SHORT TECHNICAL NOTE

# Appendix: New and simplified manual controls for projection and slice tours

Ursula Laa[a], Alex Aumann[b], Dianne Cook[c], German Valencia[b]

[a]Institute of Statistics, University of Natural Resources and Life Sciences, Vienna; [b]School of Physics and Astronomy, Monash University; [c]Department of Econometrics and Business Statistics, Monash University

**ARTICLE HISTORY**
Compiled September 24, 2022

**KEYWORDS**
data visualisation; grand tour; statistical computing; statistical graphics; multivariate data; dynamic graphics

## 1. Refinements to enforce exact position

The problem with the new simple method (Algorithm 1) is that the precise values for $V_m$ cannot be specified because the orthonormalisation will change them.

### 1.1. *Adjustment method 1*

A small modification to algorithm 1 will maintain the components of $V_m$ precisely (Figure 1). It is as follows:

1. Provide $A$, and $m$.
2. Change values in row $m$, giving $A^*$.
3. Store row $m$ separately, and zero the values of row $m$ in $A^*$, giving $A^{*0}$.
4. Orthonormalise $A^{*0}$, using Gram-Schmidt.
5. Replace row $m$ with the original values, giving $A^{**}$.
6. For $d = 2$, adjust the values of $a^{**}_{\cdot 2}$ using

$$a^{**}_{j2} + \frac{a_{m1}a_{m2}}{p-1}, j = 1, ..., p, j \neq m.$$

which ensures that

CONTACT Ursula Laa. Email: ursula.laa@boku.ac.at, Alex Aumann. Email: aaum0002@student.monash.edu, Dianne Cook. Email: dicook@monash.edu, German Valencia. Email: german.valencia@monash.edu

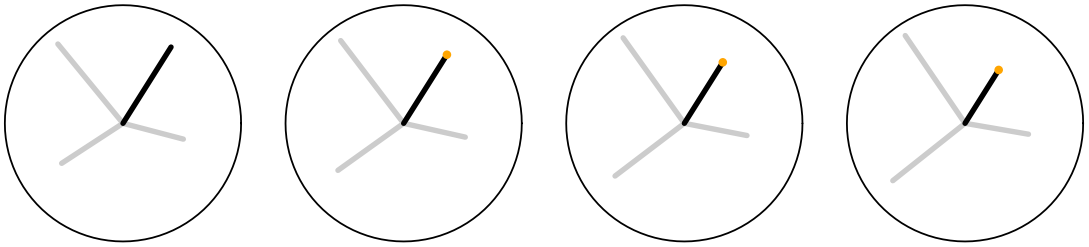

**Figure 1.** Manual controls with algorithm 2. The precise location of the axis is maintained.

$$\sum_{j=1,j\neq m}^{p} a_{j1}^{**} a_{j2}^{**} + a_{m1} a_{m2} = 0.$$

If $d > 2$ the process would be sequentially repeated in the same manner that Gram-Schmidt is applied sequentially to orthormalise the columns of a matrix. If $d = 1$ no orthonormalisation is needed, and the projection vector would simply need to be normalized after each adjustment.

### 1.2. *Adjustment method 2*

For $d = 2$ projections, the projection matrix is the sub-matrix $A$, of $O$ (an orthonormal basis for the p-dimensional space), formed by its first two columns as illustrated in Eq.˜1. Whereas orthonormality of the basis for the p-dimensional space is given by $e_i \cdot e_j = \delta_{ij}, i, j, = 1, \cdots, p$, orthonormality of the projection matrix is expressed as $\sum_{k=1}^{p} a_{ki} a_{kj} = \delta_{ij},\ i, j = 1, 2,\ k = 1, \cdots, p$. Movement of the cursor takes the two components $a_{m1}, a_{m2}$ into a selected new value $a_{m1}^*, a_{m2}^*$. Although the motion is constrained by $a_{m1}^{*2} + a_{m2}^{*2} \leq 1$, this is not sufficient to guarantee orthonormality of the new projection matrix. One possible algorithm to achieve this is

(1) Cursor movement takes $a_{m1}, a_{m2} \rightarrow a_{m1}^*, a_{m2}^*$, called $V_m$ above.
(2) The freedom to change the components $a_{m3} \cdots a_{mp}$ (the columns of $O$ not corresponding to the projection matrix $A$) is used to select a new orthonormal basis as follows:
   (a) For row $m$ one chooses $a_{m3}^* = \sqrt{1 - a_{m1}^{*2} - a_{m1}^{*2}},\ a_{m,k>3}^* = 0$
   (b) For other rows, $a_{i\neq m, j\geq 3}$ random selections in the range $(-1, 1)$ are made.
   (c) The Gram-Schmidt algorithm is then used to obtain an orthonormal basis taking $e_m$ as the first vector (which is already normalized), and then proceeding as usual $e_1 \rightarrow e_1 - (e_1 \cdot e_m) e_m$, $e_1 \rightarrow e_1/(e_1 \cdot e_1)$, etc, resulting in $A^{**}$.
(3) this results in the orthonormal basis $O^*$ and a new projection matrix with $A_{m1}^{**} = a_{m1}^*,\ A_{m2}^{**} = a_{m2}^*$.

The random completion of $O$ outside the projection matrix provides an exploration of dimensions orthogonal to the projection plane. However, it renders projections in a way that is not continuous and may be distracting. This can be alleviated by replacing the random completion with a rule restricting the size of the jumps, for example in step (b) the values $a_{i\neq m, j\geq 3}$ can be left unchanged before orthogonalisation.

$$O = \begin{array}{c} \\ e_1 \to \\ e_2 \to \\ \vdots \\ e_m \to \\ \vdots \\ e_p \to \end{array} \begin{array}{c} \begin{array}{cccccc} A_1 & A_2 & & & & \end{array} \\ \begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1p} \\ a_{21} & a_{22} & \cdot & \cdots & a_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdot & \cdots & a_{mp} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{p1} & a_{p2} & \cdot & \cdots & a_{pp} \end{pmatrix} \end{array}, \quad (a_{m1}, a_{m2}) \to (a^*_{m1}, a^*_{m2}) \qquad (1)$$

### 1.3. *Adjustment method 3*

Finally we may also consider an adjustment that works within the projection plane. The idea is that when applying Gram-Schmidt, the direction of the first basis vector is not changed. We thus rotate the projection matrix within the plane such that the first basis is aligned with the direction of the selected variable $m$ before applying orthonormalization. In addition we need to keep the contribution of variable $m$ along that direction fixed when normalizing the vector. The following steps are needed:

1. Provide $A$, and $m$.
2. Change values in row $m$, giving $A^*$.
3. Rotate the basis within the projection plane, such that the first basis direction is aligned with the direction of $m$ in $A^*$, giving $\hat{A}^*$. In a 2D projection this means finding the angle $\theta$ by which the basis should be rotated.
4. Normalize the first direction in $\hat{A}^*$ while keeping the contribution in row $m$ fixed, giving $\hat{A}^{*0}$.
5. Orthonormalize $\hat{A}^{*0}$ using Gram-Schmidt, giving $\hat{A}^{**}$. By construction the contribution of $m$ will remain fixed.
6. Rotate $\hat{A}^{**}$ back into the original basis to obtain $A^{**}$.

Diagram needed?

{Ursula, I am more convinced now that this is exactly method 2, the only difference is that you want to rotate to get to the "first" direction, whereas I simply relabel the rows of the matrix}

## 2. Software details

Here we describe the functions implemented in the Mathematica package `mmtour.wl`. The main function and its arguments are given below:

```
SliceDynamic[data, projmat, height, heightRange, legendQ(=1),
flagQ(=1), colorFunc(=ColorData[97])]
```

- `data`: A data matrix, potentially with grouping that has a label (string) and a flag (numerical) in the last two columns.
- `projmat`: The initial projection matrix. It is possible to start with a random projection with the input "random" (including the quotes) in this entry field.
- `height`: The initial height of the slice.
- `heightRange`: The range of slice heights to be explored.

- `legendQ`: A flag signaling whether the data matrix includes a column with group labels (1) or not (0).
- `flagQ`: A flag signaling whether the data matrix includes a numerical flag for the groups.
- `colorFunc`: specifies the colour mapping for the different groups.

The first four arguments are required. As possible starting values we suggest using a random input projection, and selecting slice height parameters using the estimate given in Section 2.4.2 of the paper.

The function renders a manual tour with controls that allow the user to navigate through different projections. For a given projection, a slider permits the user to change the slice width within its range and a separate box switches the view to a projection. The user can also change the centre point of slices; the size of the points rendered and the scale of the plot through sliders. An additional box displays the coordinates of the projection matrix explicitly.

This function can be used for ungrouped data by setting legendQ and flagQ to 0. To remove a group from the visual display, the user can choose specific colours for the entry colorFunc. For example, {Red,White,Green} would display groups 1 (red) and 3 (green) while making 2 invisible.

The Slice Tour* notebooks include the additional functions: `ProjectionPlot`, `Projected2DSliderPlot`, and `VisualiseSliceDyanmic`.

does this mean that we are changing the name of the package from mmtour.wl to Slice Tour notebook?

`ProjectionPlot` is very similar to `SliceDynamic`, except it only displays projections. Using it instead of `SliceDynamic` simplifies the display and is more efficient. The interface is slightly different, the data file should not contain labels for groups, if these are desired they are passed through the optional argument legendNames as {"group 1", "group 2",...}, for example. The function and its arguments are given below:

```
ProjectionPlot[data, projmat, legendNames(={}),
colorFunc(=ColorData[97])]
```

- `data`: A data matrix with one or more groups labeled by a flag (numerical, last column).
- `projmat`: The initial projection matrix. It is possible to start with a random projection with the input "random" (including the quotes) in this entry field.
- `legendNames`: an optional list of labels for the groups.
- `colorFunc`: specifies the colour mapping for the different groups.

`ProjectedLocatorPlot` displays the interactive controls slightly differently. Several locator panes are displayed above the display of the projected data and each one corresponds to a row in the projection matrix. The updating behaviour of the projection is the same as in `ProjectionPlot`. The function and its arguments are given below:

```
ProjectedLocatorPlot[data, projmat, legendNames(={}),
colorFunc(=ColorData[97])]
```

- `data`: A data matrix with one or more groups labeled by a flag (numerical, last column).
- `projmat`: The initial projection matrix. It is possible to start with a random projection with the input "random" (including the quotes) in this entry field.
- `legendNames`: an optional list of labels for the groups.

- `colorFunc`: specifies the colour mapping for the different groups.

Finally `VisualiseSliceDynamic` allows to discern which points exist inside the slice and outside it. This function differs from `SliceDynamic` in that it looks at only one data set but displays both the points inside and outside the slice. The display panel allows the user to specify the size of both sets of points (inside and outside) along with the slice height, centre point and zoom level. The function and its arguments are given below:

```
VisualiseSliceDynamic[data, projmat, height, heightRange]
```

- `data`: A data matrix containing only one group. In this case the data matrix should not contain column labels.
- `projmat`: The initial projection matrix. It is possible to start with a random projection with the input "random" (including the quotes) in this entry field.
- `height`: The initial height of the slice.
- `heightRange`: The range of slice heights to be explored.

The points inside the slice are shown in black whereas those outside the slice are shown in light blue, with labels indicating this.