SOFTWARE

bigPint: A Bioconductor package that makes big data pint-sized

Lindsay Rutter[1*] and Dianne Cook[2]

*Correspondence:
lindsayannerutter@gmail.com

[1] Bioinformatics and Computational Biology Program, Iowa State University, Ames, USA

Full list of author information is available at the end of the article

Abstract

We developed bigPint, an interactive data visualization package available on Bioconductor. Our software introduces new visualization technology that enables independent layers of interactivity using Plotly in R, aiding in the exploration of large datasets. The ability to select and aggregate data, link between plots, and tailor aesthetics in intelligent ways are all useful features of bigPint. Researchers can analyze and present their increasingly large biological datasets using intuitive and reproducible plots from bigPint. Developers can leverage our open-source code to create additional interactive visualization tools for computational biology tasks.

Keywords: Bioconductor; clustering; ggplot2; htmlwidgets; interactive; Plotly; R; RNA-seq; Shiny; visualization

Background

Interactive data visualization is increasingly imperative in the biological sciences [1]. When performing RNA-seq studies, researchers wish to determine which genes are differentially expressed between treatment groups. Interactive visualization can help them assess differentially expressed gene (DEG) calls before performing any subsequent functional enrichment analyses. New visualization tools for genomic data have incorporated interactive capabilities, and some believe this trend could enhance the exploration of genomic data in the future [2]. Despite the growing appreciation of the inherent value of interactive graphics, the availability of effective and easy-to-use interactive visualization tools for RNA-seq data remains limited.

Interactive visualization tools for genomic data can have restricted access when only available on certain operating systems and/or when requiring payment [3, 4, 5]. These limitations can be removed when tools are published on open-source repositories. Indeed, the Bioconductor project aims to foster interdisciplinary scientific research by promoting transparency and reproducibility while allowing software content to be used on Windows, MacOS, and Linux [6]. Bioconductor software is written in the R programming language, which also provides statistical and visualization methods that can facilitate the development of robust graphical tools.

Several interactive visualization methods for genomic data have been developed using Shiny, which is also based on the R programming language [7, 8, 9].

We recently developed bigPint, an interactive data visualization software package available on Bioconductor. The bigPint package allows users to visually explore many types of large multivariate datasets, even though it was more specifically developed for RNA-seq data. In a recent methods paper, we used public RNA-seq datasets to demonstrate how bigPint graphics can help biologists detect crucial issues with normalization methods and DEG designation in ways not possible with numerical models [10]. We also applied bigPint visualization tools in a recent research paper that sought to elicit how nutrition and viral infection affect the honey bee transcriptome [11]. In the current paper, we will now explain the technical innovations and merits of the bigPint package, including new interactive visualization techniques that we believe can be helpful in the development and usage of future biological visualization software. The bigPint website is available at https://lindsayrutter.github.io/bigPint and contains short vignette articles that provide example analysis pipelines, all written in reproducible code.

Results

Basic input

Each method in bigPint requires an input parameter *data* object. If a researcher is using the package to visualize RNA-seq data, then this *data* object should be a count table that contains the read counts for all genes of interest. The value in row i and column j should indicate how many reads have been assigned to gene i in sample j. This is the same input format required in popular RNA-seq count-based statistical packages, such as DESeq2, edgeR, limma, EBSeq, and BaySeq [12, 13, 14, 15, 16].

Several methods in bigPint also require an input parameter *dataMetrics* object. If a researcher is using the package to visualize RNA-seq data, then this *dataMetrics* object should be a subset of the data (usually DEGs) where each case includes quantitative values of interest (such as fold change and FDR). This information can be easily derived from popular RNA-seq numerical analysis packages. Again, this framework allows users to work smoothly between visualizations in the bigPint package and models in other Bioconductor packages, complying with the belief that the most efficient way to analyze large datasets is to iterate between models and visualizations.

Original features

1. Independent layers of interactivity

The Bioconductor community advanced the boundaries of biological visualization in the past and generally believes that modern interactive technology must be incorporated to continue these advancements [6]. We will define the term *geom-drawing interactivity* to indicate user queries that draw geoms (graphical representations of

the data, such as lines, hexagons, and points). This could mean the user adjusts sliders or selects buttons to draw a subset of the data from the database as geoms (such as points). We will define the term *geom-manipulating interactivity* to indicate user queries that alter already-drawn geoms. This could mean the user hovers over a geom (such as a hexagon) and obtains its associated metadata (such as the names of its contained genes). It could also mean the user zooms and pans to further alter how already-drawn geoms are displayed.

Our package introduces what we believe is a fairly new interactive visualization technology that is useful in the exploration of large biological datasets. Our technique allows for two independent layers of interactivity, for the foreground and background of the plot respectively. Each layer can include both *geom-drawing* and *geom-manipulating* interactivity. Our new technology can enhance the exploration of large datasets, especially in cases where one layer contains large amounts of data (such as the full dataset) and the other layer contains smaller amounts of data (such as a data subset). Because the layers are independent, users can save time and computation by keeping the layer with more data unaltered while only redrawing the layer with less data.

The idea of multi-layered interactive graphics in R was raised in [17] and a pilot implementation was made using qt software to create different listeners for different layers, an approach that is no longer functional. While the concept of independent layers of interactivity in R is not new in itself, the technology we introduce is new and solves a difficult problem that has been raised before. We achieved our independent double-layered interactivity using the onRender() method of the htmlwidgets package [18]. This method had the potential for the foreground layer to be overlaid via plotly traces while the plotly background layer did not need to be redrawn, something that could not foreseeably be achieved with the native onRender() method of the plotly package [19]. Specifically, the htmlwidgets onRender() method contains three input parameters: an HTML Widget object, a character vector containing JavaScript code, and a list of R objects that can be serialized to JSON format. To develop our technique, we specified a plotly object as the HTML Widget object, which allowed for an interactive background. Within the method, we wrote JavaScript code that enabled interactive foregrounds to be updated without redrawing interactive backgrounds. We used the R object list to transfer count tables and DEG lists into the method. In some our our applications, users can link between the layers of different interactive plots. This functionality was achieved by sending custom messages between the Shiny software and the JavaScript code within the htmlwidgets method [20]. We provide pseudocode and documented code for readers who wish to understand the details of how we created our interactive software (see Table 2). We will now briefly explain how our two-layered interactivity method can improve upon several of the RNA-seq visualization tools in our package.

1a. Scatterplot matrices

Scatterplot matrices have appeared in statistical graphics literature for almost four decades and used across various fields of multivariate research [21, 22, 23, 24].

Previous user studies have shown that participants performed better when using animated rather than static versions of scatterplot matrices. Users also preferred animated scatterplot matrices and found them easier to understand as they can alleviate overplotting issues [25]. Rendering scatterplot matrices interactive is promising but challenging with large datasets [26]. The number of background geoms that need to be drawn grows exponentially by dimension size: n-dimensional data corresponds to n^2 scatterplots. Our two-layered interactive visualization technology improves upon this dilemma by allowing details of interest to be superimposed in the foreground while the massive number of geoms in the background does not require redrawing. See Tables 1 and 2 for details (video, pseudocode, code, and application link) regarding our interactive scatterplot matrices.

1b. Litre plots

Problems still remain when scatterplot matrices are applied to large datasets. Physical space requirements increase exponentially. Hence, when extended to large dimensions, it becomes difficult to mentally link many small plots within the matrix [27]. Several techniques have been proposed to ameliorate this problem. Three dimensional scatterplots are useful but can cause occlusion and depth perception issues [27]. Other techniques like grand tours [28], projection pursuits [29, 30], and scagnostics [31] have been proposed.

Even though these alternative techniques are useful, they may not simultaneously display distributions across all cases (genes) and variables (samples). We generally want to compare replicate and treatment variability in RNA-seq data, which can be visually accomplished by plotting all genes and samples. We also want to superimpose DEGs to determine how their read count variability compares to that of the whole dataset. In light of this, we developed a plot that collapses the scatterplot matrix onto one Cartesian coordinate system, allowing users to visualize all read counts from one DEG of interest onto all read counts of all genes in the dataset. We call this new plot a repLIcate TREatment ("litre") plot. An in depth explanation about the litre plot can be found in our previous methods paper [10].

We believe our two-layered interactive visualization method is an indispensable component of the litre plot. Drawing the background (all genes in the dataset) is the time-limiting step, whereas drawing the foreground (one DEG of interest) is immediate. Most users would like to superimpose DEGs from a list one by one onto the background. This process would be unnecessarily time-prohibiting if the background needed to be redrawn each time the user progressed to the next DEG. Fortunately, our technology allows the user to immediately redraw the interactive foreground (the DEG of interest) while the background (all genes in the data) remains unchanged but preserved in its interactive capabilities. See Tables 1 and 2 for details (video, pseudocode, code, and application link) regarding our interactive litre plots.

1c. Volcano plots

Volcano plots draw significance and fold change on the vertical and horizontal axes respectively. In RNA-seq studies, volcano plots allow users to check that genes were not falsely deemed significant due to outliers, low expression levels, and batch effects [32]. Researchers benefit from the ability to quickly identify individual gene names in the volcano plot. This was previously achieved with the identify() method in R, which identifies the closest point in a scatterplot to the position nearest the mouse click [32]. The interactive volcano plot in bigPint can identify individual gene names in a less ambiguous fashion by responding to users hovering *directly* over corresponding points. It also improves upon traditional volcano plots by allowing users to threshold on statistical values in order to immediately update the superimposed gene subset without having to redraw the more computationally-heavy background that contains all genes. See Tables 1 and 2 for details (video, pseudocode, code, and application link) regarding our interactive volcano plots.

2. Consecutive box selection

The bigPint package provides interactive tools for consecutive box selection. A box selection is a rectangular query drawn directly on a two-dimensional graph. Users can specify a box selection by clicking on the desired starting point of the rectangular query and dragging the mouse pointer to the desired opposite corner point of the rectangular query. This procedure for generating rectangles is widely used in interactive programs and should be familiar to most users [33]. After the user releases the mouse, the query is processed and only the data cases that were inside the specified rectangle remain. More precisely, a data case remains in a box selection queried between (x_1, y_1) and (x_2, y_2) if every point within $x_1 \leq x \leq x_2$ is also within $y_1 \leq y \leq y_2$ (where $y_2 \geq y_1$ and $x_2 \geq x_1$). The user can specify consecutive queries with multiple box selections. The consecutive box selection model is convenient in cases where identical thresholds are desired over adjacent features. In these cases, a single box selection of width w can be used to simultaneously query the same threshold across w features. This process is an improvement over single-feature box selection widgets, where w individual queries would be required [33].

Consecutive box selection may have originally been designed for time series data, but has since proven useful for detecting patterns in gene expression data. Combined with parallel coordinate plots, the consecutive box selection technique has been used to elicit candidate regulatory splice sequences showing high values at some positions and low values at other positions [33]. In RNA-seq, this technology can also be used to investigate differential expression showing high read counts for one treatment group and low read counts for another treatment group, requiring a consecutive query. Consecutive box selection tools have been published for gene expression analysis sofware that was restricted for certain operating systems [33]. We believe that publishing consecutive box selection tools in a platform like R can be useful for computational biologists using various operating systems. See Tables 2 for details (video, pseudocode, code, and application link) regarding our interactive parallel coordinate plots that feature consecutive box selection.

Useful features

Tailoring and saving static plots

Static plots can be saved as list objects in the R workspace and/or as JPG files to a directory chosen by the user. Saving plots into the R workspace allows users to integrate them into analysis workflows. It also allows them to tailor the plots (such as adding titles and changing label sizes) using the grammar of graphics via the conventional + syntax. Saving plots to a directory allows users to keep professional-looking files that can be inserted into proposals and talks. By default, the bigPint package saves static plots both in the R workspace and a directory (the default location is tempdir()).

Second feature layer

Both static and interactive plots allow for a subset of data to be plotted in a different manner than the full dataset. When analyzing RNA-seq data, this second feature layer could represent DEGs. There are three options for creating data subsets with static plots. First, users can threshold the previously-mentioned *dataMetrics* object by one of its quantitative variables. Second, users can simply declare a geneList object that contains the list of data subset IDs. Third, the user can simply leave the *dataMetrics* and geneList objects to their default value of NULL and not overlay any data subsets.

Group comparison filters

When users create static plots, the package automatically creates a separate plot for each pairwise combination of treatment groups from the inputted data. When users explore interactive plots, fields are dynamically generated from the inputted data so that any pairwise combination of treatment groups can be selected by buttons. Users can then quickly flip between contrasts in their data. The bigPint package comes with an example soybean cotyledon dataset that has three treatment groups, which is used across several easy-to-follow articles on the package website. These assets can assist users who have data containing more than two treatment groups.

Hexagonal binning

Most bigPint plots represent genes using point geoms (where each point represents one gene) or hexagonal binning geoms (where each hexagon color represents the number of genes in that area). Plotting each gene as a point allows for ideal levels of detail but overplotting can occur as the data increases, which makes it difficult to determine how many genes are in a given area. Hexagonal binning has been used in prior software to successfully manage overplotting issues [26, 34] and has shown superior time performance because less geom objects need to be plotted. The bigPint package allows users to draw the background using either geom, as preferences can depend on the dataset.

Hierarchical clustering

Users can conduct hierarchical clustering analyses on their data using the function plotClusters(). By default, the resulting clusters will be plotted as parallel coordinate lines superimposed onto side-by-side boxplots that represent the five-number summary of the full dataset. There are three main approaches in the plotClusters() function:

- Approach 1: The clusters are formed by clustering only on a user-defined subset of data (such as significant genes). Only these user-defined genes are overlaid as parallel coordinate lines.
- Approach 2: The clusters are formed by clustering the full dataset. Then, only a user-defined subset of data (such as significant genes) are overlaid as parallel coordinate lines.
- Approach 3: The clusters are formed by clustering the full dataset. All genes are overlaid as parallel coordinate lines.

The clustering algorithm is based on the hclust() and cutree() functions in the R stats package. It offers the same set of agglomeration methods ("ward.D", "ward.D2", "single", "complete", "average", "mcquitty", "median", and "centroid") with "ward.D" as the default. In many cases, users may want to save clusters derived from the plotClusters() function for later use, such as to overlay them onto scatterplot matrices, litre plots, and volcano plots. The gene IDs of each cluster can be saved as .RDS files for this purpose by setting the verbose option of the plotClusters() function to a value of TRUE.

Various plot aesthetics

Users can modify various aesthetics for both static and interactive plots, including geom size. Some plots also provide alpha blending, which can benefit users plotting large datasets as parallel coordinate lines [35]. Statistical coloring is inconsistent in numerous packages even though it can greatly enhance biological data visualization [36]. The bigPint package allows users to maintain consistent coloring across hierarchical clusters and when working between various plots.

Selection and aggregation

Some techniques that are effective in data exploration may lose their efficiency and eventually fail as data size increases. Two main approaches to solving this problem are data selection and data aggregation [37]. Data selection means that only a subset of the full data is displayed at a given time. The data subset can be selected through queries and interactive controls which allow the user to quickly examine different data subsets [37]. Data aggregation means that the full dataset is divided into data subsets (called aggregates) that reduce the amount of data being simultaneously visualized. Users with large datasets should ideally be able to perform both data selection and data aggregation [37]. The bigPint package allows

users to easily perform data selection using queries (such as thresholds and sliders) and interactive controls (such as zooming, box and lasso selection, and panning) and to perform data aggregation using hierarchical clustering.

Shiny interactivity

Interactive plots in the bigPint package open as Shiny applications that consist of simple dashboards with "About" tabs that explain how to use the applications. They also include "Application" tabs that provide several input fields for the user to tailor their plots. Some of these input fields are generated dynamically from the inputted dataset so that users have more convenience in how they select data subsets. In these applications, users can also download lists of selected genes and static images of interactive graphics to their local computers.

Shiny applications can be launched on a local personal computer, hosted on a local or cloud-based server, or hosted for free on the shinyapps.io website. As such, interactive bigPint packages can be deployed on a personal computer using only a local file containing the data, the bigPint package and its dependencies, R / RStudio, and a browser recommended by Shiny (Google Chrome or Mozilla Firefox). This method does not require internet connectivity, which can be useful for users who are protecting sensitive data, analyzing or presenting data in contexts without reliable connectivity, or testing and developing applications.

Discussion

Researchers benefit when they are able to view multiple perspective of their data, especially when working with large datasets [38, 39]. The ability to select and aggregate data, threshold data to create subsets, link between multiple plots, interact with plots, and tailor various aesthetics in intelligent ways are all useful features of the bigPint package [1, 2, 27]. We expect that bigPint will enable researchers to generate and interact with intuitive, high quality, and reproducible plots from increasingly large biological datasets.

Conclusion

Despite the growing appreciation of the inherent value in interactive graphics, the availability of easy-to-use and effective interactive exploratory visualization tools for RNA-seq data remains limited. In this paper, we introduced new visualization tools that enable independent layers of interactive capabilities for the foreground and background of plots using plotly in R. We believe this methodology represents a fairly novel contribution to the field of interactive data visualization that can lead to sizable performance gains when working with large datasets. Advocating state-of-the-art visualization tools is crucial for biology researchers to analyze and present their data and for visualization researchers to develop novel methods. Lessons learned from our open-source work may encourage the development of additional interactive visualization tools for various computational biology tasks.

Methods

bigPint was released under the GPL-3 license. Most bigPint visualization methods were constructed using htmlwidgets [18], ggplot2 [40], shiny [20], shinyapps.io [41], and plotly [19]. bigPint methods were tested on numerous RNA-seq datasets [10, 11]. The package website was constructed using the pkgdown software [42]. bigPint can be downloaded from the Bioconductor website [6].

Availability of data and materials

The bigPint package is available on Bioconductor at https://bioconductor.org/packages/devel/bioc/html/bigPint.html. The software website that contains thorough and reproducible vignettes is available at https://lindsayrutter.github.io/bigPint. The four public datasets used as examples in the bigPint package are available online: Three are deposited on the NCBI Sequence Read Archive with accession numbers SRA000299 [43], PRJNA318409 [44], and SRA048710 [45]. One is deposited on the NCBI Gene Expression Omnibus with accession number GSE61857 [46].

Competing interests

The authors declare that they have no competing interests.

Author's contributions

The software was implemented and the paper was written primarily by the first author, based on initial ideas, consequent advice, and small contributions from the second author.

Acknowledgements

We would like to thank Roxane Legaie from Monash University and Dr. Amy L. Toth from Iowa State University for their helpful feedback. We also thank the open-source community for their numerous outlets of support in regards to R, plotly, htmlwidgets, ggplot2, and shiny, without which this project would not have been possible.

Author details

[1] Bioinformatics and Computational Biology Program, Iowa State University, Ames, USA. [2] Econometrics and Business Statistics, Monash University, Clayton VIC, Australia.

References

1. O'Donoghue, S.I., Gavin, A.-C., Gehlenborg, N., Goodsell, D.S., Hériché, J.-K., Nielsen, C.B., North, C., Olson, A.J., Procter, J.B., Shattuck, D.W., *et al.*: Visualizing biological data—now and in the future. Nature methods **7**(3), 2 (2010)
2. Pavlopoulos, G.A., Malliarakis, D., Papanikolaou, N., Theodosiou, T., Enright, A.J., Iliopoulos, I.: Visualizing genome and systems biology: technologies, tools, implementation techniques and trends, past, present and future. Gigascience **4**(1), 38 (2015)
3. Seo, J., Shneiderman, B.: Interactively exploring hierarchical clustering results [gene identification]. Computer **35**(7), 80–86 (2002)
4. Ahlberg, C.: Spotfire: an information exploration environment. ACM SIGMOD Record **25**(4), 25–29 (1996)
5. Chu, L., Scharf, E., Kondo, T.: Genespringtm: tools for analyzing microarray expression data. Genome Informatics **12**, 227–229 (2001)
6. Gentleman, R.C., Carey, V.J., Bates, D.M., Bolstad, B., Dettling, M., Dudoit, S., Ellis, B., Gautier, L., Ge, Y., Gentry, J., *et al.*: Bioconductor: open software development for computational biology and bioinformatics. Genome biology **5**(10), 80 (2004)
7. Rue-Albrecht, K., Marini, F., Soneson, C., Lun, A.T.: isee: Interactive summarizedexperiment explorer. F1000Research **7** (2018)
8. Schultheis, H., Kuenne, C., Preussner, J., Wiegandt, R., Fust, A., Bentsen, M., Looso, M.: Wilson: Web-based interactive omics visualization. Bioinformatics **35**(6), 1055–1057 (2018)
9. Hughes, L.D., Lewis, S.A., Hughes, M.E.: Expressiondb: An open source platform for distributing genome-scale datasets. PloS one **12**(11), 0187457 (2017)
10. Rutter, L., Moran Lauter, A.N., Graham, M.A., Cook, D.: Visualization methods for differential expression analysis. Accepted to BMC Bioinformatics on June 24, 2019

11. Rutter, L., Carrillo-Tripp, J., Bonning, B.C., Cook, D., Toth, A.L., Dolezal, A.G.: Transcriptomic responses to diet quality and viral infection in apis mellifera. BMC genomics **20**(1), 412 (2019)
12. Love, M.I., Huber, W., Anders, S.: Moderated estimation of fold change and dispersion for rna-seq data with deseq2. Genome biology **15**(12), 550 (2014)
13. Robinson, M.D., McCarthy, D.J., Smyth, G.K.: edger: a bioconductor package for differential expression analysis of digital gene expression data. Bioinformatics **26**(1), 139–140 (2010)
14. Ritchie, M.E., Phipson, B., Wu, D., Hu, Y., Law, C.W., Shi, W., Smyth, G.K.: limma powers differential expression analyses for rna-sequencing and microarray studies. Nucleic acids research **43**(7), 47–47 (2015)
15. Leng, N., Dawson, J.A., Thomson, J.A., Ruotti, V., Rissman, A.I., Smits, B.M., Haag, J.D., Gould, M.N., Stewart, R.M., Kendziorski, C.: Ebseq: an empirical bayes hierarchical model for inference in rna-seq experiments. Bioinformatics **29**(8), 1035–1043 (2013)
16. Hardcastle, T.J., Kelly, K.A.: bayseq: empirical bayesian methods for identifying differential expression in sequence count data. BMC bioinformatics **11**(1), 422 (2010)
17. Xie, Y., Hofmann, H., Cheng, X.: Reactive programming for interactive graphics. Statistical Science, 201–213 (2014)
18. Vaidyanathan, R., Xie, Y., Allaire, J., Cheng, J., Russell, K.: Htmlwidgets: HTML Widgets for R, 2016. https://cran.r-project.org/package=htmlwidgets Accessed 2018-12-20
19. Sievert, C., Parmer, C., Hocking, T., Chamberlain, S., Ram, K., Corvellec, M., Despouy, P.: Plotly: Create Interactive Web Graphics Via 'plotly.js'. https://cran.r-project.org/package=plotly Accessed 2018-12-20
20. Chang, W., Cheng, J., Allaire, J., Xie, Y., McPherson, J.: Shiny: Web Application Framework for R [Computer Software]. https://cran.r-project.org/package=shiny Accessed 2018-12-20
21. Becker, R.A., Cleveland, W.S.: Brushing a scatterplot matrix: High-interaction graphical methods for analyzing multidimensional data. submitted for publication (1984)
22. Carr, D., Nicholson, W.: Graphical interaction tools for multiple 2-and 3-dimensional scatterplots. Technical report, Pacific Northwest Lab., Richland, WA (USA) (1984)
23. Tufte, E.R.: The Visual Display of Quantitative Information. Graphics press, Cheshire, CT (2001)
24. Tukey, P., Tukey, J.: Graphical display of data sets in three or more dimensions. Three papers in Interpreting Multivariate Data (ed. V. Barnett), 189–275. Chichester: Wiley (1981)
25. Chen, H., Engle, S., Joshi, A., Ragan, E.D., Yuksel, B.F., Harrison, L.: Using animation to alleviate overdraw in multiclass scatterplot matrices. In: Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems, p. 417 (2018). ACM
26. Carr, D.B., Littlefield, R.J., Nicholson, W., Littlefield, J.: Scatterplot matrix techniques for large n. Journal of the American Statistical Association **82**(398), 424–436 (1987)
27. Kerren, A., Ebert, A., Meyer, J.: Human-Centered Visualization Environments: GI-Dagstuhl Research Seminar, Dagstuhl Castle, Germany, March 5-8, 2006, Revised Papers. Springer, Dagstuhl Castle, Germany (2007)
28. Asimov, D.: The grand tour: a tool for viewing multidimensional data. SIAM journal on scientific and statistical computing **6**(1), 128–143 (1985)
29. Friedman, J.H., Tukey, J.W.: A projection pursuit algorithm for exploratory data analysis. IEEE Transactions on computers **100**(9), 881–890 (1974)
30. Cook, D., Buja, A., Cabrera, J., Hurley, C.: Grand tour and projection pursuit. Journal of Computational and Graphical Statistics **4**(3), 155–172 (1995)
31. Wilkinson, L., Anand, A., Grossman, R.: Graph-theoretic scagnostics. In: IEEE Symposium on Information Visualization, 2005. INFOVIS 2005., pp. 157–164 (2005). IEEE
32. Li, W.: Application of volcano plots in analyses of mrna differential expressions with microarrays. arXiv preprint arXiv:1103.3434 (2011)
33. Hochheiser, H., Baehrecke, E.H., Mount, S.M., Shneiderman, B.: Dynamic querying for pattern identification in microarray and genomic data. In: 2003 International Conference on Multimedia and Expo. ICME'03. Proceedings (Cat. No. 03TH8698), vol. 3, p. 453 (2003). IEEE
34. Harshbarger, J., Kratz, A., Carninci, P.: Deiva: a web application for interactive visual analysis of differential gene expression profiles. BMC genomics **18**(1), 47 (2017)
35. Unwin, A., Chen, C.-h., Härdle, W.: Computational Statistics and Data Visualization. Humboldt-Universität zu Berlin, Wirtschaftswissenschaftliche Fakultät, Berlin (2007)
36. Yin, T., Cook, D., Lawrence, M.: ggbio: an r package for extending the grammar of graphics for genomic data. Genome biology **13**(8), 77 (2012)
37. Andrienko, G., Andrienko, N.: Blending aggregation and selection: Adapting parallel coordinates for the visualization of large datasets. The Cartographic Journal **42**(1), 49–60 (2005)
38. Swayne, D.F., Lang, D.T., Buja, A., Cook, D.: Ggobi: evolving from xgobi into an extensible framework for interactive data visualization. Computational Statistics & Data Analysis **43**(4), 423–444 (2003)
39. Cook, D., Swayne, D.F., Buja, A.: Interactive and Dynamic Graphics for Data Analysis: with R and GGobi. Springer, New York (2007)
40. Wickham, H.: Ggplot2: Elegant Graphics for Data Analysis. Springer, Switzerland (2016)
41. RStudio: Integrated Development for R. http://www.rstudio.com Accessed 2018-12-20
42. Wickham, H., Hesselberth, J.: Pkgdown: Make Static HTML Documentation for a Package. https://cran.r-project.org/package=pkgdown Accessed 2018-12-20
43. Marioni, J.C., Mason, C.E., Mane, S.M., Stephens, M., Gilad, Y.: Rna-seq: an assessment of technical reproducibility and comparison with gene expression arrays. Genome research (2008)
44. Moran Lauter, A.N., Graham, M.A.: NCBI SRA Bioproject Accession: PRJNA318409. https://www.ncbi.nlm.nih.gov/bioproject/PRJNA318409 Accessed 2018-12-20
45. Risso, D., Schwartz, K., Sherlock, G., Dudoit, S.: Gc-content normalization for rna-seq data. BMC bioinformatics **12**(1), 480 (2011)
46. Brown, A.V., Hudson, K.A.: Developmental profiling of gene expression in soybean trifoliate leaves and cotyledons. BMC plant biology **15**(1), 169 (2015)

Figures

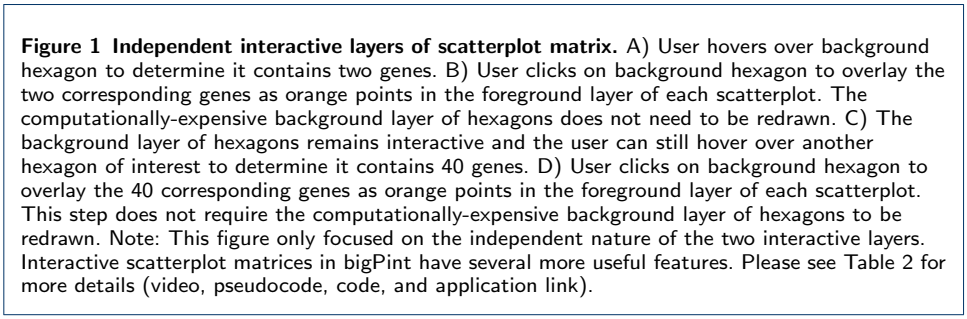

Figure 1 Independent interactive layers of scatterplot matrix. A) User hovers over background hexagon to determine it contains two genes. B) User clicks on background hexagon to overlay the two corresponding genes as orange points in the foreground layer of each scatterplot. The computationally-expensive background layer of hexagons does not need to be redrawn. C) The background layer of hexagons remains interactive and the user can still hover over another hexagon of interest to determine it contains 40 genes. D) User clicks on background hexagon to overlay the 40 corresponding genes as orange points in the foreground layer of each scatterplot. This step does not require the computationally-expensive background layer of hexagons to be redrawn. Note: This figure only focused on the independent nature of the two interactive layers. Interactive scatterplot matrices in bigPint have several more useful features. Please see Table 2 for more details (video, pseudocode, code, and application link).

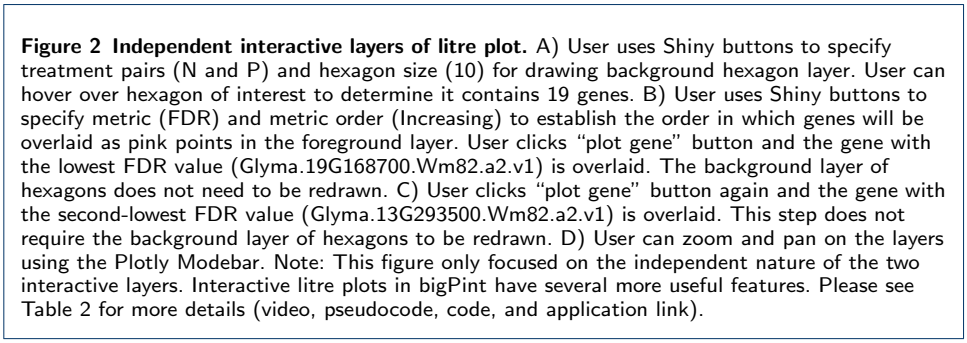

Figure 2 Independent interactive layers of litre plot. A) User uses Shiny buttons to specify treatment pairs (N and P) and hexagon size (10) for drawing background hexagon layer. User can hover over hexagon of interest to determine it contains 19 genes. B) User uses Shiny buttons to specify metric (FDR) and metric order (Increasing) to establish the order in which genes will be overlaid as pink points in the foreground layer. User clicks "plot gene" button and the gene with the lowest FDR value (Glyma.19G168700.Wm82.a2.v1) is overlaid. The background layer of hexagons does not need to be redrawn. C) User clicks "plot gene" button again and the gene with the second-lowest FDR value (Glyma.13G293500.Wm82.a2.v1) is overlaid. This step does not require the background layer of hexagons to be redrawn. D) User can zoom and pan on the layers using the Plotly Modebar. Note: This figure only focused on the independent nature of the two interactive layers. Interactive litre plots in bigPint have several more useful features. Please see Table 2 for more details (video, pseudocode, code, and application link).

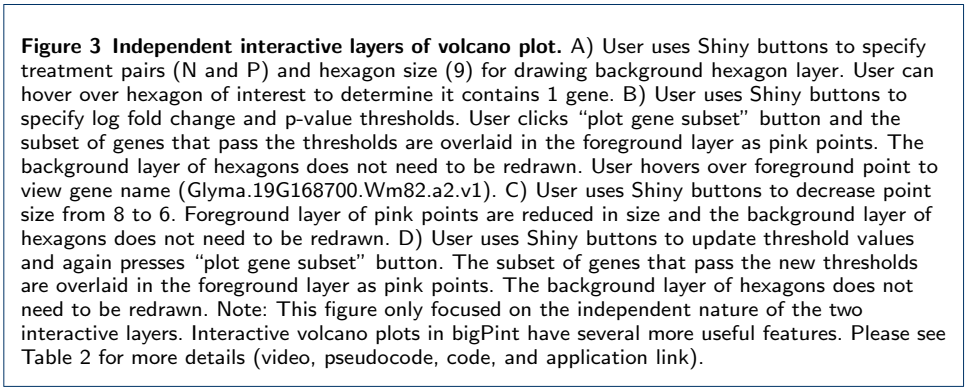

Figure 3 Independent interactive layers of volcano plot. A) User uses Shiny buttons to specify treatment pairs (N and P) and hexagon size (9) for drawing background hexagon layer. User can hover over hexagon of interest to determine it contains 1 gene. B) User uses Shiny buttons to specify log fold change and p-value thresholds. User clicks "plot gene subset" button and the subset of genes that pass the thresholds are overlaid in the foreground layer as pink points. The background layer of hexagons does not need to be redrawn. User hovers over foreground point to view gene name (Glyma.19G168700.Wm82.a2.v1). C) User uses Shiny buttons to decrease point size from 8 to 6. Foreground layer of pink points are reduced in size and the background layer of hexagons does not need to be redrawn. D) User uses Shiny buttons to update threshold values and again presses "plot gene subset" button. The subset of genes that pass the new thresholds are overlaid in the foreground layer as pink points. The background layer of hexagons does not need to be redrawn. Note: This figure only focused on the independent nature of the two interactive layers. Interactive volcano plots in bigPint have several more useful features. Please see Table 2 for more details (video, pseudocode, code, and application link).

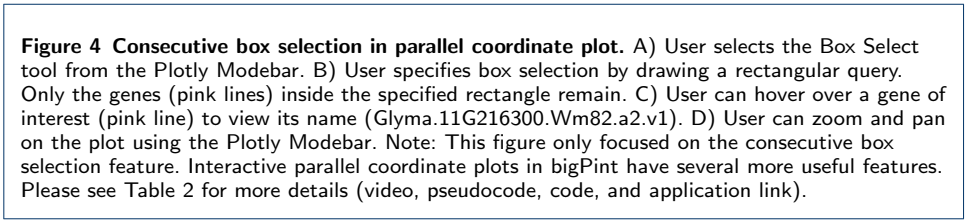

Figure 4 Consecutive box selection in parallel coordinate plot. A) User selects the Box Select tool from the Plotly Modebar. B) User specifies box selection by drawing a rectangular query. Only the genes (pink lines) inside the specified rectangle remain. C) User can hover over a gene of interest (pink line) to view its name (Glyma.11G216300.Wm82.a2.v1). D) User can zoom and pan on the plot using the Plotly Modebar. Note: This figure only focused on the consecutive box selection feature. Interactive parallel coordinate plots in bigPint have several more useful features. Please see Table 2 for more details (video, pseudocode, code, and application link).

Tables

Table 1 Examples of independent layers of interactivity

Plot	Layer	Geom-drawing interactivity	Geom-manipulation interactivity
Scatterplot matrix	Background	None	User hovers over background hexagons to view gene counts
	Foreground	User clicks on background hexagon to draw corresponding genes as foreground points. Background layer does not need to be redrawn	User hovers over foreground points to view gene names
Litre plot	Background	User uses Shiny buttons to specify treatment pairs and hexagon sizes for drawing background hexagons	User hovers over background hexagons to view gene counts
	Foreground	User uses Shiny buttons to specify metric, metric order, and point size for drawing foreground points. Background layer does not need to be redrawn	User hovers over foreground points to view gene names
Volcano plot	Background	User uses Shiny buttons to specify treatment pairs and hexagon sizes for drawing background hexagons	User hovers over background hexagons to view gene counts
	Foreground	User uses Shiny buttons to specify point size, log fold changes, p-values to draw foreground points. Background layer does not need to be redrawn	User hovers over foreground points to view gene names

Table 2 Helpful resources about interactive graphics in bigPint

Plot	Figure	Video	Application	Pseudocode	Code
Scatterplot matrix	Figure 1	bit.ly/smplotVid	bit.ly/smplotApp	Algorithm 1	bit.ly/smplotCode
Litre plot	Figure 2	bit.ly/litreVid	bit.ly/litreApp	Algorithm 2	bit.ly/litreCode
Volcano plot	Figure 3	bit.ly/volcVid	bit.ly/volcApp	Algorithm 3	bit.ly/volcCode
Parallel coordinate plot	Figure 4	bit.ly/pcplotVid	bit.ly/pcplotApp	Algorithm 4	bit.ly/pcplotCode

```
Data: Data frame input by user
Result: Interactive scatterplot matrix
/* Declare Shiny server
server ← function(input, output, session){
    /* Declare Shiny output scatterplot matrix
    output$scatMatPlot ← renderPlotly({
        /* Draw hexagons and x=y line in bottom-left corner of matrix
        my_fn ← function(data, mapping){}
        /* Create static scatterplot matrix
        p ← ggpairs(data, lower = list(continuous = my_fn))
        /* Convert ggplot2::ggplot() object to plotly object
        ggP ← ggplotly(p)
        /* Tailor plotly scatterplot matrix interactivity with JavaScript
        ggPR ← ggP %>% onRender("function(el, x, data){
            /* If the user clicks on the plotly scatterplot matrix object
            el.on('plotly_click', function(e){
                /* Delete any old superimposed plotly geoms (orange dots)
                if (x.data.length > 0){Plotly.deleteTraces(el.id)}
                /* Determine gene IDs selected by user click. Save as object called selID with handle called
                   'selID' so it can be read outside current JavaScript function back in Shiny
                Shiny.onInputChange('selID', selID)
                /* Create traces for selected gene IDs as orange points that state gene names upon hovering
                trace = {mode: 'markers', color: 'orange', size: 6, text: selID, hoverinfo: 'text'}
                /* Superimpose traces onto the plotly scatterplot matrix object
                Plotly.addTraces(el.id, Traces)
            })
        })
        /* Pass the R data object into the JavaScript function
        ", data = data
    })
    /* Read into Shiny the gene IDs that user clicked on
    selID ← reactive(input$selID)
    /* Create data subset (read counts) for only the selected gene IDs
    pcpDat ← reactive(data[which(data$ID %>% selID()), ])
    /* Create static box plot of the full dataset
    BP ← ggplot(data) + geom_boxplot()
    /* Render boxplot interactive as a plotly object
    ggBP ← ggplotly(BP)
    /* Declare Shiny output boxplot
    output$boxPlot ← renderPlotly({
        /* Tailor interactivity of the plotly boxplot object using custom JavaScript
        ggBP %>% onRender("function(el, x, data){
        /* Create traces for selected gene IDs as orange lines that state gene names upon hovering
        trace = {mode: 'lines', color: 'orange', width: 1.5, text: selID, hoverinfo: 'text'}
        /* Push traces to be superimposed onto the plotly scatterplot matrix object
        Plotly.addTraces(el.id, Traces)
        /* Pass R objects into the JavaScript function
        }", data = list(pcpDat = pcpDat())
    })
}
```

Algorithm 1: Pseudocode for interactive scatterplot matrix

```
Data: Data frame input by user
Result: Interactive litre plot
/* Declare Shiny server
server ← function(input, output, session){
    /* User input options
    observeEvent(input$goButton, values$x ← values$x + 1)
    observeEvent(input$selPair, values$x ← 0)
    observeEvent(input$selMetric, values$x ← 0)
    observeEvent(input$selOrder, values$x ← 0)
    observeEvent(input$binSize, values$x ← 0)
    /* Create reactive expression of plotly background litre plot
    gP ← reactive(p ← ggplot(data); gP ← ggplotly(p))
    /* Declare shiny output litre plot
    output$litrePlot ← renderPlotly({
        /* Create reactive expression of plotly background litre plot
        plotlyLitre ← reactive(gP())
        /* Tailor interactivity of the plotly litre plot object using custom JavaScript
        plotlyLitre() %>% onRender("function(el, x, data){
            /* Read handle called 'points' to obtain variables sent from R into JavaScript
            Shiny.addCustomMessageHandler('points', function(drawPoints){
                /* Delete any old superimposed plotly geoms (dots)
                if (x.data.length > 0){Plotly.deleteTraces(el.id)}
                /* Create traces for selected gene IDs as points that state gene names upon hovering
                trace = {x: drawPoints.geneX, y: drawPoints.geneY, mode: 'markers', color:
                  drawPoints.pointColor, size: drawPoints.pointSize, text: drawPoints.geneID, hoverinfo: 'text'}
                /* Superimpose traces onto the plotly litre plot object
                Plotly.addTraces(el.id, trace)
            })
        }")
    })
    /* If the user changes their input, store information about new superimposed genes with a handle called
      'points'. These values can then be sent from R to JavaScript
    observe({session$sendCustomMessage(type = "points", message=list(geneX=geneX, geneY=geneY,
      pointSize = pointSize, geneID=geneID, pointColor=pointColor))})
    /* Declare Shiny output boxplot
    output$boxPlot ← renderPlotly({
        /* Create reactive expression of plotly background boxplot BP ← reactive(ggplot() +
          geom_boxplot()) ggBP ← reactive(ggplotly(BP()))
        /* If the user changes their input, store information about new superimposed genes with a handle
          called 'lines'. These values can then be sent from R to JavaScript
        observe({session$sendCustomMessage(type = "lines", message=list(geneInfo=currGene(),
          geneID=geneID, pointColor=pointColor))})
        /* Tailor interactivity of the plotly boxplot object using custom JavaScript
        ggBP() %>% onRender("function(el, x, data){
            /* Read handle called 'lines' to obtain variables sent from R into JavaScript
            Shiny.addCustomMessageHandler('lines', function(drawLines){
                /* Delete any old superimposed plotly geoms (lines)
                Plotly.deleteTraces(el.id, traceLine)
                /* Create traces for selected gene IDs as lines that state gene names upon hovering
                traceLine = {x: drawLines.geneInfo, y: drawLines.geneInfo, mode: 'lines', color:
                  drawLines.pointColor, width: 2, opacity: 0.9; text: drawLines.geneID, hoverinfo: 'text'}
                /* Superimpose traces onto the plotly litre plot object
                Plotly.addTraces(el.id, traceLine)
            })
        })
    })
}
```

Algorithm 2: Pseudocode for interactive litre plot

```
Data: Data frame input by user
Result: Interactive volcano plot
/* Declare Shiny server
server ← function(input, output, session){
    /* Shiny user input options
    /* Define largest fold change dynamically based on data
    fcInMax ← max(ldply(dataMetrics, rbind)[["logFC"]])
    /* Construct dynamic input Shiny slider for fold change
    output$slider ← renderUI(sliderInput("logFC", "Log fold change:", min=0, max=fcInMax, step=0.1))
    /* Declare shiny output volcano plot
    output$volPlot ← renderPlotly({
        /* Create reactive expression of plotly background volcano plot
        gP ← reactive(p ← ggplot(data); gP ← ggplotly(p))
        /* Create reactive expression of plotly background volcano plot
        plotlyVol ← reactive(gP())
        /* Tailor interactivity of the plotly volcano plot object using custom JavaScript
        plotlyVol() %>% onRender("function(el, x, data){
            /* Read handle called 'points' to obtain variables sent from R into JavaScript
            Shiny.addCustomMessageHandler('points', function(drawPoints){
                /* Delete any old superimposed plotly geoms (dots)
                if (x.data.length > 0){Plotly.deleteTraces(el.id)}
                /* Create traces for selected gene IDs as points that state gene names upon hovering
                trace = {x: drawPoints.geneX, y: drawPoints.geneY, mode: 'markers', color:
                  drawPoints.pointColor, size: drawPoints.pointSize, text: drawPoints.geneID, hoverinfo: 'text'}
                /* Superimpose traces onto the plotly volcano plot object
                Plotly.addTraces(el.id, trace)
            })
        }")
    })
    /* If the user changes their input, store information about new superimposed genes with a handle called
      'points'. These values can then be sent from R to JavaScript
    observe({session$sendCustomMessage(type = "points", message=list(geneX=geneX, geneY=geneY,
      pointSize = pointSize, geneID=geneID, pointColor=pointColor))})
    /* Declare Shiny output boxplot
    output$boxPlot ← renderPlotly({
        /* Create reactive expression of plotly background boxplot
        BP ← reactive(ggplot() + geom_boxplot())
        ggBP ← reactive(ggplotly(BP()))
        /* Tailor interactivity of the plotly boxplot object using custom JavaScript
        ggBP() %>% onRender("function(el, x, data){
            /* Read handle called 'lines' to obtain variables sent from R into JavaScript
            Shiny.addCustomMessageHandler('lines', function(drawLines){
                /* Delete any old superimposed plotly geoms (lines)
                Plotly.deleteTraces(el.id, traceLine)
                /* Create xArr and yArr, array of x and y values for superimposed lines, from drawLines
                /* Create traces for selected gene IDs as lines that state gene names upon hovering
                traceLine = {x: xArr, y: yArr, mode: 'lines', color: drawLines.pointColor, width: 2, opacity: 0.9;
                  text: drawLines.geneID, hoverinfo: 'text'}
                /* Superimpose traces onto the plotly volcano plot object
                Plotly.addTraces(el.id, traceLine)
            })
        })
    })
    /* If the user changes their input, store information about new superimposed genes with a handle called
      'lines'. These values can then be sent from R to JavaScript
    observe({session$sendCustomMessage(type = "lines", message=list(geneID=geneID,
      pointColor=pointColor))})
}
```

Algorithm 3: Pseudocode for interactive volcano plot

```
Data: Data frame input by user
Result: Interactive parallel coordinate plot
/* Declare Shiny server
server ← function(input, output, session){
    /* Create an empty ggplotly object with same dimensions as data
    p ← ggplot(); gp ← ggplotly(p)
    /* Send user-defined line color into onRender() function
    session$sendCustomMessage(type = "lines", message = list(lineColor = lineColor))
    /* Declare Shiny output parallel coordinate plot
    output$pcpPlot ← renderPlotly({
        /* Tailor interactivity of plotly parallel coordinate plot using custom JavaScript
        gp %>% onRender("function(el, x, data){
            /* Define rects array object to hold IDs of remaining lines
            var rects = []
            /* Draw lines for the original dataset
            var pcpLine = {x: xArr, y: yArr, mode: 'lines', color: data.lineColor}
            Traces.push(pcpLine)
            Plotly.addTraces(el.id, Traces)
            /* If the user draws a rectangle
            el.on('plotly_selected', function(e){
                /* Delete lines outside the user-defined rectangle
                Traces.push(delLine)
                Plotly.addTraces(el.id, Traces)
                /* Create new object called pcpDat that only contains genes within rectangle
                /* Push IDs of remaining lines to rects array object
                for (a=0; a<pcpDat.length; a++){rects.push(pcpDat[a]['ID'])}
                /* Save remaining gene IDs in an object called rects with handle called 'rects' so it can be read
                   outside current JavaScript function back in Shiny
                Shiny.onInputChange('rects', rects)
            })
        })
        /* Pass R objects into the JavaScript function
        ", data = list(pcpDat = pcpDat, lineColor = lineColor)
    })
    /* Read into Shiny the gene IDs that remain within rectangle drawn by user
    inputRectDf ← reactive(input$rects)
    /* Print the selected gene IDs in interactive data table
    output$rectdf = renderDataTable(pcpDat %>% filter(ID %in% inputRectDf()))
}
```

Algorithm 4: Pseudocode for interactive parallel coordinate plot