Data Viz Tutorial: Projections, Rotations, and Tours

Gradon Nicholls *PhD Student, University of Waterloo*

Acknowledgements

This tutorial relies on the `tourr` package and corresponding article (Wickham et al., 2011), as well as additional background from Chen et al. (2008) and Buja et al. (2005). I also make use of navigation graph examples from the course slides by Wayne Oldford. Finally, I use the `loon.tourr` package (Xu and Oldford, 2021) which implements the `tourr` methods in `loon` (Waddell and Oldford, 2023).

Pre-reqs

```
# need graph package for PairViz
if (!requireNamespace("graph", quietly = TRUE)){
    if (!require("BiocManager")) install.packages("BiocManager")
    BiocManager::install("graph")
}

# list packages to load
pkgs = c("tidyverse","tourr","loon","loon.data","loon.tourr","PairViz","ash")

# install pacman (if needed) and use to load packages
if (!require("pacman")) install.packages("pacman")
pacman::p_load(char=pkgs)
rm(pkgs)
```

1 Recall: Projections and Navigation Graphs

Three-dimensional data can be difficult to visualize on a two-dimensional screen. Moving beyond three-dimensions, it becomes impossible to plot data using our standard Cartesian coordinates methods.

One method for visualization of higher-dimensional data is to project the data onto a lower-dimensional space. However, there are infinitely many possible projections!

We could plot scatter plots of only two variables at a time. In fact, we can view these plots as projections onto the span of the two chosen variables. However, this could be masking clusters, dependencies, non-linear relationships, etc.

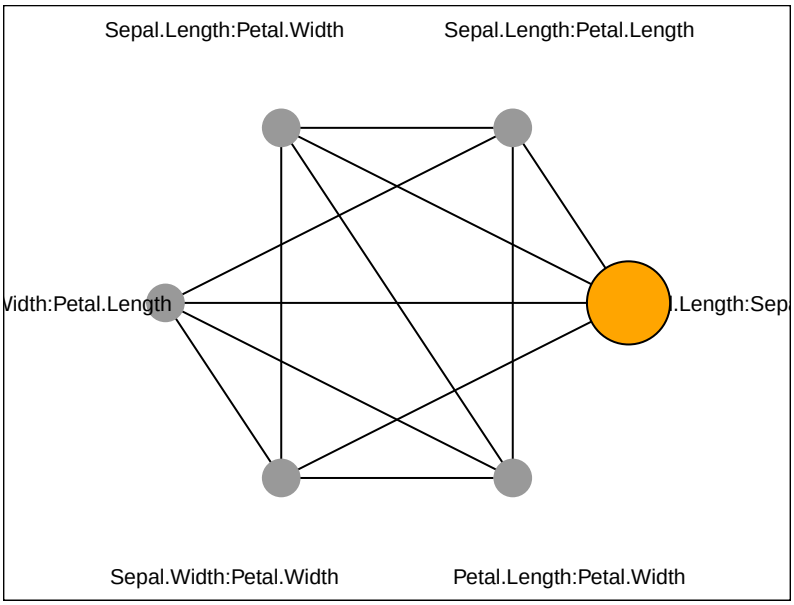
One option is to view a series of 2D projections which constitute a "path" through the higher-dimensional space. We have seen this already in this course when we used "navigation graphs".

1.1 Example: iris data

In the `iris` data we have 4 variables (dimensions): `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`. While there are infinitely-many possible projections from 4D space to 2D space, we can consider a finite number of paths:

- 3D paths: (x_i, x_j) to (x_i, x_k); $j \neq k$
- 4D paths: (x_i, x_j) to (x_k, x_l); none of i, j, l, l are equal

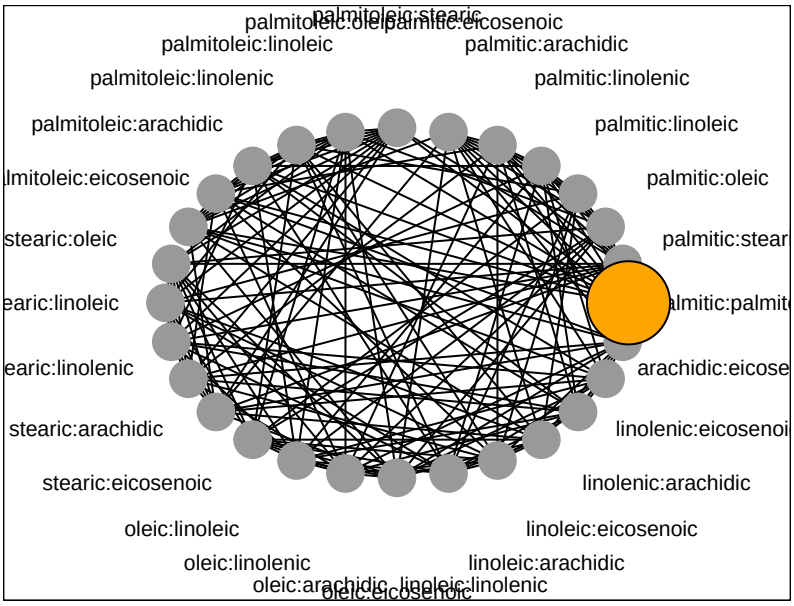
```
p = l_navgraph(
  data=iris[,1:4],
  color="grey70",
  linkingGroup="iris"
)
plot(p)
```



1.2 Example: olive data

In the olive data, we had 8 variables (dimensions), and saw how unwieldy the navigation graph starts to be.

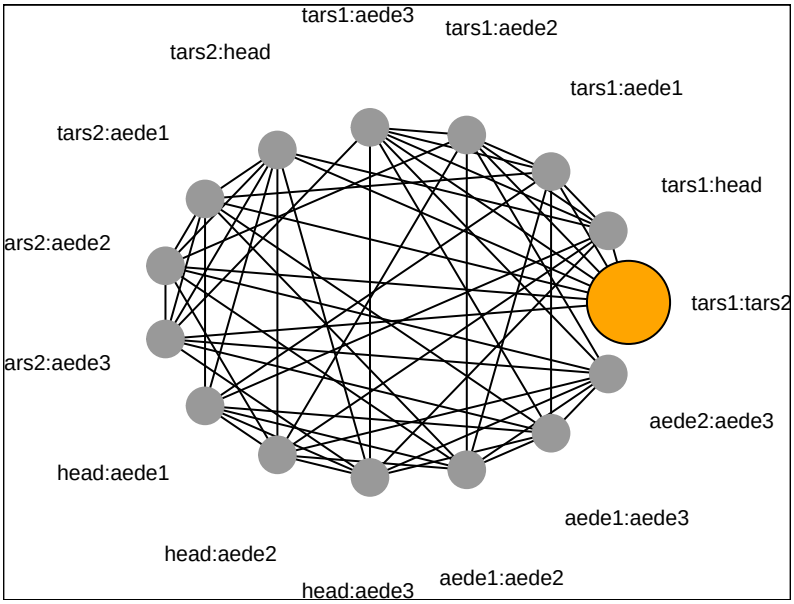
```
p = l_navgraph(
  data=olive[, -c(1,2)],
  color="grey70",
  linkingGroup="olive"
)
plot(p)
```



2 Rotations

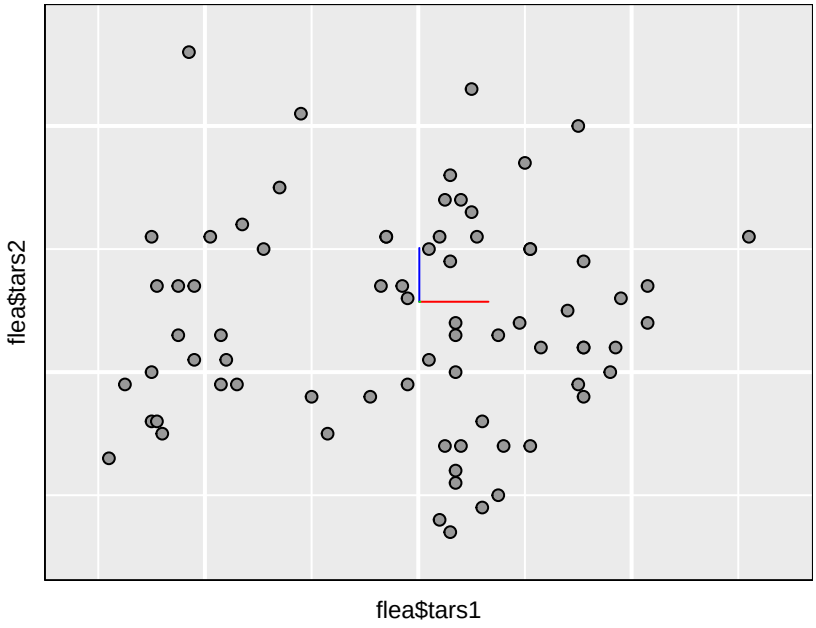
Consider the 6-dimensional `flea` dataset. How do we conduct the rotation from `tars1:tars2` to `head:aede1`?

```
p = l_navgraph(
  data=flea[, 1:6],
  color="grey70",
  linkingGroup="flea"
)
plot(p)
```



Rotation in 3 dimensions is intuitive. We imagine an axis of rotation about which the data points rotate in a circular way.

```
p = l_plot3D(
  flea$tars1,
  flea$tars2,
  flea$head
)
plot(p)
```



But what about in higher dimensions? The idea of an "axis of rotation" is no longer intuitive. Instead, consider that while we perceive 3D rotation as objects rotating in 3D space, we are in fact viewing a series of 2D projections of the 3D data. In higher dimensions, we take this same tact.

Let $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, x_6)^\intercal$ be a row of our 6-dimensional `flea` dataset. Then our objective is to conduct a rotation from the scatter plot of x_1, x_2 to the scatter plot of x_3, x_4. We denote a projection by $\mathbf{y} = F\mathbf{x}$. We represent each scatter plot by the following projections:

$$\mathbf{y}_{12} = F_{12}\mathbf{x} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix} \mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

$$\mathbf{y}_{34} = F_{34}\mathbf{x} = \begin{pmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix} \mathbf{x} = \begin{pmatrix} x_3 \\ x_4 \end{pmatrix}$$

How do we "rotate" or "interpolate" between these two projections? Consider:

$$\mathbf{y}(t) = F(t)\mathbf{x} = \begin{pmatrix} \cos(t) & 0 & \sin(t) & 0 & 0 & 0 \\ 0 & \cos(t) & 0 & \sin(t) & 0 & 0 \end{pmatrix} \mathbf{x} = \begin{pmatrix} \cos(t)x_1 + \sin(t)x_3 \\ \cos(t)x_2 + \sin(t)x_4 \end{pmatrix}$$

where $0 \leq t \leq \pi/2$. We see that $F(0) = F_{12}$ and $F(\pi/2) = F_{34}$. For other values of t, we have a series of interpolations. Notice that we can think of this as rotating 90 degrees ($\pi/2$ radians) from one projection to the other. I.e. the two projections are "perpendicular".

This is not the only path we could choose, but it has a few good properties:

- it is "smooth" (because sin and cos are smooth functions)
- it avoids "within-plane" rotation
- it is the "locally shortest" path between the projections–i.e. it is a "geodesic" path

Let's code and animate this:

```
# scale data to be mean 0, sd 1
X = t(scale(flea[,1:6]))

# initial and target projections
F_12 = matrix(0,nrow=2,ncol=6); F_12[1,1] = 1; F_12[2,2] = 1
 y_12 = F_12 %*% X
F_34 = matrix(0,nrow=2,ncol=6); F_34[1,3] = 1; F_34[2,4] = 1
 y_34 = F_34 %*% X

# smooth interpolation function
interpolation = function(t) {
  F_t = F_12 * cos(t) + F_34 * sin(t)
  F_t %*% X
}

# check that interpolation matches initial and target
all(abs(interpolation(0) - y_12) < 1e-15)
all(abs(interpolation(pi/2) - y_34) < 1e-15)

# animate inerpolations from 0 to pi/2
for (tau in seq(0,1,0.005)) {
  t = tau*pi/2
  y = interpolation(t)
  plot(
    x=y[1,],
    y=y[2,],
    xlim=c(-3,3),
    ylim=c(-3,3),
    xlab=paste0(round(cos(t),2),"x1 + ",round(sin(t),2),"x3"),
    ylab=paste0(round(cos(t),2),"x2 + ",round(sin(t),2),"x4")
  )
  Sys.sleep(0.01)
}
```

3 Tours

The objective of navigation graphs was to take a "tour" of the data by animating a smooth path (the edges) between any pair of scatter plots (the nodes). Recall that each scatter plot can be considered a projection from the higher dimensional space to the two-dimensional space spanned by the two scatter plot variables. In principle, there is no reason to constrain ourselves to these particular projections. In this section we consider visualizing higher-dimensional data using sequences of projections more generally, with emphasis on the *grand tour* and *projection pursuit* methods and their implementation in the `tourr` package.

We can break this problem into two steps:

1. the **generator** chooses a sequence of projections, and,
2. the **interpolator** smoothly interpolates from one projection to the next.

3.1 Generator

Sequences of projections can be chosen in one of several ways (Buja et al., 2005):

- **Manual Choice** or **Precomputed Choice**: We could choose the sequence of projections ourselves in real time, or we could choose a pre-defined sequence to animate. A navigation graph can be used to conduct either a manual tour–when we direct the path ourselves–or a precomputed tour–when we select a sequence of nodes in the graph and animate the path along the sequence.

- **Random Choice**: Rather than constrain ourselves to particular projections, we may wish to randomly sample from the set of all projections. This is referred to as the *grand tour*. In the `tourr` package, projections are generated by sampling from a multivariate normal distribution.

- **Data-driven Choice**: Manual/precomputed choice constrains us to a particular set of projections. Random choice allows us to see all projections, but only after sufficient time has passed. Data-driven choice is attractive because we actively search the entire set of projections for particularly "interesting" ones as determined by the data itself. These are referred to as *guided tours*.

3.2 Interpolator

Let d be the dimensionality of the data and $p \leq d$ be the dimensionality of the projection. Typically, $p < d$, but this need not be the case in theory. Often we will have $p = 2$ (i.e. two-dimensional scatter plots).

Assume the generator has chosen a (possibly countably infinite) sequence of projections $F_1, F_2, \ldots$ with dimensions $p \times d$. That is, each projection defines $y_i = F_i x$, where x is a data point (a $d \times 1$ vector), and y_i is the projected data point (a $p \times 1$ vector). The job of the interpolator is to come up with a smooth path between each pair of projections F_i, F_{i+1}. In other words, we are looking for $y_i(t) = G_i(t)x$ where $0 \leq t \leq 1$, $G_i(0) = F_{i-1}$, and $G_i(1) = F_i$.

To animate this, we break up time into small segments $0 = t_1 < t_2 < \cdots < t_m = 1$ and plot each $y_i(t_j)$ in sequence. The trick is coming up with a path $G_i(t)$ that is smooth, and does not appear to be speeding up or slowing down. We also want to avoid "within-plane spin"–that is we don't want our scatter plot to simply rotate clockwise or counter-clockwise. The method used by `tourr`, and described in more detail in Buja et al. (2005), is to decompose $G_i(t)$ into a sequence of rotations along the locally shortest paths–i.e. *geodesics*.

4 The `tourr` package

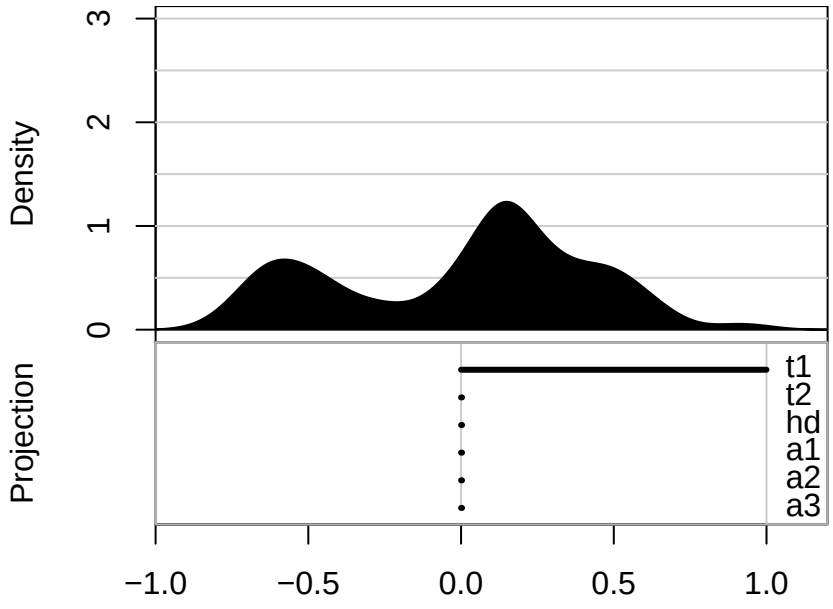
There are different variations of the "Grand Tour" algorithm. The one implemented in the `tourr` package is called the "geodesic random walk". The package defines a *tour* as consisting of three components: the data, a path (defined by the generator and interpolator), and a display method. Functions in the package follow this philosophy, as we will see.

4.1 Example: 1D Display

Here we see a tour with the following components:

1. Data: `flea`
2. Path: grand tour (1 dimension)
3. Display: density plot

```
animate(
  data = flea[, 1:6],
  tour_path = grand_tour(d = 1),
  display = display_dist(),
  max_frames = 0 # remove this to animate
)
```

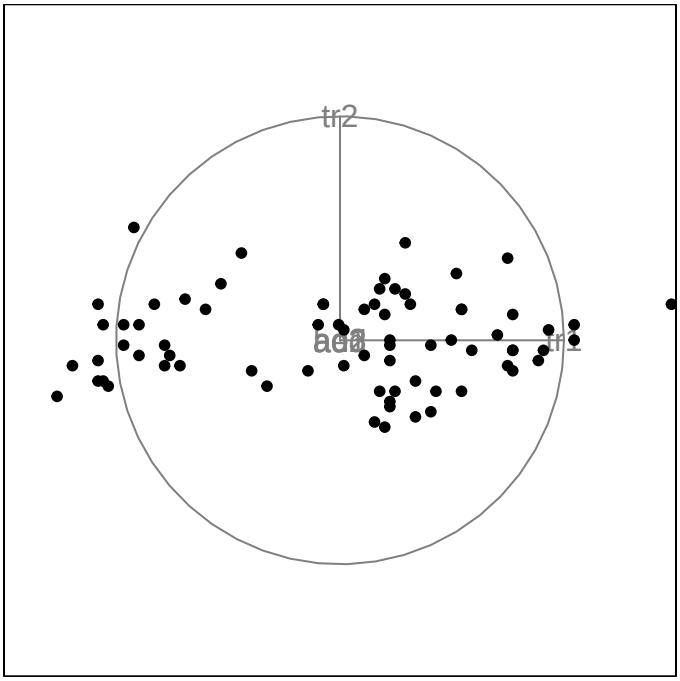


4.2 Example: 2D Display

Here we see a tour with the following components:

1. Data: `flea`
2. Path: grand tour (2 dimensions)
3. Display: scatter plot

```
animate(
  data = flea[, 1:6],
  tour_path = grand_tour(d = 2),
  display = display_xy(),
  max_frames = 0 # remove this to animate
)
```

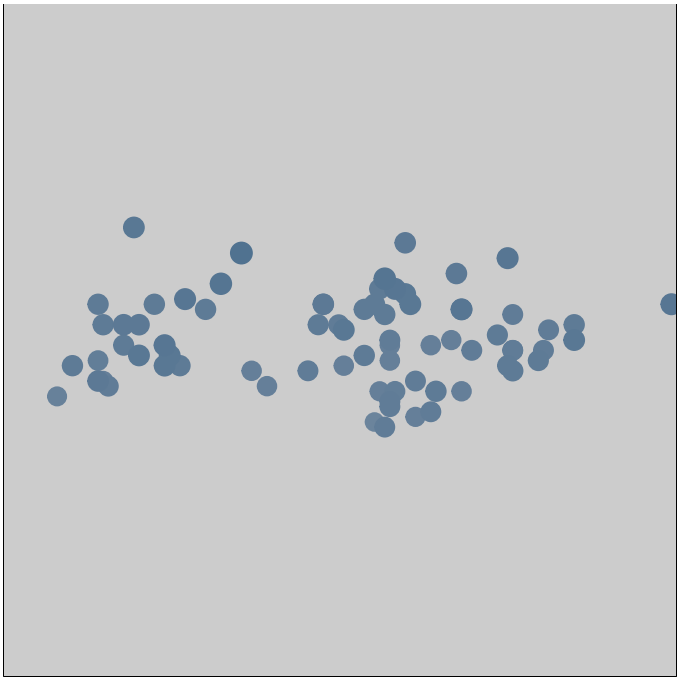


4.3 Example: 3D Display

Here we see a tour with the following components:

1. Data: `flea`
2. Path: grand tour (3 dimensions)
3. Display: depth plot

```
animate(
  data = flea[, 1:6],
  tour_path = grand_tour(d = 3),
  display = display_depth(),
  max_frames = 0 # remove this to animate
)
```



4.4 Guided Tours

For guided tours, we need to define what it means to be an "interesting" projection. Three methods implemented in `tourr`:

1. `cmass()`: searches for projections with points closer together
2. `holes()`: searches for projections with points further apart
3. `lda_pp(cl)`: searches for projections which separate groups defined by `cl`

We implement these by using a `guided_tour` as our `tour_path`:

```
flea_std = scale(flea[,1:6])

# 1D
animate(
  data = flea_std,
  tour_path = guided_tour(
    index_f = cmass(),
    d = 1
  ),
  display = display_dist()
)
animate(
  data = flea_std,
  tour_path = guided_tour(
    index_f = holes(),
    d = 1
  ),
  display = display_dist()
)

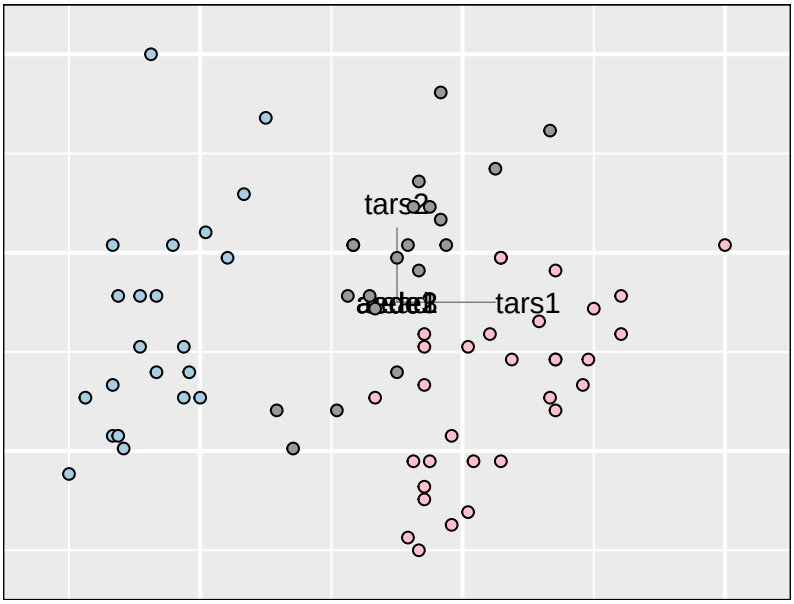

# 2D
animate(
  data = flea_std,
  tour_path = guided_tour(
    index_f = cmass(),
    d = 2
  ),
  display = display_xy()
)
animate(
  data = flea_std,
  tour_path = guided_tour(
    index_f = holes(),
```

```
    d = 2
  ),
  display = display_xy()
)
animate(
  data = flea_std,
  tour_path = guided_tour(
    index_f = lda_pp(flea$species),
    d = 2
  ),
  display = display_xy()
)
```

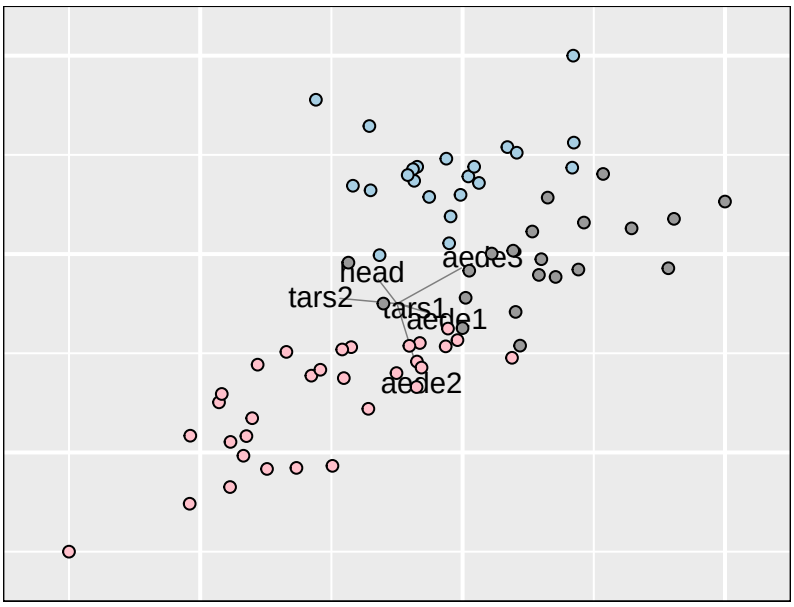
4.5 `tourr` in `loon`

We can convert our `tourr` plots into `loon` plots:

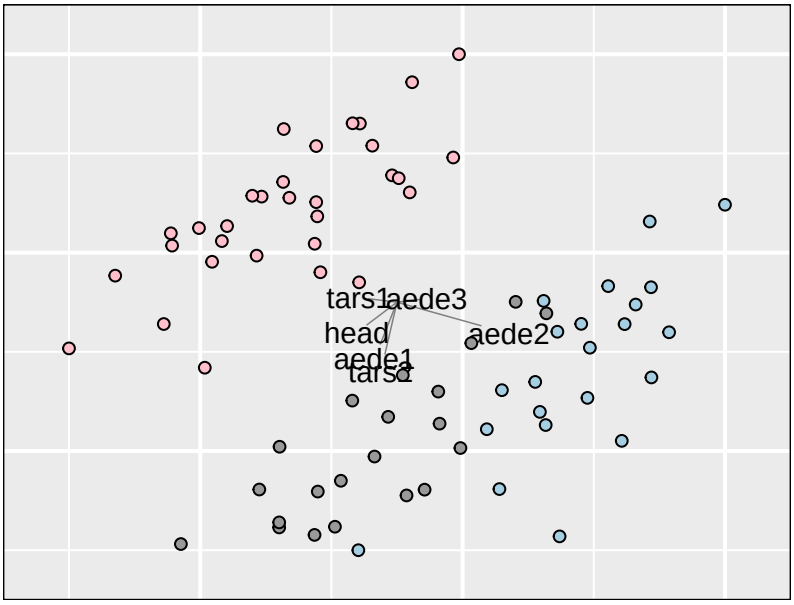
```
p = l_tour(
  data = flea %>% dplyr::select(-species),
  tour_path = tourr::grand_tour(2),
  color = flea$species
)
plot(p)
```




```
p = l_tour(
  data = flea %>% dplyr::select(-species),
  tour_path = tourr::guided_tour(
    index_f = tourr::holes(),
    d = 2
    ),
  color = flea$species
)
plot(p)
```




```
p = l_tour(
  data = flea %>% dplyr::select(-species),
  tour_path = tourr::guided_tour(
    index_f = tourr::lda_pp(flea$species),
    d = 2
    ),
  color = flea$species
)
plot(p)
```



References

Buja, Andreas, Dianne Cook, Daniel Asimov and Catherine Hurley. 2005. "Computational methods for high-dimensional rotations in data visualization." *Handbook of statistics* 24:391–413.

Chen, Chun-houh, Wolfgang Härdle, Antony Unwin, Dianne Cook, Andreas Buja, Eun-Kyung Lee and Hadley Wickham. 2008. "Grand tours, projection pursuit guided tours, and manual controls." *Handbook of data visualization* pp. 295–314.

Waddell, Adrian and R. Wayne Oldford. 2023. *loon: Interactive Statistical Data Visualization*. R package version 1.4.1.
URL: *https://CRAN.R-project.org/package=loon*

Wickham, Hadley, Dianne Cook, Heike Hofmann and Andreas Buja. 2011. "tourr: An R package for exploring multivariate data with projections." *Journal of Statistical Software* 40:1–18.

Xu, Zehao and R. Wayne Oldford. 2021. *loon.tourr: Tour in 'Loon'*. R package version 0.1.3.
URL: *https://CRAN.R-project.org/package=loon.tourr*