interactiveDisplay: A package for enabling interactive visualization of Bioconductor objects

Shawn Balcome

October 24, 2023
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Figures</td>
<td>2</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>2</td>
</tr>
<tr>
<td>2 Installation</td>
<td>2</td>
</tr>
<tr>
<td>3 Support</td>
<td>3</td>
</tr>
<tr>
<td>4 Citation</td>
<td>3</td>
</tr>
<tr>
<td>5 Bioconductor Object Methods</td>
<td>4</td>
</tr>
<tr>
<td>5.1 GRanges and GRangesList</td>
<td>4</td>
</tr>
<tr>
<td>5.1.1 Object Background - GRanges</td>
<td>4</td>
</tr>
<tr>
<td>5.1.2 Object Background - GRangesList</td>
<td>5</td>
</tr>
<tr>
<td>5.1.3 Method</td>
<td>6</td>
</tr>
<tr>
<td>5.1.4 UI</td>
<td>6</td>
</tr>
<tr>
<td>5.1.5 Plots</td>
<td>8</td>
</tr>
<tr>
<td>5.1.6 Metadata Tabset</td>
<td>10</td>
</tr>
<tr>
<td>5.2 ExpressionSet</td>
<td>10</td>
</tr>
<tr>
<td>5.2.1 Object Background</td>
<td>10</td>
</tr>
<tr>
<td>5.2.2 Method</td>
<td>11</td>
</tr>
<tr>
<td>5.2.3 UI</td>
<td>11</td>
</tr>
<tr>
<td>5.2.4 Plots</td>
<td>14</td>
</tr>
<tr>
<td>5.2.5 GO Tabset</td>
<td>17</td>
</tr>
<tr>
<td>5.3 RangedSummarizedExperiment</td>
<td>18</td>
</tr>
<tr>
<td>5.3.1 Object Background</td>
<td>18</td>
</tr>
<tr>
<td>5.3.2 Method</td>
<td>19</td>
</tr>
<tr>
<td>5.3.3 UI</td>
<td>19</td>
</tr>
<tr>
<td>5.3.4 Plot</td>
<td>20</td>
</tr>
<tr>
<td>6 Additional Functions/Methods</td>
<td>20</td>
</tr>
<tr>
<td>6.1 Dataframe - display()</td>
<td>20</td>
</tr>
<tr>
<td>6.2 Dataframe - simplenet()</td>
<td>21</td>
</tr>
<tr>
<td>6.3 gridsvgs()</td>
<td>21</td>
</tr>
<tr>
<td>7 Special Mention Components</td>
<td>21</td>
</tr>
<tr>
<td>7.1 Shiny</td>
<td>22</td>
</tr>
<tr>
<td>7.2 Gviz/ggbio</td>
<td>22</td>
</tr>
<tr>
<td>7.3 gridSVG</td>
<td>22</td>
</tr>
<tr>
<td>8 JavaScript Libraries</td>
<td>22</td>
</tr>
<tr>
<td>8.1 Data-Driven Documents</td>
<td>22</td>
</tr>
<tr>
<td>8.2 Zoom/Pan JavaScript libraries</td>
<td>22</td>
</tr>
<tr>
<td>8.3 JavaScript Color Chooser</td>
<td>23</td>
</tr>
<tr>
<td>9 Acknowledgments</td>
<td>23</td>
</tr>
<tr>
<td>10 SessionInfo</td>
<td>23</td>
</tr>
<tr>
<td>11 References</td>
<td>25</td>
</tr>
</tbody>
</table>
interactiveDisplay: A package for enabling interactive visualization of Bioconductor objects

1 Introduction

interactiveDisplay makes use of Bioconductor visualization packages, the Shiny [8] web framework, the D3.js visualization JavaScript library, and other libraries to produce various web applications built around Bioconductor objects.

Four popular Bioconductor data objects are currently supported: \texttt{GRanges}, \texttt{GRangesList}, \texttt{ExpressionSet} and \texttt{RangedSummarizedExperiment}. In addition, the second April 2014 release version now includes a method for data frames and is imported by the \texttt{AnnotationHub} [1] package.

This vignette will provide some background and guidance on the use of the current supported methods.

2 Installation

The webpage for the release version of interactiveDisplay is available at:
interactiveDisplay: A package for enabling interactive visualization of Bioconductor objects


And the development version is available at:

It can also be installed within the R console using BiocManager::install().

```r
if (!requireNamespace("BiocManager", quietly=TRUE))
  install.packages("BiocManager")
BiocManager::install("interactiveDisplay")
```

interactiveDisplay is actively maintained and it may be desirable to use the developer branch of the package. If the user is willing to install development versions of Bioconductor packages, this can be accomplished by toggling the setting and reinstalling packages with BiocManager::install(version = "devel"). The development branch is actively changed, so users should expect to encounter errors. Generally, development and release versions of packages should not be mixed.

```r
if (!requireNamespace("BiocManager", quietly=TRUE))
  install.packages("BiocManager")
BiocManager::install(version = "devel")
BiocManager::install("interactiveDisplay")
```

Because interactiveDisplay depends on many different R packages, some of which are used in only one individual method, many dependencies are suggested and will only be required when a method is run for the first time. This helps with load time when other packages (currently AnnotationHub) import methods from interactiveDisplay. If a required package is missing when a method is run, a message with instructions on obtaining it will be output in the R console.

3 Support

- Send me an email directly at balc0022@umn.edu
- Search, subscribe or contact the bioconductor or bioc-devel mailing lists at http://www.bioconductor.org/help/mailing-list/

4 Citation

```r
citation("interactiveDisplay")
```

```
## To cite package 'interactiveDisplay' in publications use:
##
## @Manual{,
##   title = {interactiveDisplay: Package for enabling powerful shiny web displays of Bioconductor objects},
##   author = {Shawn Balcome},
##   year = {2023},
##   note = {BibTeX entry for LaTeX users is}
##   @Manual{,
##     title = {interactiveDisplay: Package for enabling powerful shiny web displays of Bioconductor objects},
##     author = {Shawn Balcome},
##     year = {2023},
##   }
## }`
```
5 Bioconductor Object Methods

5.1 GRanges and GRangesList

5.1.1 Object Background - GRanges

The GRanges object is a standardized container for genomic location data used in many Bioconductor packages. It is built on S4 classes from the infrastructure package IRanges.

First, load the libraries needed for the examples in this document and load the example GRanges data provided with the interactiveDisplay package.

```r
options(width=80)
options(continue=" ")
suppressMessages(library(ggplot2))
suppressMessages(library(interactiveDisplay))
suppressMessages(library(Biobase))
data(mmgr)
```

```r
# Loading required package: GenomicRanges
# Loading required package: stats4
# Loading required package: S4Vectors
#
# Attaching package: 'S4Vectors'
# The following object is masked from 'package:utils':
# findMatches
# The following objects are masked from 'package:base':
# I, expand.grid, unname
# Loading required package: IRanges
# Loading required package: GenomeInfoDb
```

```r
# GRanges object with 1000 ranges and 2 metadata columns:
#
## seqnames ranges strand | tx_id tx_name
## <Rle> <IRanges> <Rle> | <character> <character>
## [1] chr5 53267178-53278539 + | 27296 ENSMUST00000147148
## [2] chr2 170346991-170427845 - | 40171 ENSMUST00000172093
## [3] chr7 4472475-4474584  + | 28623 ENSMUST00000084338
## [5] chr7 127588698-127603605 + | 28623 ENSMUST0000038824
## ... ... ... ... ... ... ...
## [996] chr5 129715558-129718290 + | 24633 ENSMUST00000148260
## [997] chr4 115915120-115939926 - | 27296 ENSMUST00000147148
```
## 

Each row in the `GRanges` object represents an individual range using the IRanges Bioconductor object to store the first and last position on the sequence. The chromosome and strand designation are both stored using run-length encoding (rle) for efficiency. The sequence lengths of are also stored in the `GRanges`. Any columns to the right of the pipe symbols in the object are metadata columns. Practically any R data object can be stored in the metadata, it is left up to the user and is often dependent on the experimental context of the data that is being conformed to the `GRanges` standard. For instance, to represent interactions between ranges, it is possible to store a `GRanges` object within another `GRanges` object's metadata. In the example here, transcript information has been inserted in the metadata.

### 5.1.2 Object Background - `GRangesList`

The `GRangesList` class is a container for storing a list of `GRanges` objects, the class is instanciated with constructor function `GRangesList()`.

```r
grl <- GRangesList(list(mmgr, mmgr))
```

`GRangesList` objects are useful for compound objects, an example given in the documentation being spliced transcripts that are are made up of exons [4]

```r
data(mmgrl)
```

```
## GRangesList object of length 3:
## $gr1
## GRanges object with 189 ranges and 2 metadata columns:
## seqnames ranges strand | tx_id tx_name
## <Rle> <IRanges> <Rle> | <character> <character>
## [1] chr1 67585053-67587036 + | 14535 ENSMUST00000038824
## [2] chr1 86545392-86582547 - | 14526 ENSMUST00000147148
## [3] chr1 87264363-87310836 + | 27296 ENSMUST00000144433
## [4] chr1 88318913-88388851 + | 37251 ENSMUST00000109152
## [5] chr1 191894074-191907475 - | 14535 ENSMUST00000038824
## ... ... ... ... ... ...
## [185] chr3 35918998-35930119 - | 14526 ENSMUST00000109152
## [186] chr3 30906985-30969405 - | 14535 ENSMUST00000167068
## [187] chr3 121155865-121171694 - | 52581 ENSMUST00000147148
## [188] chr3 29952782-29997702 - | 24633 ENSMUST00000033088
## [189] chr3 67430096-67476529 + | 28623 ENSMUST00000084338
## -------
## seqinfo: 6 sequences from an unspecified genome
## $gr2
## GRanges object with 205 ranges and 2 metadata columns:
## seqnames ranges strand | tx_id tx_name
## <Rle> <IRanges> <Rle> | <character> <character>
## [1] chr1 67585053-67587036 + | 14535 ENSMUST00000038824
## [2] chr1 86545392-86582547 - | 14526 ENSMUST00000147148
## [3] chr1 87264363-87310836 + | 27296 ENSMUST00000144433
## [4] chr1 88318913-88388851 + | 37251 ENSMUST00000109152
```
5.1.3 Method

The `display()` function in `interactiveDisplay` will start a Shiny web application tailored to the particular supported object passed as an argument. Both the `GRanges` and `GRangesList` data objects can be visualized as follows:

```r
display(mmgr)

display(mmgrl)
```

It is also important to note, the GRanges and GRangesList methods are designed to send a subset of the submitted object back to the console as a result when the GUI application is exited. If the user wants to store this new object, simply assign it as illustrated in the following examples.

```r
new_mmgr <- display(mmgr)

new_mmgrl <- display(mmgrl)
```

While simple in concept, data cleaning and wrangling are universal to any workflow, so this is a reasonable first implementation of Shiny’s capability of returning results back to R.

5.1.4 UI

Unlike other visualization packages that are run in the console and lend themselves to walkthroughs and examples using the knitr package, `interactiveDisplay`’s reliance on Shiny moves many of its functional components outside of the R console. This section will be a walkthrough of the user interface controls and features of the applications produced from the `GRanges` and `GRangesList` methods. Besides a couple small exceptions, these two methods are functionally similar and their descriptions have been consolidated together.
These dropdown UI elements are used to select a particular \textit{GRanges} and ranges associated with a chosen chromosome (The \textit{GRanges} dropdown is not present in the \textit{GRanges} method). These ranges will be shown in the \textit{Interactive Plot} and the subsetting operations in the other UI elements will apply to these active ranges.

The \textit{UCSC Genome} dropdown gives the user choices for the ideogram image shown above the trackplots. This selection is entirely determined by the user and the particular data in the GRange/GRangesList may not correspond to any of the available choices. This feature also depends on access to remote UCSC servers. Because the correct ideogram may not be available, it can be suppressed via the \textit{Suppress Ideogram} checkbox element.

The \textit{Plot Window} and \textit{Range Length Filter} sliders and \textit{Choose a Strand} dropdown provide the user with options for subsetting the original data to only the current ranges in view. The \textit{Plot Window} is reflected by the highlighted region on the ideogram in addition to the trackplot below it. The \textit{Range Length Filter} can be used to filter ranges shown by their respective lengths. The \textit{Choose a Strand} dropdown gives the user a choice of displaying ranges located on positive, negative, or both strands.
Besides subsetting which ranges are in the current view, the current subset of ranges can be saved. This is a two step process. Ranges of interest are first deposited, allowing the user to select different chromosomes/GRanges to manipulate and add to the list of deposited selections. Once the user is ready to return the data back to the console, **Save to Console** stops the Shiny session and returns the subsetted GRanges or GRangesList object. **Clear Deposit** resets all selections. The usefulness of this feature is left to the user's individual needs and is also very dependent on the data submitted. Upon inspection, ranges of interest may fall on the same chromosome or chromosome region, share specific metadata tags, or individual lengths. The Shiny UI offers an alternative to filtering away unwanted ranges iteratively in the console with multiple subsetting commands (and repeatedly printing output for validation), while providing instant visual feedback.

### 5.1.5 Plots

![Static Circle Layout](image)

**Figure 5: Static Circle Layout**
The *Static Circle Layout* is a ggbio [11] based plot is provided for overview purposes and is not intended to represent the user's current subsetting operations. It provides the user with a broad summary of the data they submitted in its unaltered form.

The *Interactive Plot* is the main view for these two methods. Trackplots are drawn with the Bioconductor package Gviz [3].

Using Shiny’s advanced datatables, this is a searchable, sortable table of all ranges contained in the current *GRanges*. In the case of the *GRanges* method, this table contains all ranges in the object, and in the *GRangesList* method the table contains ranges in the current active *GRanges* selected from the dropdown.
This table is similar to the previous view, except it reflects the current state of subsetting operations performed by the UI and what is currently displayed in the trackplot.

![Figure 9: Deposited Selections](image)

This table view doesn't show individual ranges, instead it gives a summary of selected ranges that have been locked in with the Deposit Ranges in View button. Here the user can keep track of sets of ranges of interest that can be sent back to the console when the application is closed.

### 5.1.6 Metadata Tabset

The lower tabset consists of individual tabs that are dynamically produced based on the metadata contents of the object submitted. Each tab has a list of checkboxes that allow the user to subset the selection based on the metadata. The subsetting is applied across all deposited selections. The usefulness of this panel really depends on the content of the object submitted. Objects containing no metadata columns obviously have no use for this UI, but metadata rich objects with categorical, biologically relevant labels can potentially be subset quickly according to them.

### 5.2 ExpressionSet

#### 5.2.1 Object Background

The *ExpressionSet* class is a standardized container for gene expression data, specifically from a microarray.

```r
data(expr)
expr
## ExpressionSet (storageMode: lockedEnvironment)
## assayData: 500 features, 26 samples
```
We can view the expression values with the exprs() function.

```r
exprs(expr)[1:10,1:7]
```

The structure of the `ExpressionSet` object:

- expression data (`assayData`)
- sample metadata (`phenoData`)
- annotations and instrument metadata (`featureData`, `annotation`)
- sample protocol (`protocolData`)
- experimental design (`experimentData`)

### 5.2.2 Method

```r
display(expr)
```

### 5.2.3 UI

The method for `ExpressionSet` has the most features of any of the current objects handled by `interactiveDisplay`. The main panel consists of three tabs that visualize the data as a heatmap, network and dendrogram. Gene ontology summaries of probes or selected probe clusters are available in tables in the lower tabset of the main panel.
The Experiment Info table at the top of the sidebar is entirely dependent on whether the *ExpressionSet* object submitted has these fields filled out.

Depending on whether the parameter `sflag=FALSE` is passed to the `display()` method and the selected dimensions of the *ExpressionSet*, replotting the heatmap can introduce delays. Using the checkbox to suppress the heatmap plotting can circumvent this. Also, depending on the relative dimensions of probes/samples, transposing the plot can make better use of the space available.

This UI element simply toggles between displaying plots that represent distances/clustering across probes or samples.

This set of UI elements can be used to provide a gene ontology summary of a chosen cluster of probes in the network view. Simply mouseover a node within a cluster of interest and the cluster number will pop-up. Input that cluster number into the input field and hit the *View/Update GO Summary* button. This can take 10-30 seconds, which is why this is not automatically reactive. Adjusting the Minimum probe population for GO term allows the user to explore general or specific top ranking terms. The summary table is located in the second tab of the lower tabset.
This drop-down contains options from the current subset of probes being visualized. Selecting a probe refreshes the first tab of the lower tabset with a GO summary for that probe, as well as associated gene names and Entrez IDs. When toggled to probe view, clicking on a probe node in the network view also refreshes this selection and may be more convenient.

This UI set is used for subsetting the submitted ExpressionSet object. The initial values are set to the maximum number of samples and to a very conservative 20 probes. The user is free to expand out to the full dimensions.

Depending on the dimensions of the ExpressionSet object submitted/subset, it may be necessary to adjust label sizes to prevent label overlapping or illegibly small text.

This UI element is used in the network view, allowing the user to determine the number of edges and the distance threshold represented in the graph.
This is a cosmetic control for the force directed graph, which allows the user to adjust how nodes spread themselves out and the length of their edges.

These controls set the parameters for the distance and hierarchical clustering methods used. Only base R methods are currently implemented but more may be included in future releases as well as possible option of allowing the user to manually provide their own via a textbox element.

These elements affect the coloration of the heatmap. The color picker for the tricolor scale is not provided by the Shiny framework. This uses an additional JavaScript library which will be discussed in a later section. This is not just a cosmetic option, different modes can better visualize different data and accommodate color blind users.

This stops the application. The ExpressionSet method does not currently return any data back to the R console.
Many packages on CRAN and Bioconductor have specialized heatmap functions, HeatPlus [6] has a specific function for ExpressionSet objects. Given that one of the goals of this package is modularity and reuse of code, the more generic plotting package ggplot2 [10] was used instead of other options. Often heatmap and dendrogram plots are paired together but given that there is a great deal of screen real estate taken up by tables, UI controls and on-screen documentation, the dendrograms are moved to a separate tab.
The **Network View** uses a D3.js based force directed graph to visualize distance/similarity between samples or probes based on user selected distance methods and threshold settings. Nodes in the graph are colored based on cluster settings. As stated previously, when the view is toggled to display probes, there is the added interface functionality of being to use the graph to conveniently gather GO information. When used in conjunction with the heatmap view, the network graph can quickly convey to the user which samples have similar profiles or which probes have similar expression across samples.
The decision to separate the dendrogram plots from the heatmap was a difficult one, paired heatmap/dendrogram plots quickly convey clustering groupings alongside the data that they were generated from. However, due to the forementioned design/layout constraints, dendrograms are placed in their own tab. This keeps the already complex UI as clean as possible with adequate space for all plots. To accommodate this decision some care was taken to keep some consistency between views. The clustering colors are consistent across the three views and the sample/probe order is the same between the heatmap and dendrogram views. This consistency helps the user quickly switch between tabs and stay oriented with the data they are examining.

5.2.5 GO Tabset

These summary tables provide Entrez ID and gene names for the selected probe and the top ranking results for GO descriptors. This table allows the user to manually choose and characterize probes that could be exhibiting differential expression in the submitted espressionSet.
This tab provides a table of the results of the hyperg() function from the GOstats package [2]. Ranked in order of p-value, the results give a general GO summary across the group of selected probes. Much like the previous table, this provides an interactive tool for characterizing differentially expressed sets of probes.

5.3 **RangedSummarizedExperiment**

5.3.1 Object Background

data(se)

```r
## Loading required package: SummarizedExperiment
## Loading required package: MatrixGenerics
## Loading required package: matrixStats
```

```
## Attaching package: 'matrixStats'
## The following objects are masked from 'package:Biobase':
## anyMissing, rowMedians
## Attaching package: 'MatrixGenerics'
## The following objects are masked from 'package:matrixStats':
## colAlls, colAnyNAs, colAnyNAs, colAvgPerRowSet, colCollapse,
## colCounts, colCummins, colCummzs, colCumprods, colCumsums,
## colDiffs, colIQRDiffs, colIQRs, colLogSumExp, colMADiffs,
## colMads, colMams, colMeans2, colMedians, colMins, colOrderStats,
```
The `RangedSummarizedExperiment` class is similar to a dataframe where rows represent ranges (using `GRanges/GRangesList`) and columns represent samples (with sample data summarized as a dataframe). It can contain one or more assays. Ranges have read counts associated with them [4].

### 5.3.2 Method

```r
display(se)
```

### 5.3.3 UI

![Summarized Experiment UI](image)

**Figure 30:** `RangedSummarizedExperiment` UI
The UI for the `RangedSummarizedExperiment` method is relatively simple. It is the best candidate of the existing methods for future development. The `RangedSummarizedExperiment` class is widely used, data rich object and has a lot of potential for improvement.

### 5.3.4 Plot

![Binned Mean Counts by Position](image)

**Figure 31: Binned Mean Counts by Position**

Currently the plot gives a simple depiction of expression score/counts across regions of the selected chromosome. However, this plot currently obfuscates the relative coverage of the ranges of interest. It will likely need to be replaced with a multiple track plot in future releases.

### 6 Additional Functions/Methods

#### 6.1 Dataframe - display()

```r
display(mtcars)
```

Similar to the GRanges and GRangesList methods, the dataframe method can send a subset of the submitted object back to the console and can be stored in a new variable.

```r
new_mtcars <- display(mtcars)
```

![Dataframe Table](image)

**Figure 32: Dataframe Table**

Despite the dataframe being a base R object and not exclusive to use with Bioconductor software, it’s still useful to write methods for base R objects with genomic workflows in mind. The Bioconductor package Annotation-
Hub imports this dataframe method. This method makes use of Shiny's new advanced datatables with a slight modification to return selected rows back to the console.

6.2 Dataframe - simplenet()

`simplenet(mtcars)`

![Figure 33: simplenet](image)

Code to produce the force network tab from the ExpressionSet tab was recently converted to a stand-alone method for numeric dataframes in the developer branch of interactiveDisplay.

6.3 gridsvgjs()

One of the goals of this package is to provide users and future contributors with simple, modular functions that can be inserted into more complex methods or serve as the foundation or inspiration for future components. This motivation was behind the decision to export the helper function gridsvgjs(). This function takes any grid based plot, converts it to SVG format (see section on gridSVG), inserts JavaScript for the Pan/Zoom functionality (see JavaScript Libraries section) and initiates a local Shiny instance. The four display() methods make use of the majority of the code in gridsvgjs(). After considering the potential usefulness of doing this with any plot, I repackaged the code as a stand-alone function. The standard R graphics viewer can sometimes be restrictive, particularly for larger plots. Viewing a plot as a vector based rendering with pan and zoom capabilities seemed to be a desirable addition to the package. The inserted JavaScript functionality is relatively simple, but hopefully it will encourage more interactivity in the future.

```r
data(mtcars)
qp <- qplot(mpg, data=mtcars, geom="density", fill=factor(cyl), alpha=I(.4))
plot(qp)
```

To view the plot in the browser with JavaScript Pan/Zoom capabilities:

```r
gridsvgjs(qp)
```

7 Special Mention Components
7.1 Shiny

The Shiny web framework is the component that was the main inspiration behind the proposal for this package. The relative simplicity of prototyping applications and its use of existing universal web standards makes it an enticing piece of software to build an open source package around. It means, unlike some packages built on low level code, there is a possibility for other biologists to contribute to the open source development. Shiny provides the web service and predefined UI elements, but relies on existing R packages to provide the data analysis and plotting. The interactivity of Shiny apps depends on its reactive programming model.

http://shiny.rstudio.com/

7.2 Gviz/ggbio

Gviz and ggbio provide the biology specific plots in the GRanges and GRangesList methods in interactiveDisplay. Both offer track-based plotting methods for genomic data. Gviz was the initial choice for GRanges/GRangesList trackplots because it offered more generic, extensible plotting. Later in development, ggbio plotting was also added as it handles additional complex plotting methods.

7.3 gridSVG

Simon Potter's gridSVG [5] is an important asset in developing interactive R plots. His package converts a grid [7] based plot to SVG format directly. Two major plotting packages for R, lattice [9] and ggplot2 are built around grid (and many other packages depend on ggplot2), which gives gridSVG a lot of potential interaction with many Bioconductor libraries.

Figure 34: gridSVG [5]

8 JavaScript Libraries

8.1 Data-Driven Documents

Michael Bostock's JavaScript library, D3.js (http://d3.js.org/), allows for highly customizable data visualizations using universal web standards. It can work along with Shiny to bind data to the Document Object Model (DOM). Unlike Shiny which handles the interactive elements but leaves R to handle the visualization, D3.js can handle both. Currently only the ExpressionSet method's Network View makes use of this library.

8.2 Zoom/Pan JavaScript libraries

This Zoom/Pan library is used in all four main display() methods and in gridsvgjs(). While a relatively simple feature, hopefully it will lead to more complex JavaScript interactivity that directly manipulates plots produced by R plotting packages. The Andrea Leofreddi's original JavaScript code was expanded upon by John Krauss to produce the version of this library used in interactiveDisplay.
Andrea Leofreddi
https://code.google.com/p/svgpan/
John Krauss
https://github.com/talos/jquery-svgpan

### 8.3 JavaScript Color Chooser

The colorpicker included in the ExpressionSet method should eventually be moved into the Shiny package itself. Many R users outside of the Bioconductor community use RColorBrewer and could make use of this UI element in their Shiny-based applications.

Jan Odvarko
http://jscolor.com/

### 9 Acknowledgments

Shiny
Joe Cheng and Winston Chang
http://www.rstudio.com/shiny/

Force Layout
Jeff Allen
https://github.com/trestletech/shiny-sandbox/tree/master/grn

gridSVG
Simon Potter
http://sjp.co.nz/projects/gridsvg/

Zoom/Pan JavaScript libraries
John Krauss
https://github.com/talos/jquery-svgpan
Andrea Leofreddi
https://code.google.com/p/svgpan/

JavaScript Color Chooser
Jan Odvarko
http://jscolor.com/

Data-Driven Documents
Michael Bostock
http://d3js.org/

### 10 SessionInfo

```r
sessionInfo()
```

```text
## R version 4.3.1 (2023-06-16)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 22.04.3 LTS
##
## Matrix products: default
## BLAS: /home/biocbuild/bbs-3.18-bioc/R/lib/libRblas.so
## LAPACK: /usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.10.0
```
interactiveDisplay: A package for enabling interactive visualization of Bioconductor objects

## locale:
## [1] LC_CTYPE=en_US.UTF-8   LC_NUMERIC=C
## [3] LC_TIME=en_GB          LC_COLLATE=C
## [5] LC_MONETARY=en_US.UTF-8 LC_MESSAGES=en_US.UTF-8
## [7] LC_PAPER=en_US.UTF-8   LC_NAME=C
## [9] LC_ADDRESS=C           LC_TELEPHONE=C

## time zone: America/New_York
## tzcode source: system (glibc)

## attached base packages:
## [1] stats4     grid       stats       graphics  grDevices    utils    datasets
## [8] methods    base

## other attached packages:
## [1] SummarizedExperiment_1.32.0 MatrixGenerics_1.14.0
## [3] matrixStats_1.0.0   GenomicRanges_1.54.0
## [5] GenomeInfoDb_1.38.0 Ranges_2.36.0
## [7] S4Vectors_0.40.0    Biobase_2.62.0
## [9] interactiveDisplay_1.40.0 BiocGenerics_0.48.0
## [11] ggplot2_3.4.4       knitr_1.44

## loaded via a namespace (and not attached):
## [1] tidyselect_1.2.0     dplyr_1.1.3
## [3] blob_1.2.4           Biostrings_2.70.0
## [5] bitops_1.0-7         fastmap_1.1.1
## [7] RCurl_1.98-1.12      promises_1.2.1
## [9] XML_3.99-0.14        digest_0.6.33
## [11] mime_0.12            lifecycle_1.0.3
## [13] ellipsis_0.3.2       survival_3.5-7
## [15] KEGGREST_1.42.0      interactiveDisplayBase_1.40.0
## [17] RSQLite_2.3.1        magrittr_2.0.3
## [19] genefilter_1.84.0    compiler_4.3.1
## [21] rlang_1.1.1          tools_4.3.1
## [23] utf8_1.2.4           S4Vectors_1.2.0
## [25] bit_4.0.5            DelayedArray_0.28.0
## [27] plyr_1.8.9           RColorBrewer_1.1-3
## [29] abind_1.4-5          withr_2.5.1
## [31] fansi_1.0.5          xtable_1.8-4
## [33] colorspace_2.1-0     scales_1.2.1
## [35] cli_3.6.1            crayon_1.5.2
## [37] generics_0.1.3       httr_1.4.7
## [39] reshape2_1.4.4       gridSVG_1.7-5
## [41] DBI_1.1.3             cachem_1.0.8
## [43] stringr_1.5.0        zlibbioc_1.48.0
## [45] splines_4.3.1        AnnotationDbi_1.64.0
## [47] XVector_0.42.0       vctrs_0.6.4
## [49] Matrix_1.6-1.1       jsonlite_1.8.7
## [51] bit64_4.0.5          RBGL_1.78.0
## [53] GSEABase_1.64.0       annotate_1.80.0
## [55] glue_1.6.2           stringi_1.7.12
## [57] gtable_0.3.4         later_1.3.1
interactiveDisplay: A package for enabling interactive visualization of Bioconductor objects

## References


