ViDGER Supplementary Material

Brandon Monier\textsuperscript{1,2}, Adam McDermaid\textsuperscript{1,3,4}, Jing Zhao\textsuperscript{5,6}, and Qin Ma\textsuperscript{1,3,7,8}

\textsuperscript{1}South Dakota State University
\textsuperscript{2}Department of Biology and Microbiology
\textsuperscript{3}Bioinformatics and Mathematical Biosciences Lab
\textsuperscript{4}Department of Mathematics and Statistics
\textsuperscript{5}Population Health Group, Sanford Research
\textsuperscript{6}Department of Internal Medicine, Sanford School of Medicine
\textsuperscript{7}Department of Agronomy, Horticulture, and Plant Science
\textsuperscript{8}BioSNTR

2018-07-27

Abstract

Differential gene expression (DGE) is one of the most common applications of RNA-sequencing (RNA-seq) data. This process allows for the elucidation of differentially expressed genes (DEGs) across two or more conditions. Interpretation of the DGE results can be non-intuitive and time consuming due to the variety of formats based on the tool of choice and the numerous pieces of information provided in these results files. Here we present an R package, ViDGER (Visualization of Differential Gene Expression Results using R), which contains nine functions that generate information-rich visualizations for the interpretation of DGE results from three widely-used tools, Cuffdiff, DESeq2, and edgeR.

Contents

Example S1: Installation and data examples ..................... 4
   An overview of the data used .................................. 5
Example S2: Creating box plots ................................. 6
   With Cuffdiff .................................................. 6
   With DESeq2 .................................................. 7
   With edgeR ..................................................... 8
   Aesthetic variants to box plots ................................ 9
   Color palette variants to box plots .......................... 15
Example S3: Creating scatter plots ............................. 18
   With Cuffdiff .................................................. 18
   With DESeq2 .................................................. 19
   With edgeR ..................................................... 20
Highlighting with \texttt{vsFourWay()} \hspace{1cm} \textbf{47}

Example S12: Extracting datasets from plots \hspace{1cm} \textbf{48}

\hspace{1cm} Overview. \hspace{1cm} \textbf{48}
\hspace{1cm} The data extraction process. \hspace{1cm} \textbf{48}
\hspace{1cm} Return the plot. \hspace{1cm} \textbf{48}

Example S13: Changing text sizes \hspace{1cm} \textbf{50}

\hspace{1cm} Overview. \hspace{1cm} \textbf{50}
\hspace{1cm} What exactly can you manipulate? \hspace{1cm} \textbf{50}

Method S1: Determining data point shape and size changes. \hspace{1cm} \textbf{52}

Method S2: Determining function performance. \hspace{1cm} \textbf{53}

\hspace{1cm} Scatterplots \hspace{1cm} \textbf{53}
\hspace{1cm} Scatterplot matrices \hspace{1cm} \textbf{54}
\hspace{1cm} Box plots \hspace{1cm} \textbf{55}
\hspace{1cm} Differential gene expression matrices \hspace{1cm} \textbf{56}
\hspace{1cm} Volcano plots \hspace{1cm} \textbf{57}
\hspace{1cm} Volcano plot matrices \hspace{1cm} \textbf{58}
\hspace{1cm} MA plots \hspace{1cm} \textbf{59}
\hspace{1cm} MA matrices \hspace{1cm} \textbf{60}
\hspace{1cm} Four way plots \hspace{1cm} \textbf{61}

Session info \hspace{1cm} \textbf{62}
Example S1: Installation and data examples

The stable version of this package is available on Bioconductor. You can install it by running the following:

```r
source("http://bioconductor.org/biocLite.R")
biocLite("vidger")
```

The latest developmental version of ViDGER can be installed via GitHub using the devtools package:

```r
if (!require("devtools")) install.packages("devtools")
devtools::install_github("btmonier/vidger", ref = "devel")
```

Once installed, you will have access to the following functions:

- `vsBoxplot()`
- `vsScatterPlot()`
- `vsScatterMatrix()`
- `vsDEGMatrix()`
- `vsMAPlot()`
- `vsMAMatrix()`
- `vsVolcano()`
- `vsVolcanoMatrix()`
- `vsFourWay()`

Further explanation will be given to how these functions work later on in the documentation. For the following examples, three toy data sets will be used: `df.cuff`, `df.deseq`, and `df.edger`. Each of these data sets reflect the three RNA-seq analyses this package covers. These can be loaded in the R workspace by using the following command:

```r
data(<data_set>)
```

Where `<data_set>` is one of the previously mentioned data sets. Some of the recurring elements that are found in each of these functions are the `type` and `d.factor` arguments. The `type` argument tells the function how to process the data for each analytical type (i.e. "cuffdiff", "deseq", or "edger"). The `d.factor` argument is used specifically for DESeq2 objects which we will discuss in the DESeq2 section. All other arguments are discussed in further detail by looking at the respective help file for each functions (i.e. `?vsScatterPlot`).
An overview of the data used

As mentioned earlier, three toy data sets are included with this package. Summaries of this data can be found in the following tables:

Table 1: An overview of the toy data sets included in this package. In this table, each data set is summarized in terms of what analytical software was used, organism ID, experimental layout (replicates and treatments), number of transcripts (IDs), and size of the data object in terms of megabytes (MB).

<table>
<thead>
<tr>
<th>Data</th>
<th>Software</th>
<th>Organism</th>
<th>Reps</th>
<th>Treat.</th>
<th>IDs</th>
<th>Size (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>df.cuff</td>
<td>CuffDiff</td>
<td><em>H. sapiens</em></td>
<td>2</td>
<td>3</td>
<td>1200</td>
<td>0.2</td>
</tr>
<tr>
<td>df.deseq</td>
<td>DESeq2</td>
<td><em>D. melanogaster</em></td>
<td>2</td>
<td>3</td>
<td>29391</td>
<td>2.3</td>
</tr>
<tr>
<td>df.deseq</td>
<td>edgeR</td>
<td><em>A. thaliana</em></td>
<td>2</td>
<td>3</td>
<td>724</td>
<td>0.1</td>
</tr>
</tbody>
</table>
Example S2: Creating box plots

Box plots are a useful way to determine the distribution of data. In this case we can determine the distribution of FPKM or CPM values by using the `vsBoxPlot()` function. This function allows you to extract necessary results-based data from analytical objects to create a box plot comparing $\log_{10}$ (FPKM or CPM) distributions for experimental treatments.

With Cuffdiff

```r
vsBoxPlot(
  data = df.cuff, d.factor = NULL, type = 'cuffdiff', title = TRUE,
  legend = TRUE, grid = TRUE
)
```

![Box plot example](image)

**Figure 1:** A box plot example using the `vsBoxPlot()` function with `cuffdiff` data

In this example, FPKM distributions for each treatment within an experiment are shown in the form of a box and whisker plot.
With DESeq2

```r
vsBoxPlot(
  data = df.deseq, d.factor = 'condition', type = 'deseq',
  title = TRUE, legend = TRUE, grid = TRUE
)
```

**Figure 2: A box plot example using the `vsBoxPlot()` function with DESeq2 data**

In this example, FPKM distributions for each treatment within an experiment are shown in the form of a box and whisker plot.
With edgeR

```r
vsBoxPlot(
    data = df.edger, d.factor = NULL, type = 'edger',
    title = TRUE, legend = TRUE, grid = TRUE
)
```

**Figure 3**: A box plot example using the `vsBoxPlot()` function with `edgeR` data

In this example, CPM distributions for each treatment within an experiment are shown in the form of a box and whisker plot.
Aesthetic variants to box plots

`vsBoxPlot()` can allow for different iterations to showcase data distribution. These changes can be implemented using the `aes` parameter. Currently, there are 6 different variants:

- **box**: standard box plot
- **violin**: violin plot
- **boxdot**: box plot with dot plot overlay
- **viiodot**: violin plot with dot plot overlay
- **viosumm**: violin plot with summary stats overlay
- **notch**: box plot with notch

**box variant**

```r
data("df.edger")
vsBoxPlot(
  data = df.edger, d.factor = NULL, type = "edger", title = TRUE,
  legend = TRUE, grid = TRUE, aes = "box"
)
```

![Box plot example](image)

**Figure 4**: A box plot example using the `aes` parameter: `box`
violin variant

```r
data("df.edger")
vsBoxPlot(
  data = df.edger, d.factor = NULL, type = "edger", title = TRUE,
  legend = TRUE, grid = TRUE, aes = "violin"
)
```

Figure 5: A box plot example using the 'aes' parameter: 'violin'
Figure 6:  A box plot example using the 'aes' parameter: 'boxdot'
Figure 7: A box plot example using the 'aes' parameter: 'viodot'
data("df.edger")
vsBoxPlot(
data = df.edger, d.factor = NULL, type = "edger", title = TRUE,
legend = TRUE, grid = TRUE, aes = "viosumm"
)

Figure 8: A box plot example using the 'aes' parameter: 'viosumm'
notch variant

```r
data("df.edger")
vsBoxPlot(
  data = df.edger, d.factor = NULL, type = "edger", title = TRUE,
  legend = TRUE, grid = TRUE, aes = "notch"
)
```

**Figure 9:** A box plot example using the `aes` parameter: `notch`
Color palette variants to box plots

In addition to aesthetic changes, the fill color of each variant can also be changed. This can be implemented by modifying the `fill.color` parameter.

The palettes that can be used for this parameter are based off of the palettes found in the RColorBrewer package. A visual list of all the palettes can be found [here](#).

Color variant example 1

```r
data("df.edger")
vsBoxPlot(
    data = df.edger, d.factor = NULL, type = "edger", title = TRUE,
    legend = TRUE, grid = TRUE, aes = "box", fill.color = "RdGy"
)
```

![Box plot example using the 'fill.color' parameter: 'RdGy'.](#)

**Figure 10: Color variant 1**

A box plot example using the ‘fill.color’ parameter: ‘RdGy’.
Color variant example 2

```r
data("df.edger")
vsBoxPlot(
    data = df.edger, d.factor = NULL, type = "edger", title = TRUE,
    legend = TRUE, grid = TRUE, aes = "viosumm", fill.color = "Paired"
)
```

![Violin plot example using the 'fill.color' parameter: 'Paired' with the 'aes' parameter: 'viosumm'.](image)

**Figure 11: Color variant 2**
A violin plot example using the ‘fill.color’ parameter: ‘Paired’ with the ‘aes’ parameter: ‘viosumm’.
Color variant example 3

```r
data("df.edger")
vsBoxPlot(
  data = df.edger, d.factor = NULL, type = "edger", title = TRUE,
  legend = TRUE, grid = TRUE, aes = "notch", fill.color = "Greys"
)
```

![Box plot example using 'fill.color' parameter: 'Greys' with 'aes' parameter: 'notch'. Using these parameters, we can also generate grey-scale plots.](image)

**Figure 12: Color variant 3**
A notched box plot example using the 'fill.color' parameter: 'Greys' with the 'aes' parameter: 'notch'. Using these parameters, we can also generate grey-scale plots.
Example S3: Creating scatter plots

This example will look at a basic scatter plot function, `vsScatterPlot()`. This function allows you to visualize comparisons of $\log_{10}$ values of either FPKM or CPM measurements of two treatments depending on analytical type.

With Cuffdiff

```r
vsScatterPlot(
    x = 'hESC', y = 'iPS', data = df.cuff, type = 'cuffdiff',
    d.factor = NULL, title = TRUE, grid = TRUE
)
```

**Figure 13:** A scatterplot example using the `vsScatterPlot()` function with 'Cuffdiff' data

In this visualization, $\log_{10}$ comparisons are made of fragments per kilobase of transcript per million mapped reads (FPKM) measurements. The dashed line represents regression line for the comparison.
With DESeq2

```r
vsScatterPlot(
  x = 'treated_paired.end', y = 'untreated_paired.end',
  data = df.deseq, type = 'deseq', d.factor = 'condition',
  title = TRUE, grid = TRUE
)
```

**Figure 14:** A scatterplot example using the `vsScatterPlot()` function with `DESeq2` data

In this visualization, $\log_{10}$ comparisons are made of fragments per kilobase of transcript per million mapped reads (FPKM) measurements. The dashed line represents regression line for the comparison.
With edgeR

```r
vsScatterPlot(
  x = 'WM', y = 'MM', data = df.edger, type = 'edger',
  d.factor = NULL, title = TRUE, grid = TRUE
)
```

**Figure 15: A scatterplot example using the 'vsScatterPlot()' function with 'edgeR' data**

In this visualization, \( \log_{10} \) comparisons are made of fragments per kilobase of transcript per million mapped reads (FPKM) measurements. The dashed line represents regression line for the comparison.
Example S4: Creating scatter plot matrices

This example will look at an extension of the `vsScatterPlot()` function which is `vsScatterMatrix()`. This function will create a matrix of all possible comparisons of treatments within an experiment with additional info.

With Cuffdiff

```r
vsScatterMatrix(
    data = df.cuff, d.factor = NULL, type = 'cuffdiff',
    comp = NULL, title = TRUE, grid = TRUE, man.title = NULL
)
```

Figure 16: A scatterplot matrix example using the `vsScatterMatrix()` function with 'Cuffdiff' data

Similar to the scatterplot function, this visualization allows for all comparisons to be made within an experiment. In addition to the scatterplot visuals, FPKM distributions (histograms) and correlation (Corr) values are generated.
With DESeq2

```r
vsScatterMatrix(
  data = df.deseq, d.factor = 'condition', type = 'deseq',
  comp = NULL, title = TRUE, grid = TRUE, man.title = NULL
)
```

Figure 17: A scatterplot matrix example using the `vsScatterMatrix()` function with `DESeq2` data. Similar to the scatterplot function, this visualization allows for all comparisons to be made within an experiment. In addition to the scatterplot visuals, FPKM distributions (histograms) and correlation (Corr) values are generated.
With edgeR

```r
vsScatterMatrix(
    data = df.edger, d.factor = NULL, type = 'edger', comp = NULL,
    title = TRUE, grid = TRUE, man.title = NULL
)
```

**Figure 18:** A scatterplot matrix example using the `vsScatterMatrix()` function with `edgeR` data

Similar to the scatterplot function, this visualization allows for all comparisons to be made within an experiment. In addition to the scatterplot visuals, FPKM distributions (histograms) and correlation (Corr) values are generated.
Example S5: Creating differential gene expression matrices

Using the `vsDEGMatrix()` function allows the user to visualize the number of differentially expressed genes (DEGs) at a given adjusted *p*-value (padj = ) for each experimental treatment level. Higher color intensity correlates to a higher number of DEGs.

With Cuffdiff

```r
vsDEGMatrix(
  data = df.cuff, padj = 0.05, d.factor = NULL, type = 'cuffdiff',
  title = TRUE, legend = TRUE, grid = TRUE
)
```

![Figure 19: A matrix of differentially expressed genes (DEGs) at a given *p*-value using the 'vsDEG-Matrix()' function with 'Cuffdiff' data](image)

With this function, the user is able to visualize the number of DEGs at a given adjusted *p*-value for each experimental treatment level. Higher color intensity correlates to a higher number of DEGs.
With DESeq2

```r
vsDEGMatrix(
  data = df.deseq, padj = 0.05, d.factor = 'condition',
  type = 'deseq', title = TRUE, legend = TRUE, grid = TRUE
)
```

**Figure 20:** A matrix of differentially expressed genes (DEGs) at a given *p*-value using the 'vsDEG-Matrix()' function with 'DESeq2' data.

With this function, the user is able to visualize the number of DEGs at a given adjusted *p*-value for each experimental treatment level. Higher color intensity correlates to a higher number of DEGs.
ViGER Supplementary Material

With edgeR

```r
vsDEGMatrix(
    data = df.edger, padj = 0.05, d.factor = NULL, type = 'edger',
    title = TRUE, legend = TRUE, grid = TRUE
)
```

![Significant transcripts at $\alpha = 0.05$](image)

**Figure 21:** A matrix of differentially expressed genes (DEGs) at a given *p*-value using the `vsDEGMatrix()` function with `edgeR` data. With this function, the user is able to visualize the number of DEGs at a given adjusted *p*-value for each experimental treatment level. Higher color intensity correlates to a higher number of DEGs.
Grey-scale DEG matrices

A grey-scale option is available for this function if you wish to use a grey-to-white gradient instead of the classic blue-to-white gradient. This can be invoked by setting the `grey.scale` parameter to `TRUE`.

```r
vsDEGMatrix(data = df.deseq, d.factor = "condition", type = "deseq", grey.scale = TRUE)
```

Significant transcripts at $\alpha = 0.05$

![Graph showing transcript counts under different conditions](image)
Example S6: Creating MA plots

`vsMAPlot()` visualizes the variance between two samples in terms of gene expression values where logarithmic fold changes of count data are plotted against mean counts. For more information on how each of the aesthetics are plotted, please refer to the figure captions and Method S1.

With Cuffdiff

```r
vsMAPlot(
  x = 'iPS', y = 'hESC', data = df.cuff, d.factor = NULL,
  type = 'cuffdiff', padj = 0.05, y.lim = NULL, lfc = NULL,
  title = TRUE, legend = TRUE, grid = TRUE
)
```

![MA plot visualization using the 'vsMAPlot()' function with 'Cuffdiff' data](image)

Figure 22: MA plot visualization using the `vsMAPlot()` function with 'Cuffdiff' data

LFCs are plotted mean counts to determine the variance between two treatments in terms of gene expression. Blue nodes on the graph represent statistically significant LFCs which are greater than a given value than a user-defined LFC parameter. Green nodes indicate statistically significant LFCs which are less than the user-defined LFC parameter. Gray nodes are data points that are not statistically significant. Numerical values in parantheses for each legend color indicate the number of transcripts that meet the prior conditions. Triangular shapes represent values that exceed the viewing area of the graph. Node size changes represent the magnitude of the LFC values (i.e. larger shapes reflect larger LFC values). Dashed lines indicate user-defined LFC values.
With DESeq2

```r
vsMAPlot(
  x = 'treated_paired.end', y = 'untreated_paired.end',
  data = df.deseq, d.factor = 'condition', type = 'deseq',
  padj = 0.05, y.lim = NULL, lfc = NULL, title = TRUE,
  legend = TRUE, grid = TRUE
)
```

Figure 23: MA plot visualization using the `vsMAPlot()` function with `DESeq2` data

LFCs are plotted mean counts to determine the variance between two treatments in terms of gene expression. Blue nodes on the graph represent statistically significant LFCs which are greater than a given value than a user-defined LFC parameter. Green nodes indicate statistically significant LFCs which are less than the user-defined LFC parameter. Gray nodes are data points that are not statistically significant. Numerical values in parantheses for each legend color indicate the number of transcripts that meet the prior conditions. Triangular shapes represent values that exceed the viewing area of the graph. Node size changes represent the magnitude of the LFC values (i.e. larger shapes reflect larger LFC values). Dashed lines indicate user-defined LFC values.
viDGER Supplementary Material

With edgeR

```r
vsMAPlot(
  x = 'WW', y = 'MM', data = df.edger, d.factor = NULL,
  type = 'edger', padj = 0.05, y.lim = NULL, lfc = NULL,
  title = TRUE, legend = TRUE, grid = TRUE
)
```

![MA Plot Visualization](image)

**Figure 24: MA plot visualization using the `vsMAPlot()` function with `edgeR` data**

LFCs are plotted mean counts to determine the variance between two treatments in terms of gene expression. Blue nodes on the graph represent statistically significant LFCs which are greater than a given value than a user-defined LFC parameter. Green nodes indicate statistically significant LFCs which are less than the user-defined LFC parameter. Gray nodes are data points that are not statistically significant. Numerical values in parentheses for each legend color indicate the number of transcripts that meet the prior conditions. Triangular shapes represent values that exceed the viewing area of the graph. Node size changes represent the magnitude of the LFC values (i.e. larger shapes reflect larger LFC values). Dashed lines indicate user-defined LFC values.
Example S7: Creating MA plot matrices

Similar to a scatter plot matrix, vsMAMatrix() will produce visualizations for all comparisons within your data set. For more information on how the aesthetics are plotted in these visualizations, please refer to the figure caption and Method S1.

With Cuffdiff

```r
vsMAMatrix(
    data = df.cuff, d.factor = NULL, type = 'cuffdiff',
    padj = 0.05, y.lim = NULL, lfc = 1, title = TRUE,
    grid = TRUE, counts = TRUE, data.return = FALSE
)
```

Figure 25: A MA plot matrix using the ‘vsMAMatrix()’ function with ‘Cuffdiff’ data

Similar to the ‘vsMAPlot()’ function, ‘vsMAMatrix()’ will generate a matrix of MA plots for all comparisons within an experiment. LFCs are plotted mean counts to determine the variance between two treatments in terms of gene expression. Blue nodes on the graph represent statistically significant LFCs which are greater than a given value than a user-defined LFC parameter. Green nodes indicate statistically significant LFCs which are less than the user-defined LFC parameter. Gray nodes are data points that are not statistically significant. Numerical values in parantheses for each legend color indicate the number of transcripts that meet the prior conditions. Triangular shapes represent values that exceed the viewing area of the graph. Node size changes represent the magnitude of the LFC values (i.e. larger shapes reflect larger LFC values). Dashed lines indicate user-defined LFC values.
**ViDGER Supplementary Material**

**With DESeq2**

```r
vsMAMatrix(
    data = df.deseq, d.factor = 'condition', type = 'deseq',
    padj = 0.05, ylim = NULL, lfc = 1, title = TRUE,
    grid = TRUE, counts = TRUE, data.return = FALSE
)
```

---

**Figure 26: A MA plot matrix using the `vsMAMatrix()` function with ‘DESeq2’ data**

Similar to the `vsMAPlot()` function, `vsMAMatrix()` will generate a matrix of MA plots for all comparisons within an experiment. LFCs are plotted mean counts to determine the variance between two treatments in terms of gene expression. Blue nodes on the graph represent statistically significant LFCs which are greater than a given value than a user-defined LFC parameter. Green nodes indicate statistically significant LFCs which are less than the user-defined LFC parameter. Gray nodes are data points that are not statistically significant. Numerical values in parentheses for each legend color indicate the number of transcripts that meet the prior conditions. Triangular shapes represent values that exceed the viewing area of the graph. Node size changes represent the magnitude of the LFC values (i.e. larger shapes reflect larger LFC values). Dashed lines indicate user-defined LFC values.
With edgeR

\[
\text{vsMAMatrix(}
\text{data = df.edger, d.factor = NULL, type = 'edger', padj = 0.05, y.lim = NULL, lfc = 1, title = TRUE, grid = TRUE, counts = TRUE, data.return = FALSE}
\]

**Figure 27: A MA plot matrix using the `vsMAMatrix()` function with `edgeR` data**

Similar to the `vsMAPlot()` function, `vsMAMatrix()` will generate a matrix of MA plots for all comparisons within an experiment. LFCs are plotted mean counts to determine the variance between two treatments in terms of gene expression. Blue nodes on the graph represent statistically significant LFCs which are greater than a given value than a user-defined LFC parameter. Green nodes indicate statistically significant LFCs which are less than the user-defined LFC parameter. Gray nodes are data points that are not statistically significant. Numerical values in parantheses for each legend color indicate the number of transcripts that meet the prior conditions. Triangular shapes represent values that exceed the viewing area of the graph. Node size changes represent the magnitude of the LFC values (i.e. larger shapes reflect larger LFC values). Dashed lines indicate user-defined LFC values.
Example S8: Creating volcano plots

The next few visualizations will focus on ways to display differential gene expression between two or more treatments. Volcano plots visualize the variance between two samples in terms of gene expression values where the $-\log_{10}$ of calculated $p$-values (y-axis) are plotted against the $\log_2$ changes (x-axis). These plots can be visualized with the `vsVolcano()` function. For more information on how each of the aesthetics are plotted, please refer to the figure captions and Method S1.

With Cuffdiff

```r
vsVolcano(
    x = 'iPS', y = 'hESC', data = df.cuff, d.factor = NULL,
    type = 'cuffdiff', padj = 0.05, x.lim = NULL, lfc = NULL,
    title = TRUE, legend = TRUE, grid = TRUE, data.return = FALSE
)
```

Figure 28: A volcano plot example using the `vsVolcano()` function with ‘Cuffdiff’ data
In this visualization, comparisons are made between the $-\log_{10}$ *p*-value versus the $\log_2$ fold change (LFC) between two treatments. Blue nodes on the graph represent statistically significant LFCs which are greater than a given value than a user-defined LFC parameter. Green nodes indicate statistically significant LFCs which are less than the user-defined LFC parameter. Gray nodes are data points that are not statistically significant. Numerical values in parantheses for each legend color indicate the number of transcripts that meet the prior conditions. Left and right brackets (< and >) represent values that exceed the viewing area of the graph. Node size changes represent the magnitude of the LFC values (i.e. larger shapes reflect larger LFC values). Vertical and horizontal lines indicate user-defined LFC and adjusted *p*-values, respectively.
With DESeq2

```r
vsVolcano(
  x = 'treated_paired.end', y = 'untreated_paired.end',
  data = df.deseq, d.factor = 'condition', type = 'deseq',
  padj = 0.05, x.lim = NULL, lfc = NULL, title = TRUE,
  legend = TRUE, grid = TRUE, data.return = FALSE
)
```

**Figure 29: A volcano plot example using the `vsVolcano()` function with DESeq2 data**

In this visualization, comparisons are made between the $-\log_{10}$ *p*-value versus the $\log_2$ fold change (LFC) between two treatments. Blue nodes on the graph represent statistically significant LFCs which are greater than a given value than a user-defined LFC parameter. Green nodes indicate statistically significant LFCs which are less than the user-defined LFC parameter. Gray nodes are data points that are not statistically significant. Numerical values in parantheses for each legend color indicate the number of transcripts that meet the prior conditions. Left and right brackets (< and >) represent values that exceed the viewing area of the graph. Node size changes represent the magnitude of the LFC values (i.e. larger shapes reflect larger LFC values). Vertical and horizontal lines indicate user-defined LFC and adjusted *p*-values, respectively.
With edgeR

```r
vsVolcano(
  x = 'WW', y = 'MM', data = df.edger, d.factor = NULL,
  type = 'edger', padj = 0.05, x.lim = NULL, lfc = NULL,
  title = TRUE, legend = TRUE, grid = TRUE, data.return = FALSE
)
```

**Figure 30: A volcano plot example using the 'vsVolcano()' function with 'edgeR' data**

In this visualization, comparisons are made between the $-\log_{10}$ *p*-value versus the log$_2$ fold change (LFC) between two treatments. Blue nodes on the graph represent statistically significant LFCs which are greater than a given value than a user-defined LFC parameter. Green nodes indicate statistically significant LFCs which are less than the user-defined LFC parameter. Gray nodes are data points that are not statistically significant. Numerical values in parantheses for each legend color indicate the number of transcripts that meet the prior conditions. Left and right brackets ($<$ and $>$) represent values that exceed the viewing area of the graph. Node size changes represent the magnitude of the LFC values (i.e. larger shapes reflect larger LFC values). Vertical and horizontal lines indicate user-defined LFC and adjusted *p*-values, respectively.
Example S9: Creating volcano plot matrices

Similar to the prior matrix functions, `vsVolcanoMatrix()` will produce visualizations for all comparisons within your data set. For more information on how the aesthetics are plotted in these visualizations, please refer to the figure caption and Method S1.

With Cuffdiff

```r
vsVolcanoMatrix(  
data = df.cuff, d.factor = NULL, type = 'cuffdiff',  
padj = 0.05, x.lim = NULL, lfc = NULL, title = TRUE,  
legend = TRUE, grid = TRUE, counts = TRUE
)
```

*Figure 31: A volcano plot matrix using the `vsVolcanoMatrix()` function with 'Cuffdiff' data*

Similar to the `vsVolcano()` function, `vsVolcanoMatrix()` will generate a matrix of volcano plots for all comparisons within an experiment. Comparisons are made between the $-\log_{10}$ *p*-value versus the $\log_2$ fold change (LFC) between two treatments. Blue nodes on the graph represent statistically significant LFCs which are greater than a given value than a user-defined LFC parameter. Green nodes indicate statistically significant LFCs which are less than the user-defined LFC parameter. Gray nodes are data points that are not statistically significant. The blue and green numbers in each facet represent the number of transcripts that meet the criteria for blue and green nodes in each comparison. Left and right brackets (< and >) represent values that exceed the viewing area of the graph. Node size changes represent the magnitude of the LFC values (i.e. larger shapes reflect larger LFC values). Vertical and horizontal lines indicate user-defined LFC and adjusted *p*-values, respectively.
**ViDGER Supplementary Material**

**With DESeq2**

```r
vsVolcanoMatrix(
    data = df.deseq, d.factor = 'condition', type = 'deseq',
    padj = 0.05, x.lim = NULL, lfc = NULL, title = TRUE,
    legend = TRUE, grid = TRUE, counts = TRUE
)
```

**Figure 32: A volcano plot matrix using the ‘vsVolcanoMatrix()’ function with ‘DESeq2’ data**

Similar to the ‘vsVolcano()’ function, ‘vsVolcanoMatrix()’ will generate a matrix of volcano plots for all comparisons within an experiment. Comparisons are made between the \(-\log_{10} \text{p-value}\) versus the \(\log_2\) fold change (LFC) between two treatments. Blue nodes on the graph represent statistically significant LFCs which are greater than a given value than a user-defined LFC parameter. Green nodes indicate statistically significant LFCs which are less than the user-defined LFC parameter. Gray nodes are data points that are not statistically significant. The blue and green numbers in each facet represent the number of transcripts that meet the criteria for blue and green nodes in each comparison. Left and right brackets (< and >) represent values that exceed the viewing area of the graph. Node size changes represent the magnitude of the LFC values (i.e. larger shapes reflect larger LFC values). Vertical and horizontal lines indicate user-defined LFC and adjusted \(*p\)-values, respectively.
With edgeR

```r
vsVolcanoMatrix(
    data = df.edger, d.factor = NULL, type = 'edger', padj = 0.05,
    x.lim = NULL, lfc = NULL, title = TRUE, legend = TRUE,
    grid = TRUE, counts = TRUE
)
```

Figure 33: A volcano plot matrix using the ‘vsVolcanoMatrix()’ function with ‘edgeR’ data

Similar to the ‘vsVolcano()’ function, ‘vsVolcanoMatrix()’ will generate a matrix of volcano plots for all comparisons within an experiment. Comparisons are made between the $-\log_{10}$ *p*-value versus the $\log_2$ fold change (LFC) between two treatments. Blue nodes on the graph represent statistically significant LFCs which are greater than a given value than a user-defined LFC parameter. Green nodes indicate statistically significant LFCs which are less than the user-defined LFC parameter. Gray nodes are data points that are not statistically significant. The blue and green numbers in each facet represent the number of transcripts that meet the criteria for blue and green nodes in each comparison. Left and right brackets (< and >) represent values that exceed the viewing area of the graph. Node size changes represent the magnitude of the LFC values (i.e. larger shapes reflect larger LFC values). Vertical and horizontal lines indicate user-defined LFC and adjusted *p*-values, respectively.
Example S10: Creating four way plots

To create four-way plots, the function, `vsFourWay()` is used. This plot compares the log2 fold changes between two samples and a 'control'. For more information on how each of the aesthetics are plotted, please refer to the figure captions and Method S1.

With Cuffdiff

```r
vsFourWay(x = 'iPS', y = 'hESC', control = 'Fibroblasts', data = df.cuff, d.factor = NULL, type = 'cuffdiff', padj = 0.05, x.lim = NULL, y.lim = NULL, lfc = NULL, legend = TRUE, title = TRUE, grid = TRUE)
```

Figure 34: A four way plot visualization using the `vsFourWay()` function with 'Cuffdiff' data
In this example, LFCs comparisons between two treatments and a control are made. Blue nodes indicate statistically significant LFCs which are greater than a given user-defined value for both x and y-axes. Green nodes reflect statistically significant LFCs which are less than a user-defined value for treatment y and greater than said value for treatment x. Similar to green nodes, red nodes reflect statistically significant LFCs which are greater than a user-defined value treatment y and less than said value for treatment x. Gray nodes are data points that are not statistically significant for both x and y-axes. Triangular shapes indicate values which exceed the viewing are for the graph. Size change reflects the magnitude of LFC values (i.e. larger shapes reflect larger LFC values). Vertical and horizontal dashed lines indicate user-defined LFC values.
With DESeq2

```r
vsFourWay(
  x = 'treated_paired.end', y = 'untreated_single.read',
  control = 'untreated_paired.end', data = df.deseq,
  d.factor = 'condition', type = 'deseq', padj = 0.05, x.lim = NULL,
  y.lim = NULL, lfc = NULL, legend = TRUE, title = TRUE, grid = TRUE
)
```

Figure 35: A four way plot visualization using the ‘vsFourWay()’ function with ‘DESeq2’ data

In this example, LFCs comparisons between two treatments and a control are made. Blue nodes indicate statistically significant LFCs which are greater than a given user-defined value for both x and y-axes. Green nodes reflect statistically significant LFCs which are less than a user-defined value for treatment y and greater than said value for treatment x. Similar to green nodes, red nodes reflect statistically significant LFCs which are greater than a user-defined value for treatment y and less than said value for treatment x. Gray nodes are data points that are not statistically significant for both x and y-axes. Triangular shapes indicate values which exceed the viewing area for the graph. Size change reflects the magnitude of LFC values (i.e. larger shapes reflect larger LFC values). Vertical and horizontal dashed lines indicate user-defined LFC values.
ViDGER Supplementary Material

With edgeR

```r
vsFourWay(
  x = 'WW', y = 'WM', control = 'MM', data = df.edger,
  d.factor = NULL, type = 'edger', padj = 0.05, x.lim = NULL,
  y.lim = NULL, lfc = NULL, legend = TRUE, title = TRUE, grid = TRUE
)
```

**Figure 36: A four way plot visualization using the ‘vsFourWay()’ function with ‘DESeq2’ data**

In this example, LFCs comparisons between two treatments and a control are made. Blue nodes indicate statistically significant LFCs which are greater than a given user-defined value for both x and y-axes. Green nodes reflect statistically significant LFCs which are less than a user-defined value for treatment y and greater than said value for treatment x. Similar to green nodes, red nodes reflect statistically significant LFCs which are greater than a user-defined value treatment y and less than said value for treatment x. Gray nodes are data points that are not statistically significant for both x and y-axes. Triangular shapes indicate values which exceed the viewing area for the graph. Size change reflects the magnitude of LFC values (i.e., larger shapes reflect larger LFC values). Vertical and horizontal dashed lines indicate user-defined LFC values.
Example S11: Highlighting data points

Overview

For point-based plots, users can highlight IDs of interest (i.e. genes, transcripts, etc.). Currently, this functionality is implemented in the following functions:

- vsScatterPlot()
- vsMAPlot()
- vsVolcano()
- vsFourWay()

To use this feature, simply provide a vector of specified IDs to the `highlight` parameter found in the prior functions. An example of a typical vector would be as follows:

```r
important_ids <- c(
  "ID_001",
  "ID_002",
  "ID_003",
  "ID_004",
  "ID_005"
)
important_ids
## [1] "ID_001" "ID_002" "ID_003" "ID_004" "ID_005"
```

For specific examples using the toy data set, please see the proceeding 4 sub-sections.
Highlighting with `vsScatterPlot()`

```r
data("df.cuff")
hl <- c("XLOC_000033",
        "XLOC_000099",
        "XLOC_001414",
        "XLOC_001409"
)
vsScatterPlot(
    x = "hESC", y = "iPS", data = df.cuff, d.factor = NULL,
    type = "cuffdiff", title = TRUE, grid = TRUE, highlight = hl
)
```

![Plot](image)

**Figure 37: Highlighting with `vsScatterPlot()`**

IDs of interest can be identified within basic scatter plots. When highlighted, non-important points will turn grey while highlighted points will turn blue. Text tags will try to optimize their location within the graph without trying to overlap each other.
Highlighting with `vsMAPlot()`

```r
hl <- c(
  "FBgn00022201",
  "FBgn0003042",
  "FBgn0031957",
  "FBgn0033853",
  "FBgn0003371"
)

vsMAPlot(
  x = "treated_paired.end", y = "untreated_paired.end",
  data = df.deseq, d.factor = "condition", type = "deseq",
  padj = 0.05, y.lim = NULL, lfc = NULL, title = TRUE,
  legend = TRUE, grid = TRUE, data.return = FALSE, highlight = hl
)
```

Figure 38: Highlighting with `vsMAPlot()`

IDs of interest can be identified within MA plots. When highlighted, non-important points will decrease in transparency (i.e. lower alpha values) while highlighted points will turn red. Text tags will *try* to optimize their location within the graph without trying to overlap each other.
Highlighting with *vsVolcano()*

```r
hl <- c(
    "FBgn0036248",
    "FBgn0026573",
    "FBgn0259742",
    "FBgn0038961",
    "FBgn0038928"
)
vsVolcano(
    x = "treated_paired.end", y = "untreated_paired.end",
    data = df.deseq, d.factor = "condition",
    type = "deseq", padj = 0.05, x.lim = NULL, lfc = NULL,
    title = TRUE, grid = TRUE, data.return = FALSE, highlight = hl
)
```

![Volcano plot example](image)

**Figure 39: Highlighting with `vsVolcano()`**

IDs of interest can be identified within volcano plots. When highlighted, non-important points will decrease in transparency (i.e. lower alpha values) while highlighted points will turn red. Text tags will *try* to optimize their location within the graph without trying to overlap each other.
Highlighting with `vsFourWay()`

```r
data("df.edger")
hl <- c("ID_639", "ID_518", "ID_602", "ID_449", "ID_076")
vsFourWay(
x = "WM", y = "WW", control = "MM", data = df.edger,
d.factor = NULL, type = "edger", padj = 0.05, x.lim = NULL,
y.lim = NULL, lfc = 2, title = TRUE, grid = TRUE,
data.return = FALSE, highlight = hl
)
```

**Figure 40: Highlighting with `vsFourWay()`**

IDs of interest can be identified within four-way plots. When highlighted, non-important points will decrease in transparency (i.e. lower alpha values) while highlighted points will turn dark grey. Text tags will try* to optimize their location within the graph without trying to overlap each other.
Example S12: Extracting datasets from plots

Overview

For all plots, users can extract datasets used for the visualizations. You may want to pursue this option if you want to use a highly customized plot script or you would like to perform some unmentioned analysis, for example.

To use this this feature, set the `data.return` parameter in the function you are using to `TRUE`. You will also need to assign the function to an object. See the following example for further details.

The data extraction process

In this example, we will use the toy data set `df.cuff`, a cuffdiff output on the function `vsScatterPlot()`. Take note that we are assigning the function to an object `tmp`:

```r
# Extract data frame from visualization
data("df.cuff")
tmp <- vsScatterPlot(
  x = "hESC", y = "iPS", data = df.cuff, d.factor = NULL,
  type = "cuffdiff", title = TRUE, grid = TRUE, data.return = TRUE
)
```

The object we have created is a list with two elements: `data` and `plot`. To extract the data, we can call the first element of the list using the subset method (`<object>[[1]]`) or by invoking its element name (`<object>$data`):

```r
df_scatter <- tmp[[1]] ## or use tmp$data
head(df_scatter)
# df_scatter:
# # id  x      y
# # 1 XLOC_000001 3.474e-01 20.218
# # 2 XLOC_000002 0.000e+00 0.000
# # 3 XLOC_000003 0.000e+00 0.000
# # 4 XLOC_000004 6.973e+05 0.000
# # 5 XLOC_000005 6.967e+02 355.823
# # 6 XLOC_000006 0.000e+00 1.514
```

Return the plot

By assigning each of these functions to a list, we can also store the plot as another element. To extract the plot, we can call the second element of the list using the aformentioned procedures:

```r
my_plot <- tmp[[2]] ## or use tmp$plot
my_plot
```
iPS vs. hESC

\[ \log_{10}(\text{FPM}) - \text{hESC} \]

\[ \log_{10}(\text{FPM}) - \text{iPS} \]
Example S13: Changing text sizes

Overview

For all functions, users can modify the font size of multiple portions of the plot. These portions primarily revolve around these components:

- Axis text and titles
- Plot title
- Legend text and titles
- Facet titles

To manipulate these components, users can modify the default values of the following parameters:

- `xaxis.text.size`
- `yaxis.text.size`
- `xaxis.title.size`
- `yaxis.title.size`
- `main.title.size`
- `legend.text.size`
- `legend.title.size`
- `facet.title.size`

What exactly can you manipulate?

Each of parameters mentioned in the prior section refer to numerical values. These values correlate to font size in typographic points. To illustrate what exactly these parameters modify, please refer to the following figure:

*Figure 41: A visual guide to text size parameters*

Users can modify these components which are highlighted by their respective parameter.
The `facet.title.size` parameter refers to the facets which are allocated in the matrix functions (`vsScatterMatrix()`, `vsMAMatrix()`, `vsVolcanoMatrix()`). This is illustrated in the following figure:

**Figure 42: Location of facet titles**
Facet title sizes can be modified using the `facet.title.size` parameter.

Since not all functions are equal in their parameters and component layout, some functions will either have or lack some of the prior parameters. To get an idea of which have functions have which, please refer to the following figure:

**Figure 43: An overview of text size parameters for each function**
Cells highlighted in red refer to parameters (columns) which are found in their respective functions (rows). Cells which are grey indicate parameters which are not found in each of the functions.
Method S1: Determining data point shape and size changes

The shape and size of each data point will also change depending on several conditions. To maximize the viewing area while retaining high resolution, some data points will not be present within the viewing area. If they exceed the viewing area, they will change shape from a circle to a triangular orientation.

The extent (i.e. fold change) to how far these points exceed the viewing area are based on the following criteria:

- **SUB** - values that fall within the viewing area of the plot.
- **T-1** - values that are greater than the maximum viewing area and are less than the 25th percentile of values that exceed the viewing area.
- **T-2** - Similar to **T-1**; values fall between the 25th and 50th percentile.
- **T-3** - Similar to **T-2**; values fall between the 50th and 75th percentile.
- **T-4** - Similar to **T-3**; values fall between the 75th and 100th percentile.

To further clarify these conditions, please refer to the following figure:

![Figure 44: An illustration detailing the principles behind the node size for the differential gene expression functions](image)

In this figure, the data points increase in size depending on which quartile they reside as the absolute LFC increases (top bar). Data points that fall within the viewing area classified as SUB while data points that exceed this area are classified as T-1 through T-4.
Method S2: Determining function performance

Function efficiencies were determined by calculating system times by using the \texttt{microbenchmark} R package. Each function was ran 100 times with the prior code used in the documentation. All benchmarks were determined on a machine running a 64-bit Windows 10 operating system, 8 GB of RAM, and an Intel Core i5-6400 processor running at 2.7 GHz.

Scatterplots

![Scatterplot Diagram](image)

\textbf{Figure 45: Benchmarks for the `vsScatterPlot()` function}
Time (ms) distributions were generated for this function using 100 trials for each of the three RNAseq data objects. Cuffdiff, DESeq2, and edgeR example data sets contained 1200, 724, and 29391 transcripts, respectively.
Figure 46: Benchmarks for the `vsScatterMatrix()` function

Time (ms) distributions were generated for this function using 100 trials for each of the three RNAseq data objects. Cuffdiff, DESeq2, and edgeR example data sets contained 1200, 724, and 29391 transcripts, respectively.
Box plots

Figure 47: Benchmarks for the `vsBoxPlot()` function
Time (ms) distributions were generated for this function using 100 trials for each of the three RNAseq data objects. Cuffdiff, DESeq2, and edgeR example data sets contained 1200, 724, and 29391 transcripts, respectively.
**Differential gene expression matrices**

*Figure 48: Benchmarks for the 'vsDEGMatrix()' function*

Time (ms) distributions were generated for this function using 100 trials for each of the three RNAseq data objects. Cuffdiff, DESeq2, and edgeR example data sets contained 1200, 724, and 29391 transcripts, respectively.
Volcano plots

Figure 49: Benchmarks for the 'vsVolcano()' function
Time (ms) distributions were generated for this function using 100 trials for each of the three RNAseq data objects. Cuffdiff, DESeq2, and edgeR example data sets contained 1200, 724, and 29391 transcripts, respectively.
Volcano plot matrices

Figure 50: Benchmarks for the `vsVolcanoMatrix()` function
Time (ms) distributions were generated for this function using 100 trials for each of the three RNAseq data objects. Cuffdiff, DESeq2, and edgeR example data sets contained 1200, 724, and 29391 transcripts, respectively.
Figure 51: Benchmarks for the 'vsMAPlot()' function

Time (ms) distributions were generated for this function using 100 trials for each of the three RNAseq data objects. Cuffdiff, DESeq2, and edgeR example data sets contained 1200, 724, and 29391 transcripts, respectively.
**MA matrices**

![vsMAMatrix()](image)

**Figure 52: Benchmarks for the 'vsMAMatrix()' function**

Time (s) distributions were generated for this function using 100 trials for each of the three RNAseq data objects. Cuffdiff, DESeq2, and edgeR example data sets contained 1200, 724, and 29391 transcripts, respectively.
Four way plots

**Figure 53: Benchmarks for the `vsFourWay()` function**
Time (ms) distributions were generated for this function using 100 trials for each of the three RNAseq data objects. Cuffdiff, DESeq2, and edgeR example data sets contained 1200, 724, and 29391 transcripts, respectively.
Session info

```r
# R version 3.5.0 (2018-04-23)
# Platform: x86_64-w64-mingw32/x64 (64-bit)
# Running under: Windows 10 x64 (build 17134)
#
# locale:
# [1] LC_COLLATE=English_United States.1252
# [2] LC_CTYPE=English_United States.1252
# [3] LC_MONETARY=English_United States.1252
# [4] LC_NUMERIC=C
# [5] LC_TIME=English_United States.1252
#
# attached base packages:
# [1] parallel stats4 stats graphics grDevices datasets utils
# [8] methods base
#
# other attached packages:
# [1] edgeR_3.22.3
# [3] DESeq2_1.20.0
# [5] DelayedArray_0.6.1
# [7] matrixStats_0.53.1
# [9] GenomicRanges_1.32.3
# [13] BiocGenerics_0.26.0
# [15] BiocStyle_2.8.2
#
# loaded via a namespace (and not attached):
# [1] bitops_1.0-6
devtools_1.13.6
rprojroot_1.3-2
# [7] backports_1.1.2
R6_2.2.2
rpart_4.1-13
# [10] Hmisc_4.1-1
DBI_1.0.0
lazyeval_0.2.1
# [13] colorspace_1.3-2
nnet_7.3-12
withr_2.1.2
# [16] tidyselect_0.2.4
gridExtra_2.3
GGally_1.4.0
# [19] bit_1.1-14
compiler_3.5.0
htmlTable_1.12
# [22] xml2_1.2.0
labeling_0.3
bookdown_0.7
# [25] scales_0.5.0
checkmate_1.8.5
genefilter_1.62.0
# [28] commonmark_1.5
stringr_1.3.1
digest_0.6.15
# [31] foreign_0.8-70
rmarkdown_1.10
XVector_0.20.0
# [34] base64enc_0.1-3
htmltools_0.3.6
htmlwidgets_1.2
# [37] rlang_0.2.1
rsudioapi_0.7
RSQLite_2.1.1
# [40] acepack_1.4.1
RCurl_1.95-4.10
magrittr_1.5
# [43] GenomeInfoDbData_1.1.0
Formula_1.2-3
Matrix_1.2-14
# [46] Rcpp_0.12.17
munsell_0.5.0
stringi_1.2.3
# [49] yaml_2.1.19
zlibbioc_1.26.0
plyr_1.8.4
# [52] grid_3.5.0
blob_1.1.1
ggrepel_0.8.0
# [55] fortunes_1.5-4
lattice_0.20-35
splines_3.5.0
# [58] annotate_1.58.0
locfit_1.5-9.1
knitr_1.20
```
## [61] pillar_1.2.3  reshape2_1.4.3  geneplotter_1.58.0
## [64] XML_3.98-1.11  glue_1.2.0  evaluate_0.10.1
## [67] latticeExtra_0.6-28  data.table_1.11.4  png_0.1-7
## [70] testthat_2.0.0  gtable_0.2.0  purrr_0.2.5
## [73] tidyr_0.8.1  reshape_0.8.7  ggplot2_2.2.1
## [76] xfun_0.2  xtable_1.8-2  roxygen2_6.0.1
## [79] survival_2.42-3  tibble_1.4.2  tinytex_0.5
## [82] AnnotationDbi_1.42.1  memoise_1.1.0  cluster_2.0.7-1