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

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 orthonormalization 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^*.

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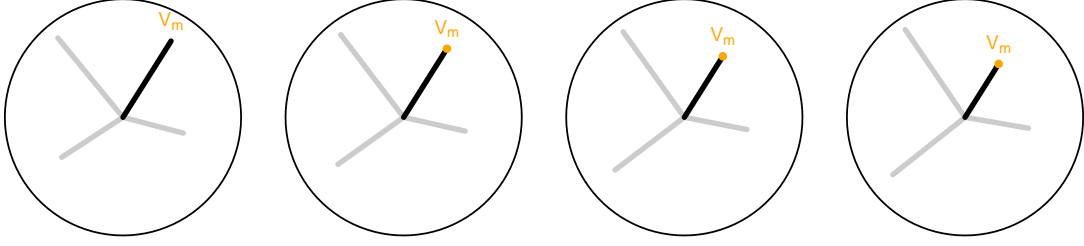
Figure 1. Manual controls with algorithm 2. The precise location of the axis is maintained.

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 $a^{**}_{.2}$ using

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

which ensures that

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

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 orthonormalization 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 the 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 $e_1 \rightarrow e_1 - (e_1 \cdot e_m)e_m$, $e_1 \rightarrow e_1/(e_1 \cdot e_1)$, 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 orthogonalization.

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

We could also take a similar approach without completing the basis, through rotation within the d-dimensional hyperplane defined by the new projection. In that case,

3

A. data center **B. center moved towards max value of X3** **C. center moved towards min value of X3**
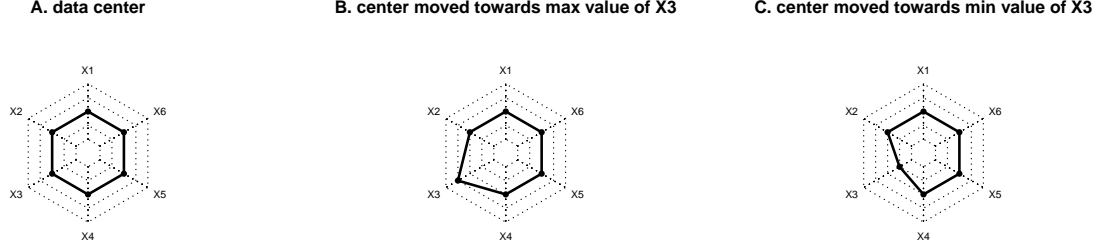
Figure 2. Visual guide for the slice center. Variable axes are displayed in polar coordinates, where the center corresponds to the minimum value and the outer end corresponds to the maximum value. The position of the center corresponds to the dark polygon. If the center is at the center of the data, this will be displayed as a regular polygon (A), and plots B and C show its position when moving the center along one axis.

we would again start from A^* as above, and then rotate the basis such that the first direction is aligned with the selected variable m. Applying Gram-Schmid within that basis definition will not alter the direction for the first vector, thus the direction of m remains fixed to what was selected with the cursor. By adjusting the normalization procedure we can also ensure that the length remains as selected manually by the user.

2. Slice center guide

The slice center, c, from Equation 1 in the main paper, is a point in the p-dimensional space. By default, it is placed in the center of the data (Figure 2 A). Moving this along a variable axis moves the slice outwards from the center of the data (Figure 2 B, C). The visual display of the center position is a form of star plot as suggested useful in Hurley, O'Connell, and Domijan (2022), where the position of the point is marked by a polygon on radial axes indicating its position relative to the variable minimum to maximum values.

3. 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 color 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 center 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 colors for the entry colorFunc. For example, {Red,White,Green} would display groups 1 (red) and 3 (green) while making 2 invisible.

The slice center position is controlled via numerical input along each axis in the original command 'SliceDynamic'. This provides numerical feedback on the position of the center. For easier navigation but without the numerical feedback, an implementation of the slice center control via a visual display like the one in Figure 2 is provided in 'SliceDynamicCC'.

The package includes the additional functions not used in the example application: `ProjectionPlot`, `Projected2DSliderPlot`, and `VisualiseSliceDyanmic`.

`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 color 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 behavior 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 color 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

6

allows the user to specify the size of both sets of points (inside and outside) along with the slice height, center 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.

References

Hurley, Catherine B., Mark O'Connell, and Katarina Domijan. 2022. "Interactive Slice Visualization for Exploring Machine Learning Models." *Journal of Computational and Graphical Statistics* 31 (1): 1–13. `https://doi.org/10.1080/10618600.2021.1983439`.