Journal of Statistical Software

December 2018, Volume 87, Code Snippet 3. *doi: 10.18637/jss.v087.c03*

ggtern: Ternary Diagrams Using ggplot2

Nicholas E. Hamilton
The University of New South Wales

Michael Ferry
The University of New South Wales

Abstract

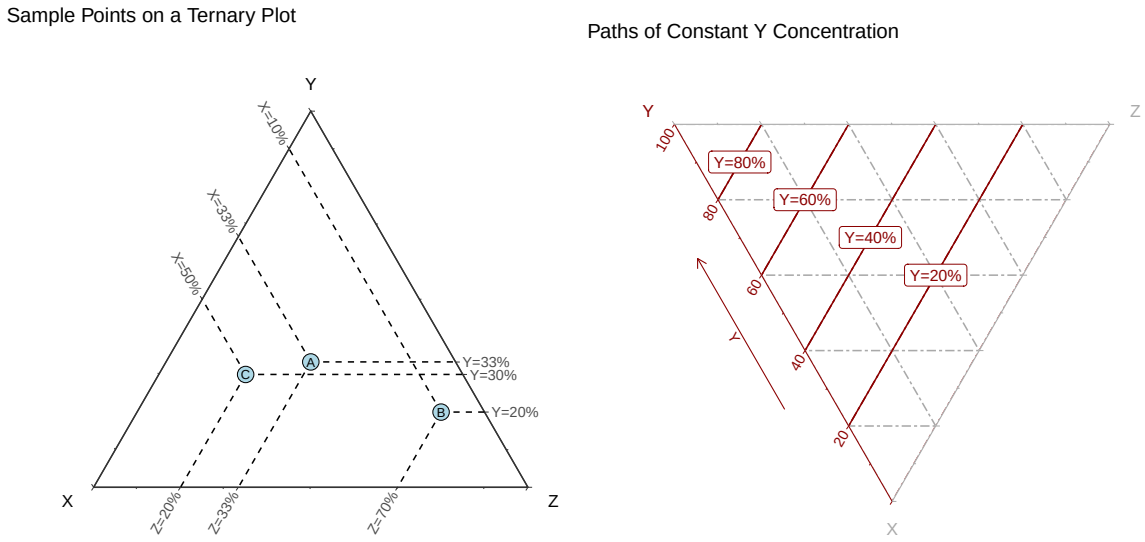
This paper presents the **ggtern** package for R, which has been developed for the rendering of ternary diagrams. Based on the well-established **ggplot2** package (Wickham 2009), the present package adopts the familiar and convenient programming syntax of its parent. We demonstrate that **ggplot2** can be used as the basis for producing specialized plotting packages and, in the present case, a package has been developed specifically for the production of high quality ternary diagrams. In order to produce **ggtern**, it was necessary to overcome a number of design issues, such as finding a means to modify existing geometries designed for a 2D Cartesian coordinate system and permitting them to function in an environment that requires an additional spatial aesthetic mapping. In the present paper, we provide examples of this package in its most basic form followed by a demonstration of its ease of use, particularly if one is familiar with, and has a predilection towards using **ggplot2** on a regular basis.

Keywords: plotting software, ternary diagrams, R, **ggplot2**.

1. Introduction

ggtern is a package for the statistical language R (R Core Team 2018), born out of the need to create ternary diagrams to an equivalent (superior) standard of the well established **ggplot2** package (Wickham 2009). Ternary diagrams, not being addressed by the default **ggplot2** implementation, are one of the tools held within the standard toolbox at the disposal of practitioners of several fields such as mineralogy (Flemming 2000), metallurgy & materials science (Baker 1992), politics (Katz and King 1999) as well as other physical sciences, where appropriate.

ggplot2 is an implementation of The Grammar of Graphics (Wilkinson 2005) and rigorously enforces Wilkinson's series of grammatical rules for creating perceivable graphs, therefore, by using **ggplot2** as a basis for the present work, we inherit this compelling philosophy, perhaps, from time-to-time, allowing the user to create graphical poetry (Wickham 2010).



(a) Sample Points. (b) Constant Y.

Figure 1: Primitive ternary diagrams, containing (a) three (3) individual points, illustrating how compositions are interpreted, and, (b) paths of constant Y concentration in increments of 20%. Additionally, in these figures, we demonstrate both (a) anticlockwise and (b) clockwise axis precession directions. Finally, the plot rotation feature can be seen in (b), which has been rotated by 60° counterclockwise.

Under peer-review, **ggtern** has already been utilized in various publications and fields, including for example, works by Witte, Bradley, Enright, and Muljo (2015), Milani, Ghiselli, Pecci, Maurizii, and Passamonti (2015); Wenger, Buzgariu, and Galliot (2016); Kalenitchenko *et al.* (2016); Chu, Ma, Prince, Antony, Seferovic, and Aagaard (2017) and Nesbitt *et al.* (2017).

2. Basics of ternary diagrams

For the readers that are not particularly familiar with ternary diagrams, they are a form of graphical data representation where the total of the respective variables sum to unity (100%), forming what is known as a 'simplex'. Using this approach is common for the handling of compositional data, for example, we could say that a liquid sugar/salt solution contains 90%H_2O, 5%NaCl and 5%$C_{12}H_{22}O_{11}$ – such a composition could be represented using the methodology described in the present work.

Ternary diagrams are a special case, where apparently three (3) variables have been projected onto a 2D surface, and this is only made possible since the 3rd component is *entirely* dependent on the first two (2), in essence, it is *not a variable at all*. This is intuitive since if we have an arbitrary ternary composition made up of, say, 25% A and 25% B, with a balancing amount of C, then this third component can *only* represent 50% of the overall composition – any value less than 50% would suggest that something is missing, and anything above 50% is simply nonsensical unless we are referring to relative ratios, in which case we know that these proportions must sum to unity. This type of system is referred to as an Aichison simplex, the likes of which can be enforced via the **compositions** package (van den Boogaart, Tolosana, and Bren 2018).

By the above logic, simplexes comprised of two (2) components (binary composition) have one (1) variable, and can be represented on a 'line', and simplexes in four (4) components (quaternary compositions) have three (3) variables, and can be described in 3D space using a tetrahedron. Taking this to the general case, if we were able to visually represent a simplex made up of n components, then we would be able to represent this in $(n-1)$D space.

The most common and accepted form of ternary diagram, is one which is rendered on an equilateral 'triangle'. A point somewhere within this triangular region indicates an individual composition ($a \cdot X + b \cdot Y + c \cdot Z = 100\%$), such that the three apexes represent $100\%X, 100\%Y$ and $100\%Z$ respectively. The centroid of the triangle indicates equal amounts of X, Y and Z. In reference to Figure 1a, three (3) points have been plotted to demonstrate reading of composition, namely, point A, representing the point of equal concentration between the X, Y and Z species, point B, having concentration (10%, 20% and 70%), and finally point C, having concentration (50%, 30% and 20%).

If we note that within the Cartesian coordinate system, the path of constant x ($x = k$) is a line which is parallel to the y axis having an x-intercept of k, similarly, the path of constant y ($y = j$) is the line which is parallel to the x axis having a y-intercept of j, then, it is intuitive to understand that for ternary diagrams, the path which is parallel to $Y - Z$ indicates compositions which have constant amounts of X, and the paths parallel to $X - Z$, and $X - Y$ represent constant amounts of Y and Z respectively. To illustrate this, paths of constant Y concentration, at 20% increments, have been plotted in Figure 1b.

Points outside the triangular region are *'mathematically sound'*, however in reality are non-sensical when dealing with physical compositions, implying that one or more of the simplex components have negative concentrations. Sometimes however, negative concentrations can be of value, say for example in Figure 3 where the text annotation in the lower-right hand corner has been positioned this way.

Ternary plots are limited in the sense that they are constricted to simplex problems using three compositional variables. Higher orders can be visualized using ternary 'slices', but this becomes arduous for more than four compositional variables. The obvious problem with ternary diagrams, is the counter-intuitive nature of reading the information, which is why reading ternary figures forms part of educational syllabus in many tertiary institutions within fields that regularly use such techniques, such as materials science and geology for instance – consider for instance the AMS handbook (Baker 1992), which as a reference book contains hundreds of compositions represented using ternary diagrams.

3. Features

ggtern is an open-source package for the statistical computational language R, which extends the functionality of **ggplot2**, best described by the following set of features:

- Creation of a *new* (ternary) coordinate system, which behind the scenes includes all the necessary transformations and inverse transformations between the Cartesian and ternary spaces.
- Implementation of a dedicated clipping mask geometry, which permits fine control of its exact point of placement (final layer as default, if not specified). See for example Figure 3.

- Necessary modifications to the specific plot assembly and rendering routines to account for axes that cannot be defined within columns or rows in a rectangular grid.
- Adaptation of the hierarchical theme-element structure as the building-blocks for customization of plot appearance. Indeed the present work includes over fifty (50) *new* theme elements specific to the ternary plot.
- Given the previous item, extensions to the existing base themes, permitting seamless use of both **ggplot2** and **ggtern** figures in close proximity to each other, as per Figure 2.
- Also included are four (4) *new* preset themes, which are unique to **ggtern**. For examples of these, please refer to Figures 4a to 4d.
- Incorporation of a mechanism to prevent the attempted use of geometries, stats or positional adjustments that would otherwise be *nonsensical* for ternary diagrams (e.g., violin plots or bar charts etc.).
- Adaptation of the base geometries (point, path, line, segment, smooth, density 2d, text, labels, annotations etc.) for compatibility with the ternary coordinate system.
- Adaptation of some of the base positional adjustment functions (identity, nudge or jitter) for compatibility with the ternary coordinate system.
- Creation of several *new* geometries, such as ternary error bars, isoproportion lines, crosshairs, interpolation models and confidence regions using isometric log-ratio transformation (Egozcue, Pawlowsky-Glahn, Mateu-Figueras, and Barcelo-Vidal 2003) and the Mahalanobis distance (Mahalanobis 1936). Additionally, **ggtern** includes modified text and label geometries, which permit convenient positioning based on fractional coordinates relative to the limits of the panel viewport, these last two geometries function equally in **ggplot2**.
- Creation of several *new* stats (confidence, density, interpolate), to accompany the respective geometries mentioned in the previous point.
- Introduction of a mechanism to permit rotation of the plotting panel to an arbitrary angle.
- A specific convenience function, `tern_limits(...)`, for specifying a limiting region to focus (zoom), minimizing user confusion (and potentially numeric degeneracy) when trying to specify both starting and finishing points for three (3) axes which together must satisfy the conditions of an Aichison simplex, see for example, Figure 3.
- Finally, creation of a large number of additional convenience functions for rapidly turning *on* or *off* specific features without having to nullify individual theme elements, such as `theme_showarrows()`, `theme_nogrid()`, `theme_clockwise()`, `theme_noticks()`, `theme_ticksinside()` and `theme_hidemask()` etc.

In order to implement the above, a number of design issues had to be overcome, by far the most difficult issue was in finding a means to modify existing geometries designed for 2D Cartesian coordinate systems, and permit them to function within an environment that requires an additional (spatial) aesthetic mapping (i.e., the `z` variable).

This was achieved by introducing the prototype member '`required_aes`' as a variable to the *coordinate system*, as additional to only being defined within any given geom or stat. Specifically, **ggplot2** makes no consideration on what is required for a particular geometry to remain valid within a potentially higher-dimensional coordinate system.

We can demonstrate this in action, via attempting to plot a ternary diagram with a mapping which we *know* to be invalid. Ordinarily, this would have been perfectly acceptable, since `geom_point` only stipulates `x` and `y` as being mandatory:

```
R> dfError <- data.frame(x = 1, y = 1)
R> ggtern(data = dfError, mapping = aes(x = x, y = y)) + geom_point()

Error: CoordTern requires the following missing aesthetics (tlr->xy) : z
```

Another design consideration, was the need to *overload* several of the internal **ggplot2** construction routines in preference to the seemingly obvious alternative of *subclassing* and creating dedicated (or modified) plot construction routines for the subclass. The former was required since, there is no difference between the following two methods, in turn making it difficult to implement the subclass pattern.

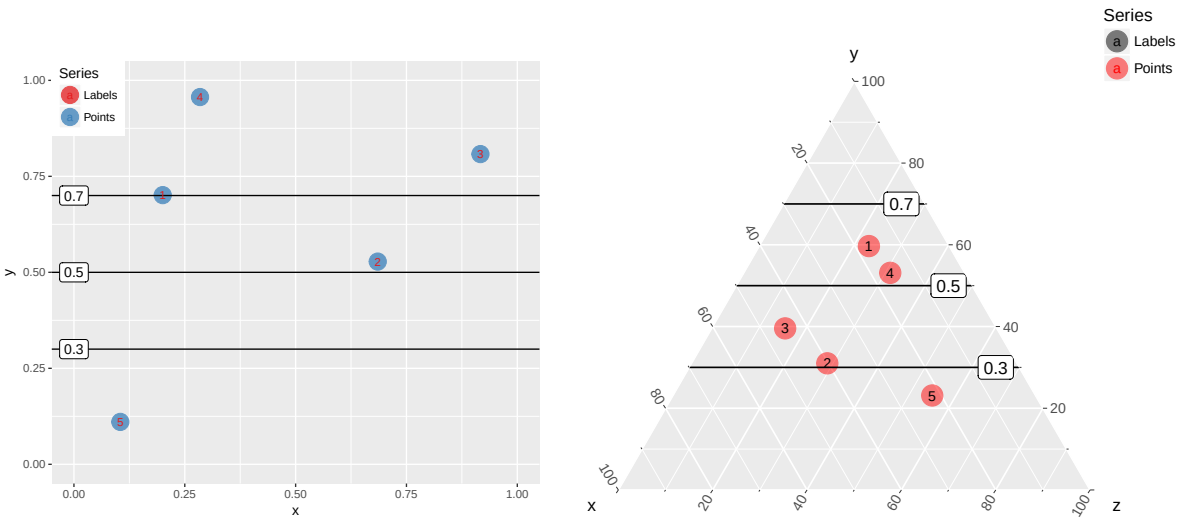
```
R> # (1) Difficult To Enforce The Desired Class/Subclass
R> ggplot() + coord_tern()
R> # (2) Easy to Enforce Desired Class/Subclass
R> ggtern()
```

Furthermore, **ggplot2**, even with its new mechanism for extensibility (post version 2.0), does *not* provide a means for users to create their own theme elements, and since the functioning of **ggtern** is inextricably reliant on many new theme elements any construction procedure that is called within **ggplot2** where theme element verification and inheritance resolution takes place was required to be overloaded so that a local copy of the element tree could be used.

One of the other design considerations, introduced from **ggtern** ($>= 2.0$), was the introduction of a clipping mask. **ggplot2** does not utilize clipping masks since the panel viewport operates in a *de-facto* manner with this regard, meaning that outliers to the panel viewport simply do not get rendered. **ggtern** on the other hand renders within a triangular shaped region that resides within a rectangular viewport, so, when the axis limits are reduced, data mappings can, and often do, lie outside of the triangular region – these need to be either deleted, masked, or shown (if the user so wishes). To accomplish this, a *dedicated* clipping mask geometry, `geom_mask()`, was created, and is added by default as the final layer, unless the user has explicitly added one earlier as part of their plot construction routine. To observe this in action, please refer to Figure 3.

4. Basic usage

ggtern is available from the Comprehensive R Archive Network (CRAN) at https://CRAN.R-project.org/package=ggtern. Like all packages available from CRAN where all exported functions must be documented, **ggtern** is no different and an extensive user manual can be freely downloaded (Hamilton 2018). In the following section, beginning with the coordinate system, we will demonstrate **ggtern**'s most basic usage and syntaxes.



(a) **ggplot2**. (b) **ggtern**.

Figure 2: The 'vanilla' output from the (a) **ggplot2** vs. (b) **ggtern** packages, using random data. Internally, the data is scaled, forming an Aitchison simplex, so that the parts sum to unity for each individual composition.

Typically, **ggplot2** utilizes Cartesian coordinates, which expects `x` and `y` mappings between the plot directives and the data, demonstrated via the snippet immediately below and rendered within Figure 2a.

```
R> library("ggplot2")
R> n <- 5
R> set.seed(n)
R> constants <- c(0.3, 0.5, 0.7)
R> dfA <- data.frame(ix = c(1:n), x = runif(n), y = runif(n))
R> f2a <- ggplot(data = dfA, mapping = aes(x = x, y = y)) +
+    coord_fixed(ratio = 0.5 * tan(pi / 3)) +
+    scale_x_continuous(limits = c(0, 1)) +
+    scale_y_continuous(limits = c(0, 1)) +
+    scale_color_brewer(palette = "Set1") +
+    geom_point(mapping = aes(color = "Points"), size = 6, alpha = 0.75) +
+    geom_hline(yintercept = constants) +
+    geom_text(mapping = aes(label = ix, color = "Labels"), size = 3) +
+    geom_label(data = data.frame(x = 0, y = constants),
+      mapping = aes(label = y)) + labs(color = "Series") +
+    theme_legend_position("topleft")
R> print(f2a)
```

By comparison, **ggtern** has its own constructor and its own coordinate system that expects an additional spatial mapping, which we have assigned as the `z` variable, demonstrated via the snippet below producing Figure 2b.

```
R> library("ggtern")
R> dfBase <- cbind(dfA, data.frame(z = runif(n), id = "Facet Label"))
R> dfLabs <- data.frame(y = constants, x = 0.05, z = 1 - constants - 0.05)
R> base <- ggtern(data = dfBase, mapping = aes(x = x, y = y, z = z)) +
+    geom_point(mapping = aes(color = "Points"), size = 6, alpha = 0.5) +
+    geom_text(mapping = aes(label = ix, color = "Labels"), size = 3) +
+    scale_color_manual(values = c("black", "red")) +
+    labs(color = "Series")
R> f2b <- base + geom_Tline(Tintercept = constants) +
+    geom_label(data = dfLabs, mapping = aes(label = y)) +
+    theme_legend_position("topright")
R> print(f2b)
```

It is often desirable to focus on a narrower compositional range – this can either be done manually via each of the `scale_*_continuous(...)` functions via the `limits` argument (`* = T, L` or `R`), or, alternatively via the convenience function `limit_tern(...)`, which requires only the user to specify the axis maxima ($T_{\max}, L_{\max}$ & $R_{\max}$), sufficient to in-turn solve for the associated minima ($T_{\min}, L_{\min}$ & $R_{\min}$) using linear algebra – a process which greatly reduces the burden on the end user to provide valid limits. The system of linear equations and solution is easily defined as follows:

$$T_{\max} + L_{\min} + R_{\min} = 1 \tag{1}$$
$$T_{\min} + L_{\max} + R_{\min} = 1 \tag{2}$$
$$T_{\min} + L_{\min} + R_{\max} = 1 \tag{3}$$

Taking Equations 1, 2 and 3, these can be converted to a system of linear equations that can be solved for the minima (given prior knowledge of the maxima), according to:

$$\begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix} \begin{bmatrix} T_{\min} \\ L_{\min} \\ R_{\min} \end{bmatrix} = \begin{bmatrix} 1 - T_{\max} \\ 1 - L_{\max} \\ 1 - R_{\max} \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} T_{\min} \\ L_{\min} \\ R_{\min} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{bmatrix}^{-1} \begin{bmatrix} 1 - T_{\max} \\ 1 - L_{\max} \\ 1 - R_{\max} \end{bmatrix} \tag{5}$$

Use of the `limit_tern(...)` function has been demonstrated within Figure 3, where we also highlight the clipping mask, and the `geom_*_isoprop(...)` geometry, for producing lines of isoproportionality. The code for producing this figure can be found immediately below, note the explicit placement of the `geom_mask` geometry, above the `geom_Risoprop` and below the `annotate` layers:

```
R> dfIsoprop <- data.frame(value = c(0.7, 0.5, 0.3))
R> f3 <- base + limit_tern(T = 0.7, L = 0.7, R = 0.7) +
+    geom_Risoprop(data = dfIsoprop,
+      mapping = aes(value = value, linetype = factor(value))) +
+    geom_mask() +
+    annotate(geom = "text", x = 0.775, y = -0.250, z = 0.475,
```

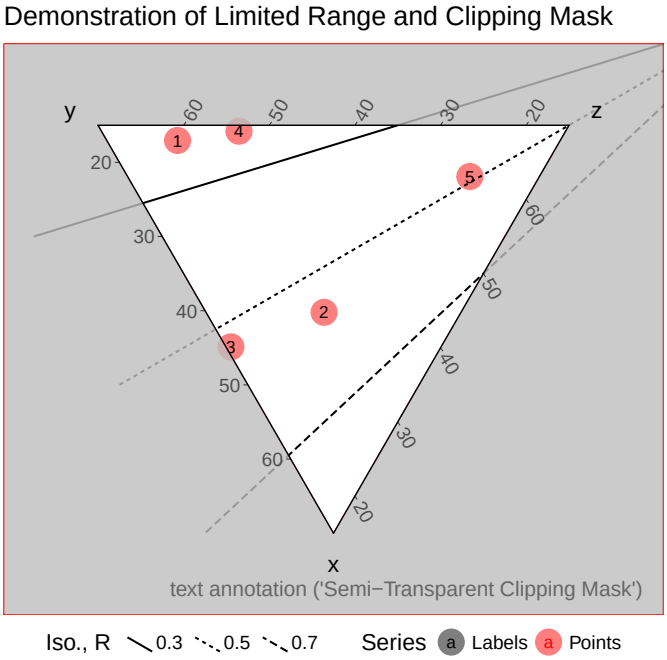


Figure 3: Demonstration of the `limit_tern(...)` convenience function for restricting axis ranges, which has also been rotated (60° rotation), and has the clipping mask highlighted via semi-transparency, using specific theme-element customization.

```
+      label = "text annotation ("Semi-Transparent Clipping Mask")",
+      color = "grey40", hjust = 1.0) + theme_classic() +
+    theme(legend.position = "bottom") +
+    theme(tern.plot.background  = element_rect(color = "red",
+      fill = alpha("gray", 0.8)),
+      tern.panel.background = element_rect(color = "red", size = 0.5),
+      tern.axis.line = element_blank()) + theme_rotate(degrees = 60) +
+    labs(title = "Demonstration of Limited Range and Clipping Mask",
+      linetype = "Iso., R")
R> print(f3)
```

It is often desirable to display arrows along each of the axes (ref. Figures 4b and 4c) in order to make reading these diagrams more intuitive. Some of the themes apply these arrows by default, and others do not. The standard `theme_gray()` function does not, although this can be easily switched back on via the `theme_showarrows()` theme modifier function.

ggtern comes with four (4) new template *'themes'* for the user to choose from, as well as a theme customization procedure that takes advantage of the theme-element inheritance structure defined by **ggplot2**. In total, there are more than fifty (50) new theme elements specific to **ggtern** in-and-around the three (3) new axes, and, although the user is free to alter any of these new elements from scratch (for the truly 'bespoke' appearance), as a matter of convenience it is often easier to use one of the base templates as a suitable starting point. These templates have been demonstrated within Figure 4.

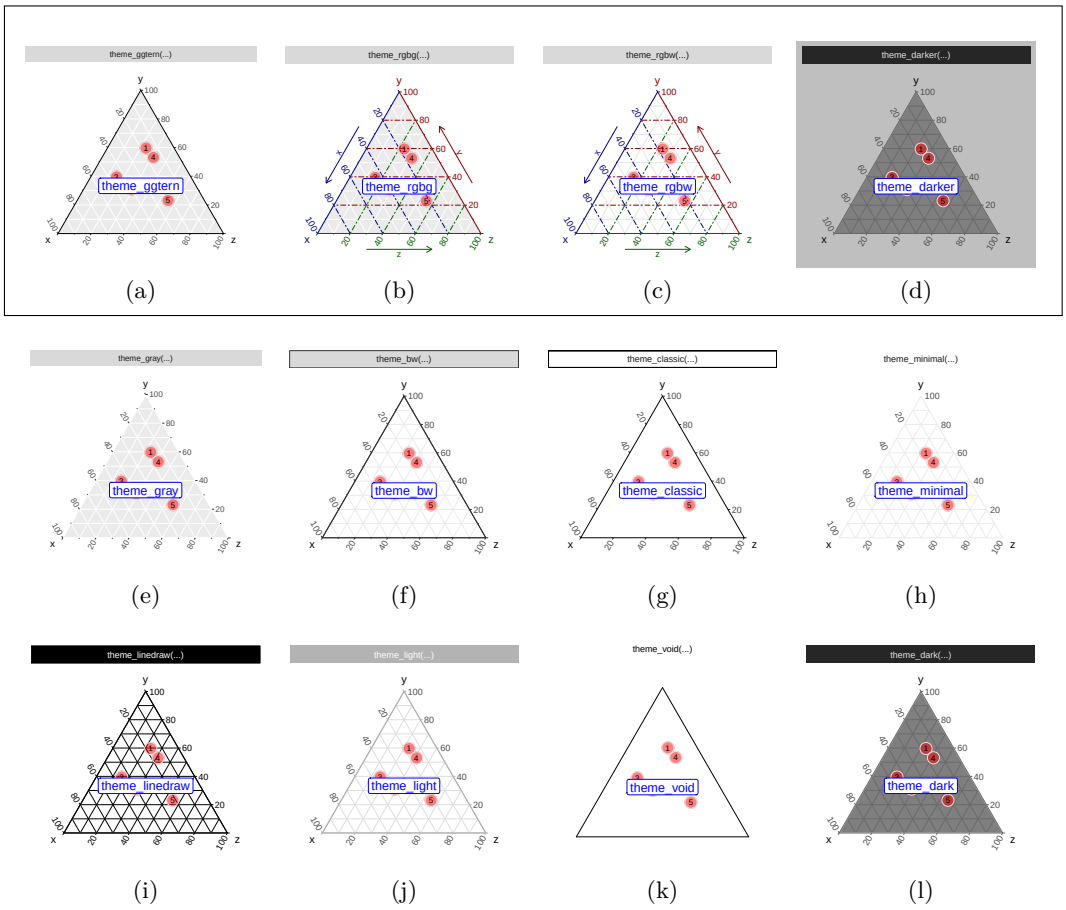


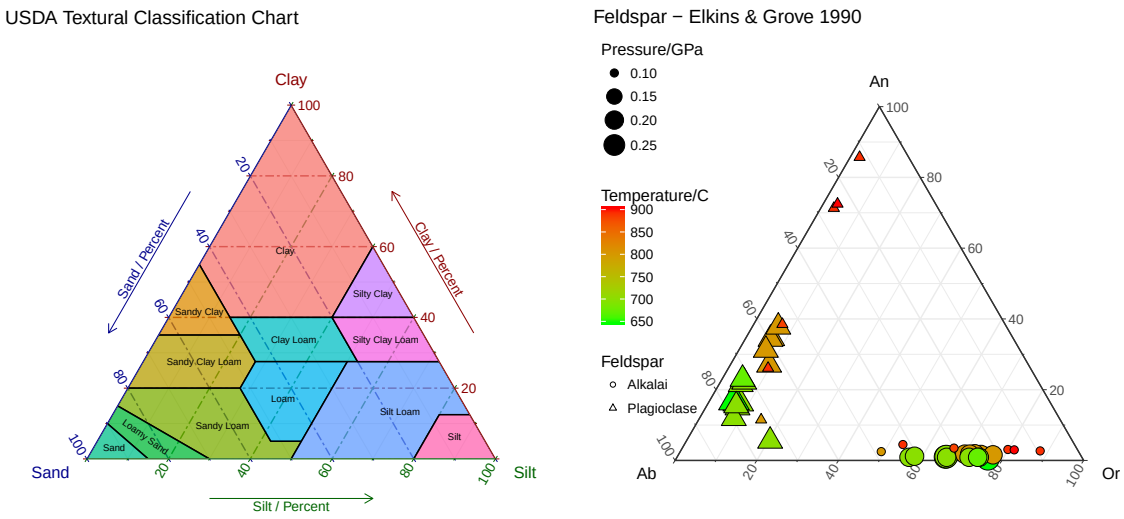
Figure 4: Themes available to **ggtern**, namely, `theme_X()`, where `X` is: (a) `ggtern`, (b) `rgbg`, (c) `rgbw`, (d) `darker`, (e) `gray`, (f) `bw`, (g) `classic`, (h) `minimal`, (i) `linedraw`, (j) `light`, (k) `void` and (l) `dark`. With the exception of the themes bound by the 'red box' above, i.e., (a), (b), (c) and (d), which are unique to **ggtern**, the remaining themes have been designed to mimic the default themes available within **ggplot2**. Also demonstrated above, the use of a new `annotate(...)` function, masking the **ggplot2** original, ensuring compatibility with an additional spatial mapping.

5. Examples

In the following examples, we replicate the United States Department of Agriculture (USDA) soil classification chart, and present various forms of Elkins and Groves Feldspar data, representing physical properties, confidence levels and temperature iso-contours. The results of the subsequent four (4) code snippets have been collectively rendered in Figures 5 and 6.

5.1. USDA textural soil classification chart

USDA textural soil classification chart can be produced (ref. Figure 5a) using the following code snippet; this can be compared with Figure 1 in the publication by Cosby, Hornberger, Clapp, and Ginn (1984):



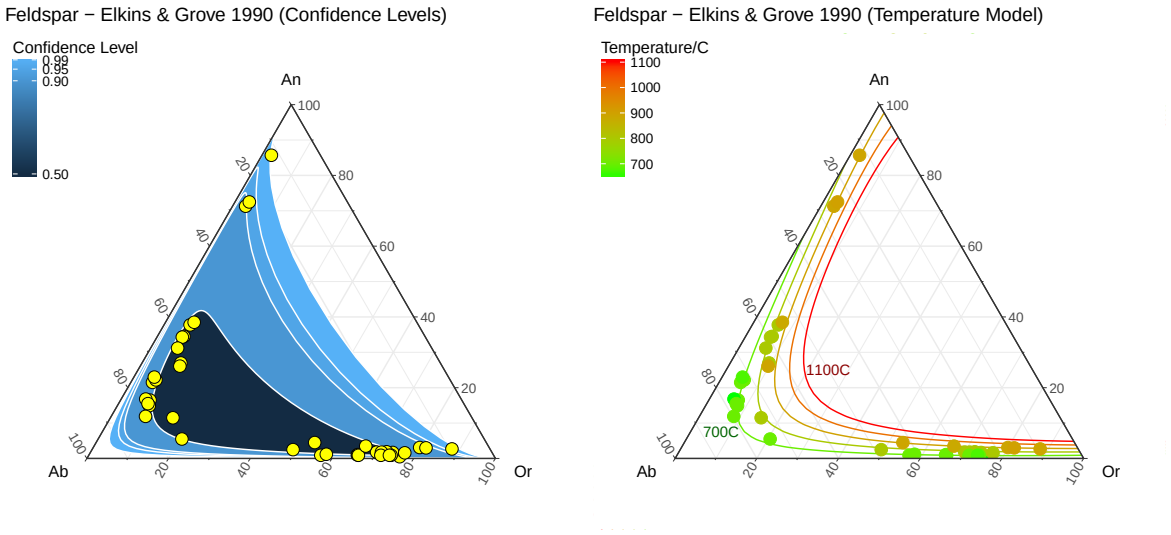
(a) (b)

Figure 5: First series of examples, including (a) USDA soil classification, and an interpretation of Elkins and Grove (1990) Feldspar data, namely, (b) mappings of properties to size, fill and shape. Please note in (b) the explicit mask placement below points and subsequent layers which has permitted the point geometry to spill over the perimeter of the plotting region.

```
R> data("USDA", package = "ggtern")
R> dfLabels <- plyr::ddply(USDA, "Label", function(df) {
+    label <- as.character(df$Label[ 1 ])
+    df$Angle <- switch(label, "Loamy Sand" = -35, 0)
+    colMeans(df[setdiff(colnames(df), "Label")])
+  })
R> f5a <- ggtern(data = USDA, mapping = aes(x = Sand, y = Clay, z = Silt)) +
+    geom_polygon(mapping = aes(fill = Label),
+      alpha = 0.75, size = 0.5, color = "black") +
+    geom_text(data = dfLabels, mapping = aes(label = Label, angle = Angle),
       size = 2.5) + theme_rgbw() + theme_showsecondary() +
+    theme_showarrows() + custom_percent("Percent") +
+    guides(color = "none", fill = "none") +
+    labs(title = "USDA Textural Classification Chart",
+      fill = "Textural Class", color = "Textural Class")
R> print(f5a)
```

5.2. Elkins and Grove properties

Elkins and Grove Feldspar data (Elkins and Grove 1990), with mappings of the various properties to shape, size and color has been produced (ref. Figure 5b) using the following code snippet:



(a) (b)

Figure 6: Second series of examples including interpretations of Elkins and Grove (1990) Feldspar data including (a) confidence regions calculated using the Mahalanobis distance and (b) basic isothermal model via 3rd order polynomial, which has been mapped to color.

```
R> data("Feldspar", package = "ggtern")
R> Feldspar <- Feldspar[ with(Feldspar, order(-P.Gpa)), ]
R> f5b <- ggtern(data = Feldspar,
+    mapping = aes(x = Ab, y = An, z = Or)) +
+    geom_point(mapping = aes(fill = T.C, size = P.Gpa, shape = Feldspar)) +
+    scale_shape_manual(values = c(21, 24)) +
+    scale_size_continuous(range = c(2.5, 7.5)) +
+    scale_fill_gradient(low = "green", high = "red") + theme_bw()  +
+    theme(tern.panel.mask.show = FALSE, legend.position = c(0, 1),
+      legend.justification = c(0, 1), legend.box.just = "left",
+      legend.background = element_rect(fill = "transparent")) +
+    labs(title = "Feldspar - Elkins & Grove 1990",
+      size = "Pressure/GPa", fill = "Temperature/C")
R> print(f5b)
```

5.3. Elkins and Grove confidence regions

Elkins and Grove Feldspar data with confidence levels calculated by the Mahalanobis distance and mapped to color can be produced (ref. Figure 6a) using the following code snippet:

```
R> confidenceBreaks <- c(0.5, 0.9, 0.95, 0.99)
R> f5c <- ggtern(data = Feldspar, mapping = aes(x = Ab, y = An, z = Or)) +
+    stat_confidence_tern(geom = "polygon", mapping = aes(fill = ..level..),
```

```
+      colour = "white", breaks = confidenceBreaks) + geom_mask() +
+    geom_point(colour = "black", fill = "yellow", shape = 21, size = 4) +
+    scale_fill_gradient(breaks = confidenceBreaks) + theme_bw() +
+    theme(legend.position = c(0, 1), legend.justification = c(0, 1),
+      legend.box.just = "left") +
+    labs(title = "Feldspar - Elkins & Grove 1990 (Confidence Levels)",
+      fill = "Confidence Level")
R> print(f5c)
```

5.4. Elkins and Grove temperature contours

Elkins and Grove Feldspar data with a temperature model at 700, 800, 900, 1000 and 1100°C mapped to color has been produced (ref. Figure 6b) using the code immediately below which can be compared with Figure 3 in the paper by Fuhrman and Lindsley (1988).

```
R> temperatureBreaks <- seq(from = 700, to = 1100, by = 100)
R> f5d <- ggtern(data = Feldspar, mapping = aes(x = Ab, y = An, z = Or)) +
+    geom_interpolate_tern(mapping = aes(color = ..level.., value = T.C),
+      breaks = temperatureBreaks, base = "identity", method = "glm",
+      formula = value ~ poly(x, y, degree = 3)) +
+    annotate(geom = "text", x = 0.50, y = 0.250, z = 0.250,
+      label = "1100C", color = "darkred" ) +
+    annotate(geom = "text", x = 0.85, y = 0.075, z = 0.075,
+      label = "700C", color = "darkgreen") +
+    geom_point(mapping = aes(colour = T.C), size = 4) + theme_bw() +
+    theme_legend_position("topleft") +
+    scale_colour_gradient(low = "green", high = "red") +
+    labs(title = "Feldspar - Elkins & Grove 1990 (Temperature Model)",
+      colour = "Temperature/C")
R> print(f5d)
```

6. Discussion

In the following section, we provide a summary of the existing packages available across different architectures and programming languages, and, outline the plans for possible future development.

6.1. Other available packages

Ternary diagrams are not exclusive to **ggtern**, they have been used extensively within the literature, and can be processed by many plotting libraries. Some of which have been included in Table 1, and, listed as follows: MATLAB (The MathWorks Inc. 2017) package (**ternPlot**), Python (Van Rossum *et al.* 2011) libraries (**python-ternary** and **Vuesz**), R packages (**vcd** and **compositions**), graphical user interfaces (GUIs: **CSpace**, **Analysen Tetraeder**, **Trinity** and **Origin**), and finally spreadsheet implementations (**TernPlot** and **Tri-Plot**).

	Name	OS	Type	Remarks
1	**TernPlot**	All	Spreadsheet	Points only.
2	**Tri-Plot**	All	Spreadsheet	Points only.
3	**CSpace**	Win	GUI	Can do quaternary, dynamic graphics.
4	**Analysen Tetraeder**	Mac	GUI	Can do quaternary.
5	**Trinity**	Mac	GUI	Points only.
6	**Origin**	Win	GUI	Can do 3D scatter and 3D surfaces.
7	**ternPlot**	All	MATLAB package	Can do 3D scatter and 3D surfaces.
8	**vcd**	All	R package	Rendered via R **grid** graphics.
9	**compositions**	All	R package	Rendered via R base graphics.
10	**python-ternary**	All	Python library	Several advanced geometries.
11	**Vuesz**	All	Python library	Several advanced geometries.

Table 1: Survey of other software packages available, across multiple languages/platforms and operating systems. References for the above are as follows: (1) Marshall (1996), (2) Graham and Midgley (2000), (3) Torres-Roldan *et al.* (2000), (4) Schmitz (2013), (5) Appel (2016), (6) Edwards (2002), (7) Sandrock (2016), (8) Meyer *et al.* (2006), (9) van den Boogaart *et al.* (2018), (10) Harper *et al.* (2015), (11) Sanders (2018).

In summary, the *main* point of difference between **ggtern** and these existing software packages, is that **ggtern** adheres to the philosophy of Wilkinson (2005), which provides a *strict* regiment for mapping numerical/categorical data, to the various visual cues, such as: *color, fill, shape, size, alpha* etc., across the various available geometries *points, paths, segments, polygons* etc.

Out of the alternatives to **ggtern**, the Python libraries provide the most compelling alternative, providing advanced geometries such as trajectories, hexbins, heatmaps and the ability to map variables to shape, size and color in a similar fashion to **ggplot2**, however, to the best knowledge of the authors, in terms of customization of appearance, *none* of the existing packages provide flexibility even remotely close to that of **ggtern** – almost every aspect of the plots appearance can be tailored in some manner using the hierarchical structure of the various theme elements, inherited and extended from **ggplot2**.

In an identical manner to **ggplot2**, plots can be constructed in a piecewise fashion, adding layers (geometries), annotations, statistics and the like, to generate extremely complex, yet clear, data representations. The clarity of the representations, is due to the adherence to Wilkinson (2005) philosophy, enforced via extension of **ggplot2** acting as a *'tried-and-tested'* foundation. Having said the above, a number of the above packages permit 3D surfaces, or are able to handle quaternary compositions, both features being unavailable within **ggtern** at the present moment.

Additionally, it is worth noting that with the release of **ggplot2** (>= 2.0.0), rebuilt partially with the ambition of providing an extensible framework, **ggtern** can be further extended (say for custom geometries) according to the needs of the user. This feature is beyond the scope of this document.

6.2. Further possible extensions

In future revisions of this software, it is very likely that we will continue to extend **ggplot2**, insomuch as introducing additional geometries presently not available to **ggtern**. One such

geometry that immediately comes to mind, is a 'ternary' version of the standard 'heatmap', which will require a compositional ternary binning statistic. There are also a few other geometries, such as the `geom_rug` and `geom_hex`, which would both work on a ternary diagram, with probably limited applicability, and would be added for the sake of completeness. Several parties have expressed an interest/or requested 3D ternary diagrams (i.e., to represent a dependent variable, say surface contours in 3D), and/or quaternary diagrams, however such an implementation would go against the philosophy of **ggplot2** and is unlikely ever to be implemented. We have also received a number of requests to introduce functionality for producing piper diagrams (Güler, Thyne, McCray, and Turner 2002), which will take considerable work and subject to major revision.

7. Conclusions

We have demonstrated that **ggplot2** can be used as a basis for producing specialized plotting packages and, in this case, a package has been developed specifically for the production of high quality ternary diagrams. **ggtern** inherits the vast majority of the strengths of **ggplot2**, including (amongst many others) its plot-construction syntax and strict adherence to the rules governed by *The Grammar of Graphics* (Wilkinson 2005). In order to produce **ggtern**, it was necessary to overcome a number of design issues, primarily finding a means to modify existing geometries designed for a 2D Cartesian coordinate system, and permit them to function in an environment that requires an additional spatial aesthetic mapping. Considerations have been made for the case where users attempt to plot nonsensical geometries relative to the ternary coordinate system. As a package, **ggtern** could best be described as being made up by the new coordinate system, modified existing and new geometries, new theme elements, new convenience functions and a slightly modified plot-construction and rendering routine.

Acknowledgments

The authors would like to acknowledge the Australian Research Council (ARC) for partly funding of this work via the ARC Centre of Excellence for Design in Light Metals (CE0561574) and ARC Discovery grants scheme (DP120102863). Furthermore, the authors would like to thank Hadley Wickham and the other developers of **ggplot2**, for their excellent package.

References

Appel P (2016). "**Trinity** – Ternary Diagram." Christian-Albrechts-Universität zu Kiel. Accessed 2016-08-20, URL `http://www.ifg.uni-kiel.de/339.html`.

Baker H (ed.) (1992). *ASM Handbook: Alloy Phase Diagrams*, volume 3. ASM International, Materials Park, Ohio.

Chu DM, Ma J, Prince AL, Antony KM, Seferovic MD, Aagaard KM (2017). "Maturation of the Infant Microbiome Community Structure and Function across Multiple Body Sites and in Relation to Mode of Delivery." *Nature Medicine*, **23**, 314–326. `doi:10.1038/nm.4272`.

Cosby BJ, Hornberger GM, Clapp RB, Ginn TR (1984). "A Statistical Exploration of the Relationships of Soil Moisture Characteristics to the Physical Properties of Soils." *Water Resources Research*, **20**(6), 682–690. doi:10.1029/wr020i006p00682.

Edwards PM (2002). "**Origin** 7.0: Scientific Graphing and Data Analysis Software." *Journal of Chemical Information and Computer Sciences*, **42**(5), 1270–1271.

Egozcue JJ, Pawlowsky-Glahn V, Mateu-Figueras G, Barcelo-Vidal C (2003). "Isometric Logratio Transformations for Compositional Data Analysis." *Mathematical Geology*, **35**(3), 279–300. doi:10.1023/a:1023818214614.

Elkins LT, Grove TL (1990). "Ternary Feldspar Experiments and Thermodynamic Models." *American Mineralogist*, **75**(5-6), 544–559.

Flemming BW (2000). "A Revised Textural Classification of Gravel-Free Muddy Sediments on the Basis of Ternary Diagrams." *Continental Shelf Research*, **20**(10), 1125–1137. doi:10.1016/s0278-4343(00)00015-7.

Fuhrman ML, Lindsley DH (1988). "Ternary-Feldspar Modeling and Thermometry." *American Mineralogist*, **73**(3–4), 201–215.

Graham DJ, Midgley NG (2000). "Graphical Representation of Particle Shape Using Triangular Diagrams: An **Excel** Spreadsheet Method." *Earth Surface Processes and Landforms*, **25**(13), 1473–1478.

Güler C, Thyne GD, McCray JE, Turner KA (2002). "Evaluation of Graphical and Multivariate Statistical Methods for Classification of Water Chemistry Data." *Hydrogeology Journal*, **10**(4), 455–474.

Hamilton N (2018). ***ggtern**: An Extension to **ggplot2**, for the Creation of Ternary Diagrams*. R package version 3.1.0, URL https://CRAN.R-project.org/package=ggtern.

Harper M, Weinstein B, Simon C, chebee7i, Swanson-Hysell N, Badger TG, Greco M, Zuidhof G (2015). "**python-ternary**: Ternary Plots in Python." doi:10.5281/zenodo.34938.

Kalenitchenko D, Dupraz M, Le Bris N, Petetin C, Rose C, West N, Galand PE (2016). "Ecological Succession Leads to Chemosynthesis in Mats Colonizing Wood in Sea Water." *The ISME Journal*, pp. 2246–2258. doi:10.1038/ismej.2016.12.

Katz JN, King G (1999). "A Statistical Model for Multiparty Electoral Data." *American Political Science Review*, **93**(1), 15–32. doi:10.2307/2585758.

Mahalanobis PC (1936). "On the Generalized Distance in Statistics." *Proceedings of the National Institute of Sciences (Calcutta)*, **2**, 49–55.

Marshall D (1996). "**Ternplot**: An **Excel** Spreadsheet for Ternary Diagrams." *Computers & Geosciences*, **22**(6), 697–699. doi:10.1016/0098-3004(96)00012-x.

Meyer D, Zeileis A, Hornik K (2006). "The Strucplot Framework: Visualizing Multi-Way Contingency Tables with **vcd**." *Journal of Statistical Software*, **17**(3), 1–48. doi:10.18637/jss.v017.i03.

Milani L, Ghiselli F, Pecci A, Maurizii MG, Passamonti M (2015). "The Expression of a Novel Mitochondrially-Encoded Gene in Gonadic Precursors May Drive Paternal Inheritance of Mitochondria." *PloS One*, **10**(9), e0137468. doi:10.1371/journal.pone.0137468.

Nesbitt SJ, Butler RJ, Ezcurra MD, Barrett PM, Stocker MR, Angielczyk KD, Smith RM, Sidor CA, Niedźwiedzki G, Sennikov AG, *et al.* (2017). "The Earliest Bird-Line Archosaurs and the Assembly of the Dinosaur Body Plan." *Nature*, **544**, 484–487. doi:10.1038/nature22037.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/.

Sanders J (2018). "**Veusz** – A Scientific Plotting Package." URL https://veusz.github.io/.

Sandrock C (2016). "**ternplot**: Plots Ternary Phase Data on a Ternary Phase Diagram." MATLAB File Exchange. Version 1.1, Updated 2016-06-25, URL https://au.mathworks.com/matlabcentral/fileexchange/2299-alchemyst-ternplot.

Schmitz C (2013). "**Analysen Tetraeder** 1.3." Accessed 2018-10-30, URL http://www.monkeybreadsoftware.de/Analysen-Tetraeder/.

The MathWorks Inc (2017). *MATLAB – The Language of Technical Computing, Version R2017b*. Natick. URL http://www.mathworks.com/products/matlab/.

Torres-Roldan RL, Garcia-Casco A, Garcia-Sanchez PA (2000). "**CSpace**: An Integrated Workplace for the Graphical and Algebraic Analysis of Phase Assemblages on 32-Bit Wintel Platforms." *Computers & Geosciences*, **26**(7), 779–793.

van den Boogaart KG, Tolosana R, Bren M (2018). ***compositions***: *Compositional Data Analysis*. R package version 1.40-2, URL https://CRAN.R-project.org/package=compositions.

Van Rossum G, *et al.* (2011). *Python Programming Language*. URL https://www.python.org/.

Wenger Y, Buzgariu W, Galliot B (2016). "Loss of Neurogenesis in Hydra Leads to Compensatory Regulation of Neurogenic and Neurotransmission Genes in Epithelial Cells." *Philosophical Transactions of the Royal Society B*, **371**(1685), 20150040. doi:10.1098/rstb.2015.0040.

Wickham H (2009). ***ggplot2***: *Elegant Graphics for Data Analysis*. Springer-Verlag. URL http://ggplot2.org/.

Wickham H (2010). "A Layered Grammar of Graphics." *Journal of Computational and Graphical Statistics*, **19**(1), 3–28. doi:10.1198/jcgs.2009.07098.

Wilkinson L (2005). *The Grammar of Graphics*. Statistics and Computing, 2nd edition. Springer-Verlag.

Witte S, Bradley A, Enright AJ, Muljo SA (2015). "High-Density P300 Enhancers Control Cell State Transitions." *BMC Genomics*, **16**(1), 903. doi:10.1186/s12864-015-1905-6.

Affiliation:

Nicholas E. Hamilton
School of Materials Science and Engineering
The University of New South Wales (UNSW Sydney)
Sydney, NSW 2052, Australia
E-mail: nick@ggtern.com
URL: http://ggtern.com/

Journal of Statistical Software http://www.jstatsoft.org/
published by the Foundation for Open Access Statistics http://www.foastat.org/
December 2018, Volume 87, Code Snippet 3 *Submitted:* 2016-07-23
doi:10.18637/jss.v087.c03 *Accepted:* 2017-10-31