClusters: Clusters generated by an unsupervised dynamic tree cut to the Hclust object. Generated by `RunCORE` function.
  - ClusteringMatrix: A matrix of clusters generated by cuts made at 40 heights, as a part of the `RunCORE` function.
  - Clusters: List of cells and their assigned cluster. Populated by `RunCORE`.
  - NumberOfClusters: Optimal number of clusters identified by `RunCORE`.
  - OptimalTreeHeight: Optimal tree height identified by `RunCORE`.
  - KeyStats: Data frame containing information used to determine the optimal number of clusters. Generated by `RunCORE`.
  - RandMatrix: Reformatted version of KeyStats, for use with `PlotStabilityDendro`.
- **Metrics** A list of values generated by the `GenerateMetrics` function.
- **Log** A record of functions used on an EMSet.

Creating an EMSet

Generally, an EMSet can be created by using the `NewEMSet` function. You can view a quick summary of this object by entering the name of the created object.

```
em.set <- NewEMSet(ExpressionMatrix = expression.matrix,
                   CellInformation = cell.information,
                   GeneInformation = gene.information,
                   Controls = control.list)
```

```
## [1] "Calculating control metrics..."
##
  |
  |                                                                      |   0%
  |
  |================================                                      |  50%
  |
  |======================================================================| 100%
```

```
em.set
```

```
## [1] "ascend Object - EMSet"
## [1] "Expression Matrix: 33020 genes and 1272 cells"
## [1] "Controls:"
## $Mt
##  [1] "MT-ND1"  "MT-ND2"  "MT-CO1"  "MT-CO2"  "MT-ATP8" "MT-ATP6" "MT-CO3"
##  [8] "MT-ND3"  "MT-ND4L" "MT-ND4"  "MT-ND5"  "MT-ND6"  "MT-CYB"
##
## $Rb
##   [1] "RPL22"          "RPL11"          "RPS6KA1"        "RPS8"
##   [5] "RPL5"           "RPS27"          "RPS6KC1"        "RPS7"
##   [9] "RPS27A"         "RPL31"          "RPL37A"         "RPL32"
##  [13] "RPL15"          "RPSA"           "RPL14"          "RPL29"
##  [17] "RPL24"          "RPL22L1"        "RPL39L"         "RPL35A"
##  [21] "RPL9"           "RPL34-AS1"      "RPL34"          "RPS3A"
##  [25] "RPL37"          "RPS23"          "RPS14"          "RPL26L1"
##  [29] "RPS18"          "RPS10-NUDT3"    "RPS10"          "RPL10A"
##  [33] "RPL7L1"         "RPS12"          "RPS6KA2"        "RPS6KA2-AS1"
##  [37] "RPS6KA3"        "RPS4X"          "RPS6KA6"        "RPL36A"
##  [41] "RPL36A-HNRNPH2" "RPL39"          "RPL10"          "RPS20"
##  [45] "RPL7"           "RPL30"          "RPL8"           "RPS6"
##  [49] "RPL35"          "RPL12"          "RPL7A"          "RPLP2"
##  [53] "RPL27A"         "RPS13"          "RPS6KA4"        "RPS6KB2"
##  [57] "RPS3"           "RPS25"          "RPS24"          "RPS26"
##  [61] "RPL41"          "RPL6"           "RPLP0"          "RPL21"
##  [65] "RPL10L"         "RPS29"          "RPL36AL"        "RPS6KL1"
##  [69] "RPS6KA5"        "RPS27L"         "RPL4"           "RPLP1"
##  [73] "RPS17"          "RPL3L"          "RPS2"           "RPS15A"
##  [77] "RPL13"          "RPL26"          "RPL23A"         "RPL23"
##  [81] "RPL19"          "RPL27"          "RPS6KB1"        "RPL38"
##  [85] "RPL17-C18orf32" "RPL17"          "RPS21"          "RPS15"
##  [89] "RPL36"          "RPS28"          "RPL18A"         "RPS16"
##  [93] "RPS19"          "RPL18"          "RPL13A"         "RPS11"
##  [97] "RPS9"           "RPL28"          "RPS5"           "RPS4Y1"
## [101] "RPS4Y2"         "RPL3"           "RPS19BP1"
```

Refer to the vignette An introduction to ascend - Processing and analysis of retinal ganglion cells for a walkthrough on how to create an EMSet.

Manipulating an EMSet

The `ascend` package comes with functions dedicated to the retrieval and modification of data stored in `EMSets`.

Retrieving data from an EMSet

The following functions can be used to retrieve data from specific slots in an `EMSet`.

Slot	Function
Expression matrix	GetExpressionMatrix
Cell information	GetCellInfo
Gene information	GetGeneInfo

Slot	Function
Controls	GetControls
PCA matrix	GetPCA
Distance matrix	GetDistanceMatrix
Hclust object	GetHclust
Rand Matrix	GetRandMatrix
Log	DisplayLog

Modifying contents of an EMSet

The following functions can be used to modify the contents of an EMSet.

Slot	Function
Expression matrix	ReplaceExpressionMatrix
Cell information	ReplaceCellInfo
Gene information	ReplaceGeneInfo
Controls	UpdateControls

These functions follow the same syntax of function(EMSet, name of replacement object).

For example, the `ReplaceCellInfo` function has the following syntax:

```
updated.em.set <- ReplaceCellInfo(em.set, new.cell.info)
```

To change or remove controls from a dataset, use `UpdateControls`. Please note that this function just removes the use of controls in an EMSet; transcript counts associated with controls will not be removed.

```
old.controls <- GetControls(em.set)
new.controls <- c(old.controls, list(ERCC = c("ERCC-00031",
                                              "ERCC-00017",
                                              "ERCC-00024")))
updated.em.set <- UpdateControls(em.set, new.controls)
```

To replace the expression matrix in a pre-existing EMSet, use `ReplaceExpressionMatrix`. Please note that replacing an expression matrix will recalculate the metrics associated with a dataset.

```
updated.em.set <- ReplaceExpressionMatrix(em.set, new.matrix)
```

Subsetting from an EMSet

The following functions can be used to subset data from an `EMSet` based on a condition.

To subset cells by batch, use `SubsetBatch`.

```
subset.batch <- SubsetBatch(em.set, batches = c("1", "2"))
```

To subset cells by cluster, use `SubsetCluster`.

```
subset.clusters <- SubsetCluster(em.set, clusters = c("2", "3"))
```

To subset cells by a condition, use `SubsetCondition`.

```
thy1.set <- SubsetCondition(em.set, conditions = c("THY1"))
```

To subset cells by a list of cell identifiers, use `SubsetCells`.

```
# Retrieve cell information from an EMSet
cell.info <- GetCellInfo(em.set)

# Randomly sample 100 cell barcodes to isolate.
cell.barcodes <- sample(cell.info$cell_barcode, 100, replace = FALSE)
hundred.cell.set <- SubsetCells(em.set, cell_barcodes = cell.barcodes)
```

ascend Plotting Functions

ascend has the following plotting functions:

Function	Description
PlotGeneralQC	Generates a series of plots related to quality control using ggplot2 and ggbeeswarm
PlotTopGeneExpression	Generates a boxplot representing the most expressed genes in the dataset using ggplot2
PlotPCA	Generates a scatter plot using ggplot2
PlotTSNE	Generates a scatter plot using ggplot2
PlotMDS	Generates a scatter plot using ggplot2
PlotPCAVariance	Generates a scree plot using ggplot2
PlotStabilityDendro	Generates a dendrogram with bars using dendextend
PlotStability	Generates a line plot using ggplot2
PlotDendrogram	Generates a cluster-labelled dendrogram using dendextend
PlotDEVolcano	Generates a scatter plot using ggplot2

Plots that use ggplot2 can be treated as normal ggplot2 objects where elements can be added after the plots are generated. Refer to the ggplot2 documentation for more information.

Adding information to PCA, TSNE and MDS plots

PCA, TSNE and MDS plots depict relative distances between cells based on expression. Additional information can be added to these plots by highlighting cells based on a condition stored in the Cell Information slot of the `EMSet` object.

Example: Colour cells by cluster

```
library(ggplot2)
# PCA PLOT
pca.plot <- PlotPCA(em.set, dim1 = 1, dim2 = 2, condition = "cluster")
pca.plot <- pca.plot + scale_color_manual(values=c("#bb5f4c", "#8e5db0", "#729b57"))
pca.plot <- pca.plot + ggtitle("PCA Plot", subtitle = "Labelled by batch")

# MDS PLOT
mds.plot <- PlotMDS(em.set, PCA = FALSE, dim1 = 1, dim2 = 2, condition = "cluster")
```

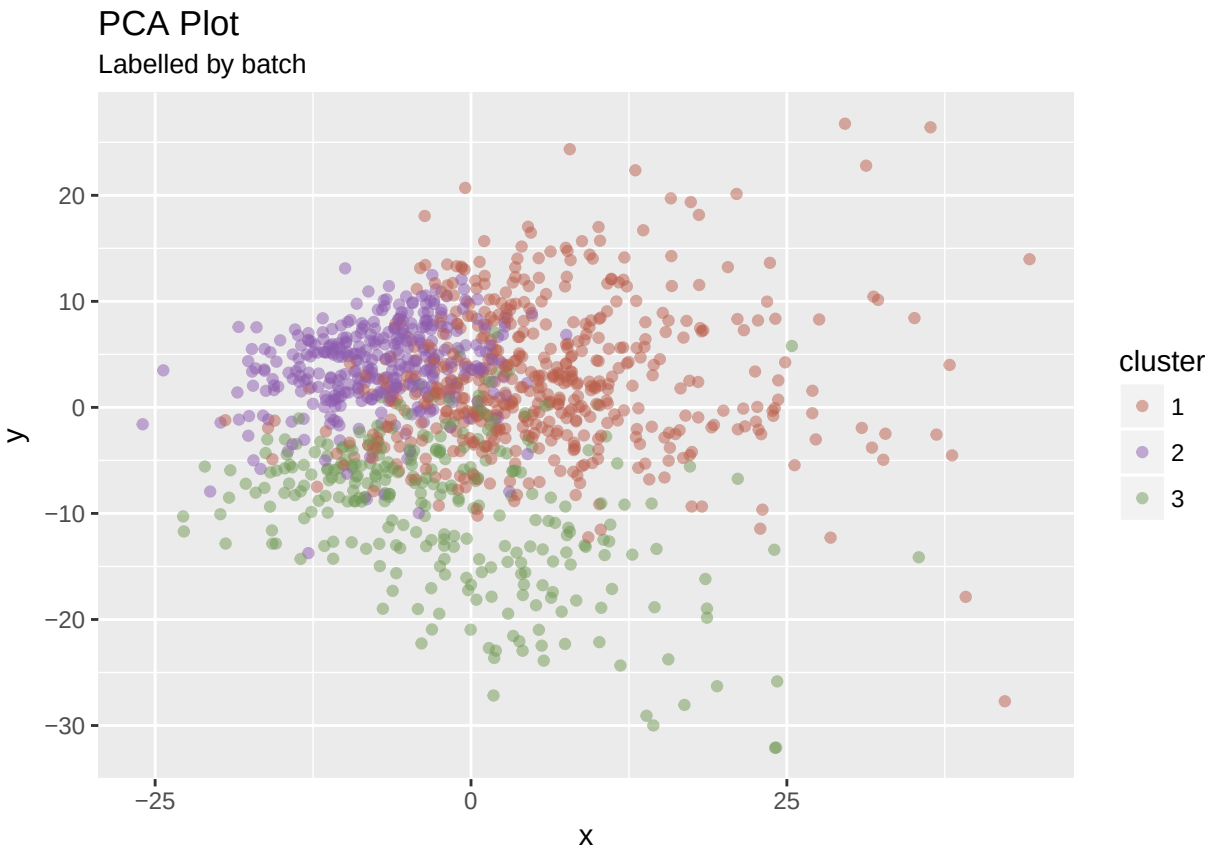
```
## [1] "Calculating distance matrix from expression data..."
## [1] "Running cmdscale..."
## [1] "Cmdscale complete! Processing scaled data..."
## [1] "Generating MDS plot..."
```

```
mds.plot <- mds.plot + scale_color_manual(values=c("#bb5f4c", "#8e5db0", "#729b57"))
mds.plot <- mds.plot + ggtitle("MDS Plot", subtitle = "Labelled by batch")
```

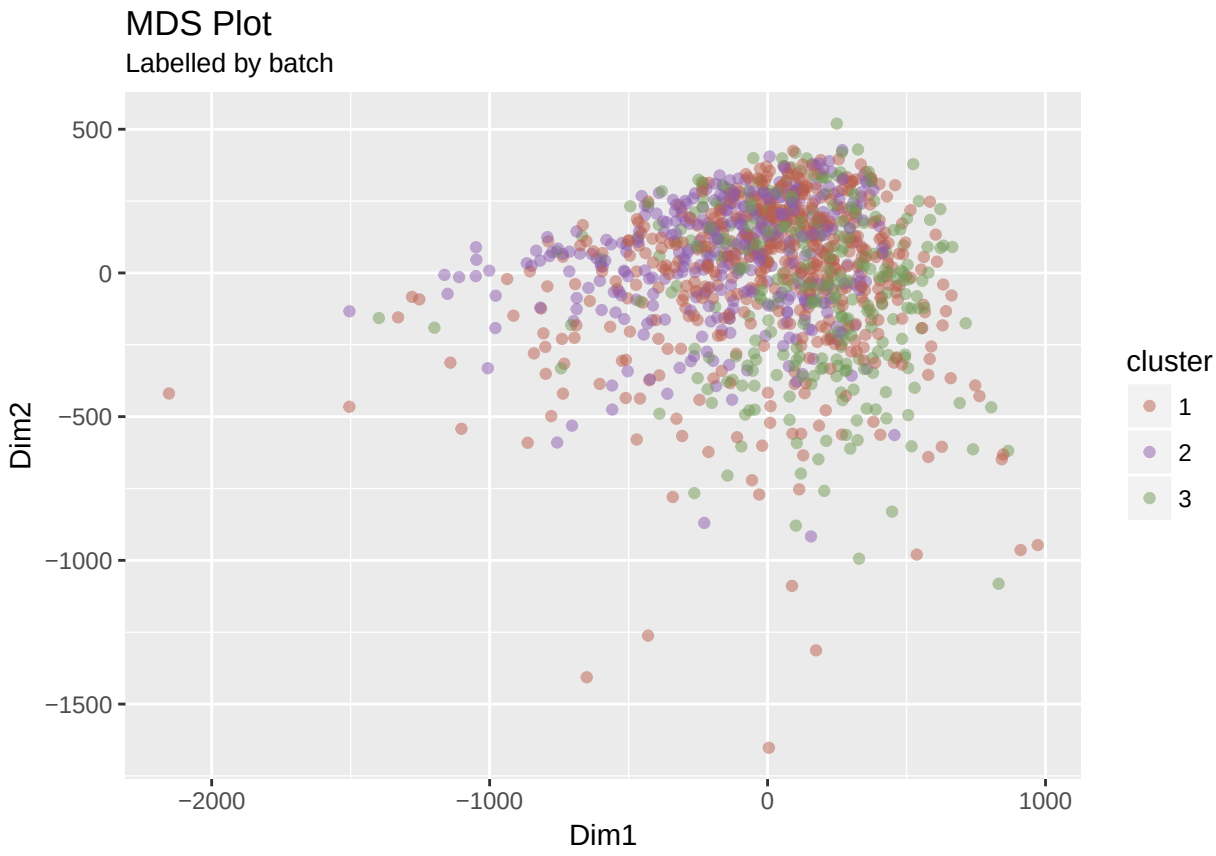
```
# TSNE PLOT
tsne.plot <- PlotTSNE(em.set,
                      PCA = TRUE,
                      condition = "cluster",
                      seed = 0, perplexity = 30, theta = 0.5)
```

```
## [1] "Running Rtsne..."
## [1] "Rtsne complete! Returning matrix..."
```

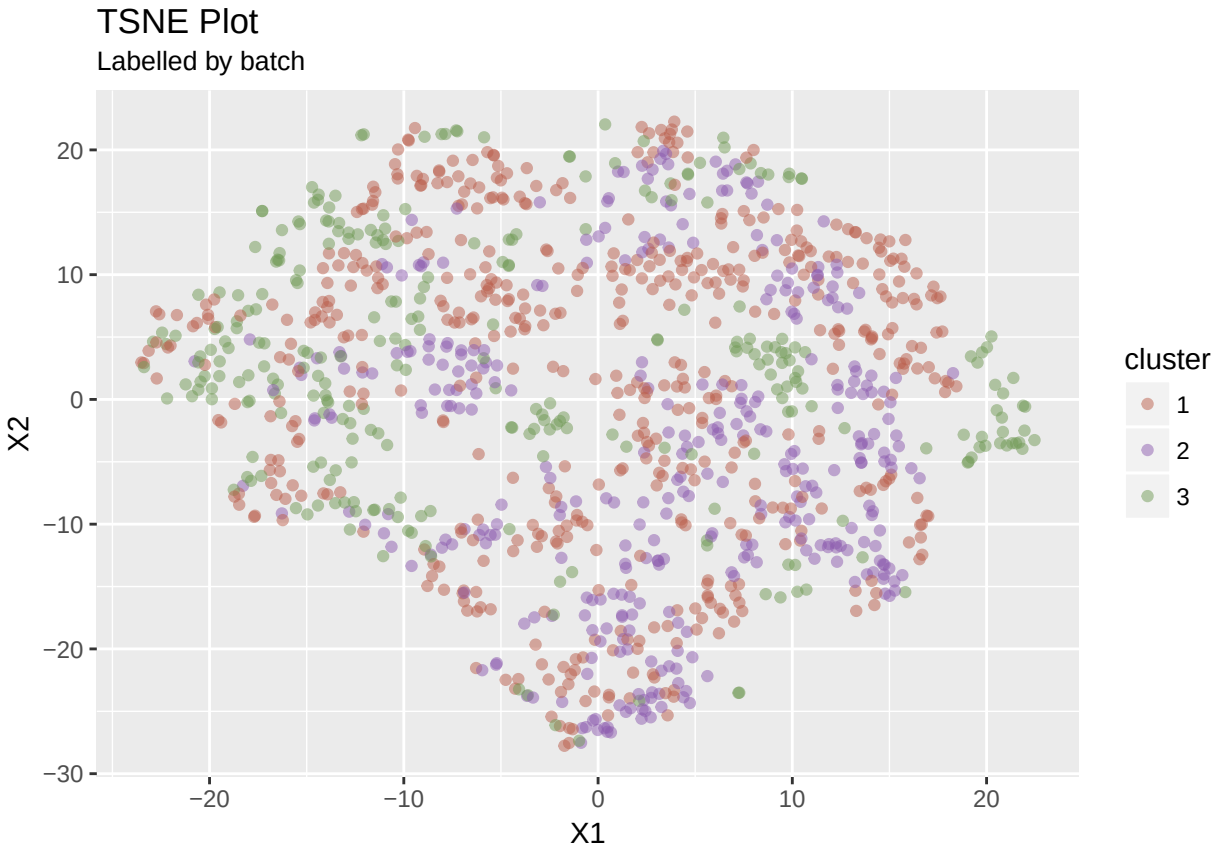
```
tsne.plot <- tsne.plot + scale_color_manual(values=c("#bb5f4c", "#8e5db0", "#729b57"))
tsne.plot <- tsne.plot + ggtitle("TSNE Plot", subtitle = "Labelled by batch")

print(pca.plot)
```




```
print(mds.plot)
```




```
print(tsne.plot)
```



Using data from ascend with other R packages

scran and scater

As ascend uses functions from `scran` and `scater`, an EMSet can be converted in to a `SCESet` with the function `ConvertToScater`.

```
sce.set <- ConvertToScater(em.set)
sce.set
```

```
## SCESet (storageMode: lockedEnvironment)
## assayData: 32904 features, 1174 samples
##   element names: counts, exprs
## protocolData: none
## phenoData: none
## featureData: none
## experimentData: use 'experimentData(object)'
## Annotation:
```