eulerr: Area-Proportional Euler Diagrams with Ellipses

Bachelor Thesis

Johan Larsson[1] and Peter Gustafsson[1]

[1]*Lund University*

2017-08-15

Contents

1 Introduction

Relationships between groups and sets are the focus in many scientific disciplines. In biomedicine, for instance, the overlap between sets of genes may be of interest. In epidemiology, the interactions between diseases are frequently studied. Likewise, the social sciences are often involved in studying demographics where the commonalities between various groups are scrutinized.

Visualizations of such relationships are central to understanding them. The commonest visualization is Venn diagrams. Venn diagrams are a subset of Euler diagrams, which were originally proposed by Leonard Euler (1707–1783) [1]. Whereas Venn diagrams require that all 2^n possible intersection are present – even if they are empty – Euler diagrams stipulate no such requirement.

Euler diagrams can be area-proportional, which is the case for non-Venn diagrams. Area-proportional diagrams are most easily understood as their interpretations do not depend on any numbers. This paper will henceforth be concerned only with such designs and the term *Euler diagram* will be considered to refer to area-proportional diagrams.

Euler diagrams may be fashioned with any conceivable closed shape.
Solutions have been developed for triangles [2], rectangles [2], ellipses [3], smooth curves, polygons [2], and circles [2,4,5]. The latter is most common, and appears to be preferred for optimal readability [citation]. Yet circles do not always lend themselves to accurate representations. Consider, for instance the combination

$$\begin{gathered} A = B = C = 2, \\ A \cap B = A \cap C = B \cap C = 1, \text{ and} \\ A \cap B \cap C = 0. \end{gathered}$$

There is no way to visualize this relationship perfectly with circles (Figure 1).

```r
library(eulerr)
library(latticeExtra)
p1 <- plot(euler(c(A = 2, B = 2, C = 2, "A&B" = 1, "A&C" = 1, "B&C" = 1)))
p2 <- plot(euler(c(A = 2, B = 2, C = 2, "A&B" = 1, "A&C" = 1, "B&C" = 1),
                    shape = "ellipse"))
c(p1, p2)
```

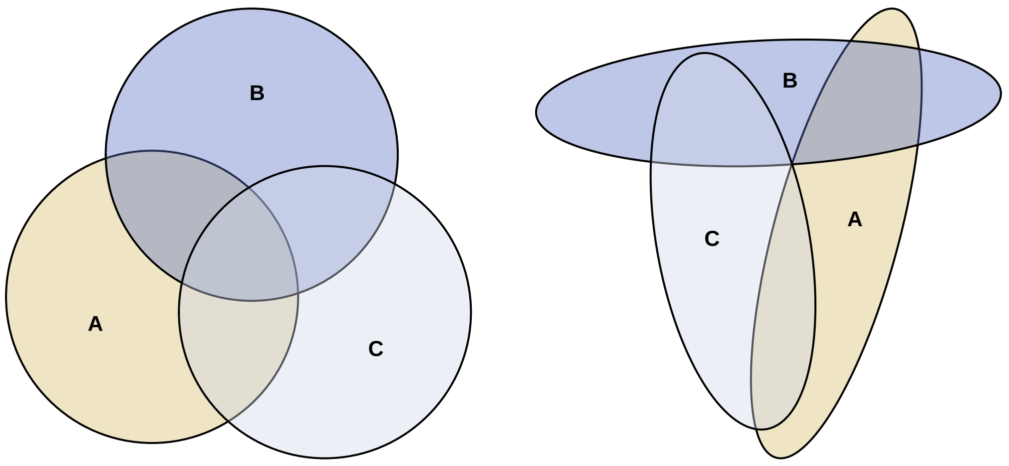

Figure 1. A set relationship depicted with circles and ellipses.

For four intersecting sets, circular Euler diagrams are in fact impossible since they require $2^4 = 16$ regions but four intersecting circles yield at most 14 unique intersections. This is not, however, the case with ellipses being that they may intersect in up to 4, rather than 2, points. And because ellipses can be rotated, they can succeed where circles fail (Figure 1).

Elliptical Euler diagrams have been successfully implemented in the **eulerAPE** software [3] but only for three sets that all need to intersect. It is implemented in

2 Method

2.1 Areas of overlapping ellipses

2.2 Visualization

3 Results

3.1 Consistency

3.2 Accuracy

3.3 Performance

4 Discussion

eulerr relies on an extensive machinery to turn user input into a pretty Euler diagram. Little, or in fact none, of this requires any tinkering from the user. To make that happen, however, **eulerr** needs to make several well-formed decisions about the design of the diagram on behalf of the user, which is not a trivial task.

This document outlines the implementation of **eulerr** from input to output. It is designed to an up-to-date paper on the innards of the program and is therefore likely to evolve as time goes on.

5 Input

The main function of **eulerr** is `euler()`. To start with, we need input in the form of

- a named numeric vector, such as `c(A = 10, B = 5, "A&B" = 3)`, where ampersands define disjoint set combinations or unions, depending on the argument `input`,
- a `data.frame` or `matrix` of logicals or binary indices where each row denotes the set relationships of
- either a single observation

```r
matrix(sample(c(TRUE, FALSE), 12, replace = TRUE),
       ncol = 3,
       dimnames = list(NULL, c("A", "B", "C")))
```

```
##          A     B     C
## [1,] FALSE FALSE FALSE
## [2,]  TRUE FALSE  TRUE
## [3,] FALSE  TRUE FALSE
## [4,]  TRUE FALSE  TRUE
```

- or of a unique set combination if a numeric vector is supplied to the argument `weights`,

```r
matrix(c(TRUE, FALSE, FALSE,
         TRUE, TRUE, FALSE,
         FALSE, FALSE, TRUE),
       ncol = 3,
       dimnames = list(NULL, c("A", "B", "C")))
```

```
##          A     B     C
## [1,]  TRUE  TRUE FALSE
## [2,] FALSE  TRUE FALSE
## [3,] FALSE FALSE  TRUE
```

- a `table` (max 3 dimensions),

```r
as.table(apply(Titanic, 2:4, sum))
```

```
## , , Survived = No
##
##         Age
## Sex      Child Adult
##   Male      35  1329
##   Female    17   109
##
## , , Survived = Yes
##
##         Age
## Sex      Child Adult
##   Male      29   338
##   Female    28   316
```

- or a list of sample spaces, such as

```r
list(A = c("x", "xy", "xyz"),
     B = c("xy"),
     C = c("x", "xyz"))
```

```
## $A
## [1] "x"   "xy"  "xyz"
##
```

```
## $B
## [1] "xy"
##
## $C
## [1] "x"   "xyz"
```

If the `data.frame` or `matrix` form is used, the user additionally has the option to split the data set by a factor and compute separate euler diagrams for each split. This is accomplished by supplying a factor variable to the by arguments (see the documentation in `?base::by`).

6 Pre-processing

eulerr organizes the input of the user into a matrix of binary indexes, which in **R** is represented as a matrix of logicals. For a three set configuration, this looks like this,

```
library(eulerr)
eulerr:::bit_indexr(3)
```

```
##       [,1]  [,2]  [,3]
## [1,]  TRUE FALSE FALSE
## [2,] FALSE  TRUE FALSE
## [3,] FALSE FALSE  TRUE
## [4,]  TRUE  TRUE FALSE
## [5,]  TRUE FALSE  TRUE
## [6,] FALSE  TRUE  TRUE
## [7,]  TRUE  TRUE  TRUE
```

and is accompanied by a vector of the *disjoint* areas of the set combinations.

To provide a starting configuration, we work exclusively with circles and, given these areas, we figure out the required pairwise distance between the sets to achieve a circle–circle overlap that matches the set intersection between the sets. We do this numerically, using the formula for a circle–circle overlap,

$$A = r_1^2 \arccos\left(\frac{d^2 + r_1^2 - r_2^2}{2dr_1}\right) + r_2^2 \arccos\left(\frac{d^2 + r_2^2 - r_1^2}{2dr_2}\right) - \frac{1}{2}\sqrt{(-d + r_1 + r_2)(d + r_1 - r_2)(d - r_1 + r_2)(d + r_1 + r_2)}, \quad (1)$$

where r_1 and r_2 are the radii of the first and second circles respectively and d the distance between the circles.

r_1 and r_2 are known but because d is not, we approximate it using one-dimensional numerical optimization. Our loss function is the squared difference between A and the desired overlap, which we then optimize using **R**'s `optimize()`, which is a "combination of golden section search and successive parabolic interpolation".

```r
r1 <- 0.7 #radius of set 1
r2 <- 0.9 #radius of set 2
overlap <- 1 #area of overlap

stats::optimize(eulerr:::discdisc, #computes the squared loss
                interval = c(abs(r1 - r2), sum(r1, r2)),
                r1 = r1,
                r2 = r2,
                overlap = overlap)
```

```
## $minimum
## [1] 0.634
##
## $objective
## [1] 2.98e-10
```

```r
# minimum is our required distance
```

Now that we have the distances, we can proceed to the next step: computing an initial configuration.

7 Initial configuration

Our initial layout can be setup in a number of ways; **eulerr** uses one of the methods from Fredrickson's venn.js, which features a constrained version of multi-dimensional scaling (MDS) based on that of Wilkinson's **R** package venneuler [4]. **venneuler** tries to place disjoint and subset exactly neck-in-neck and at the exact midpoint of the set respectively. However, since we are indifferent about where in the space outside (or respectively inside) the sets are placed, that behavior becomes problematic since it might interfere with locations of other sets that need to occupy some of that space.

The MDS algorithm from **venn.js** circumvents this by assigning a loss and gradient of 0 when, for instance, the set relationsships *and* the candidate ellipses are disjoint. Then, to optimize the pairwise relationsships between sets, **eulerr** uses the following loss and gradient functions.

$$\text{loss} = \sum_i \sum_j \begin{cases} 0 & \text{disjoint}(i,j) \\ 0 & \text{subset}(i,j) \\ ((X_i - X_j)^T(X_i - X_j) - D_{ij}^2)^2 & \text{otherwise} \end{cases}$$

$$\nabla f(X_i) = \sum_j \begin{cases} \vec{0} & \text{disjoint}(i,j) \\ \vec{0} & \text{subset}(i,j) \\ 4((X_i - X_j)^T(X_i - X_j) - D_{ij}^2)(X_i - X_j) & \text{otherwise} \end{cases}$$

Source: Better Venn Diagrams by Ben Fredrickson, which includes a nice interactive demonstration.

Fredrickson uses the *Polak–Ribière Conjugate Gradient Method* to optimize the initial layout. In our experience this method occasionally ends up in local minima, which is why we have opted to use `nlm()` from the **R** core package `stats`, which is a translation from FORTRAN code developed by [6] and uses a mixture of algorithms (Newton and Quasi-Newton).

This initial configuration will work perfectly for any 1–2 set combinations and as well as possible with 3 sets if we use circles but for all other combinations there is usually a need to fine tune the configuration.

8 Final configuration

In order to finalize the configuration we need to be able to compute the areas of the overlaps of the sets, which as it turns out, is *not* trivial. In fact, most of methods rely on approximations of the areas by, for instance, quad-tree binning (**venneuler**) or polygon intersections (**VennMaster** [5]). These methods yield reasonable estimates but, given that the computation may have to run for a vast number of iterations, are usually prohibitive in terms of performance.

venn.js and **eulerAPE** both, however, use exact algorithms. Based on the fact that any intersection of ellipses can be represented as a convex polygon with elliptical segments on the fringes, it is possible to arrive at exact area calculations.

8.1 Intersections

Finding the areas of the overlaps exactly requires that we first know the points at which the different ellipses intersect. **eulerr**'s approach to this is based on a method outlined by [7]. **eulerr** owes significant debt to the **R** package **RConics** [8], which has been tremendously helpful in developing and, especially, debugging the algorithm. Some parts of the code are in fact straight-up translations to C++ from the code in **RConics**.

The method is based in *projective geometry* (rather than euclidean). To find the intersection points, the algorithm first

- converts the two ellipses from canonical form to matrix notation. The canonical form of a rotated ellipse is given by

$$\frac{((x-h)\cos(\phi)+(y-k)\sin(\phi))^2}{a^2}+\frac{((x-h)\sin(A)-(y-k)\cos(\phi))^2}{b^2}=1,$$

  where *phi* is the counter-clockwise angle from the positive x axis to the semi-major axis *a*. *b* is the semi-minor axis whilst *(h, k)* is the center of the ellipse. This is then converted to the matrix form

$$E=\begin{bmatrix} A & B/2 & D/2 \\ B/2 & C & E/2 \\ D/2 & E/2 & F \end{bmatrix},$$

  which may be used to represent any conic. We then
- split one of the ellipses (conics) into a pencil of two lines, and subsequently
- intersect the remaining conic with these two lines, which will yield between 0 and 4 intersection points.

8.2 Areas

The next step is to calculate the area of overlap between all the possible combinations of ellipses. The solution to this was discovered, as far as I know, by Fredrickson who explains it beautifully in a blog post. It relies on finding all the intersection points between the currently examined sets that are also within these sets. It is then trivial to find the area of the convex polygon that these vertices make up. Finding the rest of the area, which is made up of the ellipse segments between subsequent points, requires a bit of trigonometry.

Here, we have used an algorithm from [9], which computes circle integral between the points on the ellipse minus the area of the triangle made up of the center of the ellipse:

$$A(\theta_0,\theta_1)=F(\theta_1)-F(\theta_1)-\frac{1}{2}|x_1y_0-x_0y_1|,$$

$$\text{where } F(\theta)=\frac{a}{b}\left[\theta-\arctan\left(\frac{(b-a)\sin 2\theta}{b+a+(b-a)\cos 2\theta}\right)\right]$$

As our loss function, we use the sum of squared differences between the disjoint set intersections and the areas we have computed and again use the `nlm()` optimizer to layout the set.

This optimization step is the bottleneck of the overall computations in terms of performance, being that we're optimizing over 5 parameters for every ellipse (or 3 in the case of circles) – nevertheless, we're profitting immensely from the implementation in the C++ programming language through **Rcpp** [10] and its plugin for the linear algebra library **Armadillo** [11] which ends up making the code much faster than the java-based **venneuler**.

9 Layout

Since the optimization steps are unconstrained, we run the risk of ending up with dispersed layouts. To fix this, we use the SKYLINE-BL rectangle packing algorithm [12] to pack the disjoint clusters of ellipses (in case there are any) into a heuristically chosen bin.

At the time of writing this algorithm is crudely implemented – for instance, it does not attempt to rotate the rectangles (boundaries for the ellipses) or attempt to use. Since we're dealing with a rather simple version of the rectangle packing problem, however, it seems to do the trick.

10 Output

Before we get to plotting the solution, it is useful to know how well the fit from **eulerr** matches the input. Sometimes euler diagrams are just not feasible, particular for combinations with many sets, in which case we should stop here and look for another design to visualize the set relationships.

It is not, however, obvious what it means for a euler diagram to "fit well". **venneuler** uses a metric called *stress*, which is defined as

$$\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} y_i^2}$$

where $\hat{y}_i$ is an ordinary least squares estimate from the regression of the fitted areas on the original areas that is being explored during optimization.

Meanwhile, **eulerAPE** [3] uses *diagError*:

$$\max_{i=1,2,\ldots,n}\left|\frac{y_i}{\sum y_i} - \frac{\hat{y}_i}{\sum \hat{y}_i}\right|$$

Both metrics are given the user after the diagram has been fit, together with a table of residuals.

```
combo <- c("A" = 1, "B" = 1, "C" = 1,
           "A&B" = 0.5, "A&C" = 0.5, "C&B" = 0.5)

fit1 <- euler(combo)
fit1
```

```
##       original fitted residuals region_error
## A          1.0  1.038    -0.038        0.021
## B          1.0  1.038    -0.038        0.021
## C          1.0  1.038    -0.038        0.021
## A&B        0.5  0.302     0.198        0.040
## A&C        0.5  0.302     0.198        0.040
## B&C        0.5  0.302     0.198        0.040
## A&B&C      0.0  0.247    -0.247        0.058
##
## diag_error:  0.058
## stress:      0.049
```

```
plot(fit1, counts = TRUE)
```

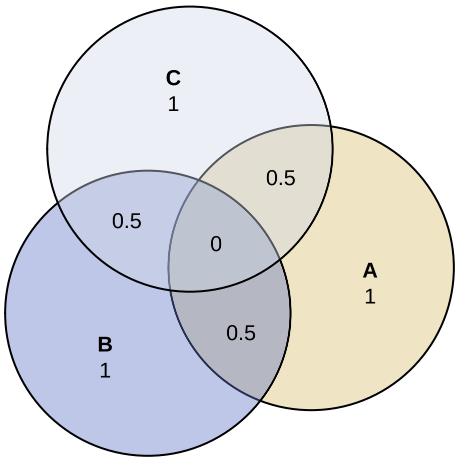

Figure 2. A plot with circles.

It is clear that this is not a good fit, which we can find out just by looking at the plot. This is a good example of when ellipses come in handy.

```
fit2 <- euler(combo, shape = "ellipse")
fit2
```

```
##        original fitted residuals region_error
## A           1.0    1.0         0            0
## B           1.0    1.0         0            0
## C           1.0    1.0         0            0
## A&B         0.5    0.5         0            0
## A&C         0.5    0.5         0            0
## B&C         0.5    0.5         0            0
## A&B&C       0.0    0.0         0            0
##
```

```
## diag_error:  0
## stress:      0
```

```
plot(fit2, counts = TRUE)
```

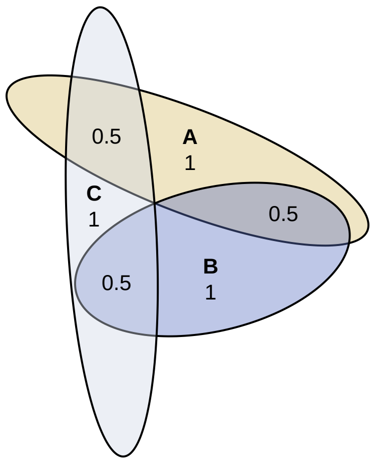

Figure 3. A plot with ellipses.

Much better.

11 Plotting

Let's face it: euler diagrams are naught without visualization. Here, **eulerr** interfaces the elegant Lattice graphics system [13] to grant the user extensive control over the output, and allow for facetted plots in case such a design was used in fitting the euler configuration.

11.1 Labelling

Most users will want to label their euler diagrams. One option is to simply add a legend

```
plot(euler(c(A = 2, B = 3, "A&B" = 1)), auto.key = TRUE)
```

but many will want to label their diagrams directly, perhaps with counts.

```
plot(euler(c(A = 2, B = 3, "A&B" = 1)), counts = TRUE)
```

In this case, layout out the diagram becomes considerably more involved. Finding a reasonable spot for the text inside the diagram only lends itself to an easy solution if the shape of the

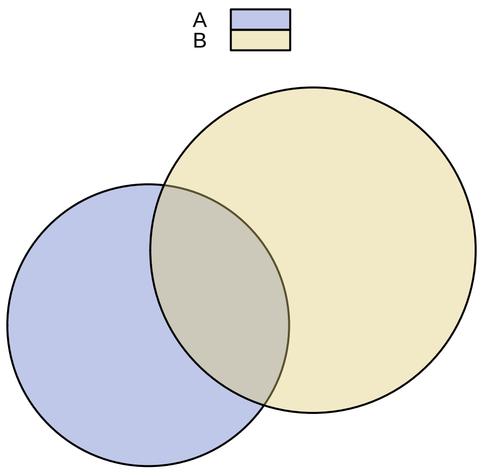

Figure 4. A simple plot.

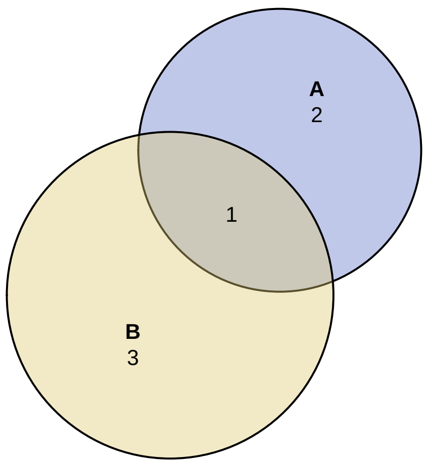

Figure 5. A plot with counts.

intersection has a center-of-gravity inside ellipse, in which case an average of some of the points might suffice. This is often not the case, however, and we need a better solution. Specifically, what we need is a method to find the point inside the circle overlap for the counts and circle complement to the intersection for our labels.

So far, we have not been able to derive at an analyitcal solution for finding a good point, or for that matter a reliable way of finding *any* point that is in the required intersection or complement. As is often the case, the next-best thing turns out to be a numerical one. First, we locate a point that is inside the required region by spreading points across one of the discs involed in the set combination. To spread points uniformly, we use *Vogel's method* [14,15]

$$\left(p_k = (\rho_k, \theta_k) = \left(r\sqrt{\frac{k}{n}}, \pi(3-\sqrt{5})(k-1)\right)\right)_{k=1}^{n},$$

which is actually based on the golden angle.

```r
n <- 500
seqn <- seq(1, n, 1)
theta <- seqn*pi*(3 - sqrt(5))
rad <- sqrt(seqn/n)
x <- rad*cos(theta)
y <- rad*sin(theta)

xyplot(x ~ y, aspect = "iso", pch = 20, xlab = "", ylab = "",
       col = 1,
       par.settings = list(axis.line = list(col = "transparent")),
       scales = list(draw = FALSE))
```

After this, we scale, translate, and rotate the points so that they fit the desired ellipse.

After we've spread our points throughout the ellipse and found one that matches our desired combination of ellipses/sets, we then proceed to optimize its position numerically. For this, we use version of the *Nelder–Mead Method* [16] which we've translated from Matlab code by [17] and customized for **eulerr** (in particular to make sure that the simplex does not escape the intersection boundaries since we for this problem *want* the local minimum).

11.2 Coloring

Per default, the ellipses are filled with colors. The default option is to use an adaptive scheme in which colors are chosen to provide a balance between dinstinctiveness, beauty, and consideration for the color deficient. The color palette has been generated from qualpalr (developed by the author), which automatically generates qualitative color palettes based on a model of color perception.

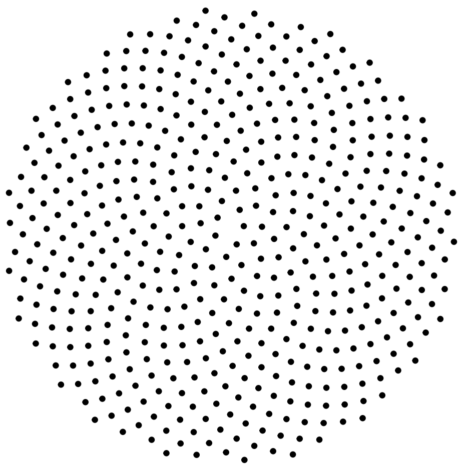

Figure 6. Spreading points on a disc with Vogel's method.

References

1. Leonhard Euler. Letters of Euler to a German Princess, on Different Subjects in Physics and Philosophy [Internet]. Murray; Highley; 1802 [cited 2017 Aug 14]. Available from: http://archive.org/details/letterseulertoaooeulegoog

2. Swinton J. Vennerable: Venn and Euler area-proportional diagrams [Internet]. Available from: https://github.com/js229/Vennerable

3. Micallef L, Rodgers P. eulerAPE: Drawing Area-Proportional 3-Venn Diagrams Using Ellipses. PLOS ONE. 201417ADJul;9(7):e101717.

4. Wilkinson L. Exact and Approximate Area-Proportional Circular Venn and Euler Diagrams. IEEE Transactions on Visualization and Computer Graphics. 2012 Feb;18(2):321–31.

5. Kestler HA, Müller A, Kraus JM, Buchholz M, Gress TM, Liu H, et al. VennMaster: Area-proportional Euler diagrams for functional GO analysis of microarrays. BMC Bioinformatics [Internet]. 2008 Jan [cited 2017 Aug 6];9:67. Available from: https://doi.org/10.1186/1471-2105-9-67

6. Schnabel RB, Koonatz JE, Weiss BE. A Modular System of Algorithms for Unconstrained Minimization. ACM Trans Math Softw [Internet]. 1985 Dec [cited 2017 Aug 6];11(4):419–40. Available from: http://doi.acm.org/10.1145/6187.6192

7. Richter-Gebert J. Perspectives on Projective Geometry: A Guided Tour Through Real and Complex Geometry. 1st ed. Berlin, Germany: Springer; 2011.

8. Huber E. RConics: Computations on Conics [Internet]. 2014. Available from: https://CRAN.R-project.org/package=RConics

9. Eberly D. The Area of Intersecting Ellipses [Internet]. Geometric Tools. 2016 [cited 2017 Aug 7]. Available from: https://www.geometrictools.com/Documentation/AreaIntersectingEllipses.pdf

10. Eddelbuettel D, François R. Rcpp: Seamless R and C++ integration. Journal of Statistical

Software [Internet]. 2011;40(8):1–18. Available from: http://www.jstatsoft.org/v40/i08/

11. Eddelbuettel D, Sanderson C. RcppArmadillo: Accelerating r with high-performance c++ linear algebra. Computational Statistics and Data Analysis [Internet]. 2014 March;71:1054–63. Available from: http://dx.doi.org/10.1016/j.csda.2013.02.005

12. Jylänki J. A Thousand Ways to Pack the Bin - A Practical Approach to Two-Dimensional Rectangle Bin Packing [Internet]. 2010 [cited 2017 Aug 7]. Available from: http://clb.demon.fi/files/RectangleBinPack.pdf

13. Sarkar D. Lattice: Multivariate Data Visualization with R [Internet]. New York, USA: Springer; 2008. (Use R!). Available from: http://www.springer.com/us/book/9780387759685

14. Arthur MK. Point Picking and Distributing on the Disc and Sphere [Internet]. Abedeen, USA: US Army Research Laboratory, Weapons; Materials Research Directorate; 2015 Jul p. 58. Report No.: ARL-TR-7333. Available from: www.dtic.mil/get-tr-doc/pdf?AD=ADA626479

15. Vogel H. A better way to construct the sunflower head. Mathematical Biosciences. 1979;44(3-4):179–89.

16. Nelder JA, Mead R. A Simplex Method for Function Minimization. The Computer Journal [Internet]. 1965 Jan;7(4):308–13. Available from: https://academic.oup.com/comjnl/article/7/4/308/354237/A-Simplex-Method-for-Function-Minimization

17. Kelley CT. Iterative Methods for Optimization. 1 edition. Philadelphia, USA: Society for Industrial; Applied Mathematics; 1999. (Frontiers in applied mathematics).