SHORT TECHNICAL NOTE

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

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

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

ARTICLE HISTORY
Compiled July 14, 2022

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

1. Refinements to enforce exact position

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

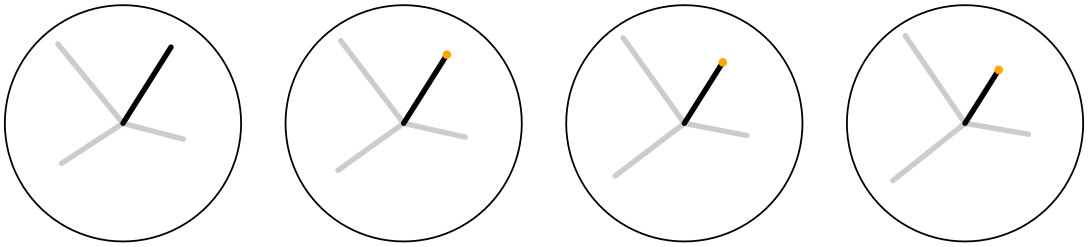

Figure 1. Manual controls with algorithm 2. The precise location of the axis is maintained.

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

.

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 of A formed by its first two columns. Whereas orthonormality of the basis for the p-dimensional space is given by $e_i \cdot e_j = \delta_{ij}, i, j, = 1, \cdots ,$, orthonormality of the projection matrix is expressed as $P_i \cdot P_j = \delta_{ij}, i, j = 1, 2$. Movement of the cursor takes the two components x_{m1}, x_{m2} into a selected new value a, b. Although the motion is constrained by $a^2 + b^2 \leq 1$, this is not sufficient to guarantee orthonormality of the new projection matrix. One possible algorithm to achieve this is

(1) Cursor movement takes $x_{m1}, x_{m2} \to a, b$
(2) The freedom to change the components $A_{i3} \cdots A_{ip}$ (the columns of A not corresponding to the projection matrix) is used to select a new orthonormal basis as follows:
   (a) For row m one chooses $A_{m3} = \sqrt{1 - a^2 - b^2},\ A_{m,k>3} = 0$
   (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 $e_1 \to e_1 - (e_1 \cdot e_m)e_m$, $e_1 \to e_1/(e_1 \cdot e_1)$, etc.
(3) this results in the orthonormal basis A^* and a new projection matrix with $P_{m1} = a,\ P_{m2} = b$.

The random choice for the components of A not in the projection matrix allows the exploration of dimensions perpendicular to the projection plane.

$$A = \begin{array}{c} \\ e_1 \to \\ e_2 \to \\ \vdots \\ e_m \to \\ \vdots \\ e_p \to \end{array} \begin{array}{c} \begin{array}{cc} \overset{P_1}{\downarrow} & \overset{P_2}{\downarrow} \end{array} \\ \begin{pmatrix} x_{11} & x_{12} & x_{13} & \cdots & x_{1p} \\ x_{21} & x_{22} & \cdot & \cdots & x_{2p} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & \cdot & \cdots & x_{mp} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{p1} & x_{p2} & \cdot & \cdots & x_{pp} \end{pmatrix} \end{array}, \quad (x_{m1}, x_{m2}) \to (a, b) \tag{1}$$

1.3. *Adjustment method 3*

A very simple adjustment method works purely within the (hyper)plane, and is using the fact that the first direction will not be rotated when applying Gram-Schmidt. 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 (hyper-)plane, such that the first direction is aligned with the direction of m in A^*, giving $\hat{A}^*$. In a 2D projection this means finding the angle θ by which the basis should be rotated.
4. Normalise the first direction in $\hat{A}^*$ while keeping the contribution in row m fixed, giving $\hat{A}^{*0}$.
5. Orthonormalise $\hat{A}^{*0}$ 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?

2. Software details

- **Why is this a good sandbox**
- **Explain the functionality available in the notebooks**

The following functions are implemented in the Mathematica package `mmtour.wl`:

- `ProjectionPlot[data, projmat, xRange(=Automatic), yRange(=Automatic), colour(=Automatic)]` with arguments
  - `data`: A list of data matrices (they must all have the same dimensionality)
  - ‘`projmat`: A projection matrix which describes the projection plane. This is the initial projection and does not need to be orthonormal (XXX is this right?).
  - ‘`xRange`: A list which details the range of the horizontal axis.
  - ‘`yRange`: A list which details the range of the vertical axis.
  - ‘`colour`: A list of graphics directives to be applied to each data matrix.

There are problems with the default values for x and y ranges and colour. It would be better to fix the ranges and colour based on the data as well as the origin of axes and the size of the points.

It would be good if the initial projection matrix (and maybe the center) had default values. Perhaps a random matrix with the correct dimensions read from the data? Fix 1

The format for the printed projection matrix can be improved also. Maybe fix the number of digits and the position of the label.

- `SlicePlot[data, projmat, centrePoint, height, heightRange, ptSize1(=0.005), ptSize2(=0.004), minDist(=0)]` with arguments (used to be called SliceDynamic in Alex's first version, the name SliceDynamic was taken by something else below)
  - `data`: A data matrix
  - `projmat`: The inital projection matrix
  - `centrepoint`: The initial position of the slice
  - `height`: The initial height of the slice
  - `heightRange`: The range of heights
  - `ptSize1`: The size of the points that exist within the slice
  - `ptSize2`: The size of the points that exist outside of the slice

This function requires some auxiliary functions. It does not accept more than one data set.

It would be good if the initial projection matrix (and maybe the center) had default values. Perhaps a random matrix with the correct dimensions read from the data?

It may be good to fix the size of the points, the center of the axes and the scales of the axes rather than allow them to change as the projection and slicing changes

- `SliceDynamic[data, projmat, centrePoint, height, heightRange, ptSize1 : 0.005, ptSize2 : 0.004]` with arguments
- `data`: A data matrix with an extra grouping or clustering column (numerical)
- `projmat`: The initial projection matrix
- `centrepoint`: The initial position of the slice
- `height`: The initial height of the slice
- `heightRange`: The range of heights
- `ptSize1`: The size of the points that exist within the slice
- `ptSize2`: The size of the points that exist outside of the slice

Does this need to be a separate function from SlicePlot? Fix 2

It would be good if the initial projection matrix (and maybe the center) had default values. Perhaps a random matrix with the correct dimensions read from the data? Fix 3

It may be good to fix the size of the points, the center of the axes and the scales of the axes rather than allow them to change as the projection and slicing changes. Fix 4

I have found it useful to input different data sets rather than have this function sort them automatically using the last column. One data set with a flag column at the end, is best! Is the code too complicated? Fix 5

- `Projected2DSliderPlot[]`

This function uses a different display for the projection that I find very useful. However as it stands it is specific to one data set. Needs to be generalised to arbitrary data if we want to keep it.

For Alex to do

- **We would need a screenshot (and video) for each of these.**
- **Fix 1, 2, 3, 4, 5**
- **Code consistent for applications to work - see slice_tour_compare_sets.nb. See below**
  - **We need to make sure the drawing range is fixed (no resizing of the axis when changing projection or slicing). It could be nice if there was a manual zoom in/zoom out to change scale occasionally.**
  - **Should also make sure the size of the points is fixed (maybe this will be automatic if the drawing range is fixed?), bigger points would be preferred, currently the size is sometimes really tiny and it becomes difficult to see anything.**
  - **Can we have some heuristic to decide the range for the slice thickness slider? In the second graph there is a tiny part of the slider where the changes are interesting, difficult to navigate since this makes it super sensitive to any small movement, while most of the options are not interesting. My suggestion would be to keep (as I understand is the case now) the maximum thickness (all points in) as the upper bound, but instead of zero use a thickness below which slices tend to be empty as the lower bound.**
  - **For the olives example I think the relation between x2 and the blue region could be interesting.**
  - **pdfsense data: can we use random samples instead of the first 10k rows? Then we should still be able to see the bigger picture (with picking the first ones we get some artificial structure from how the data was sampled initially).**
  - **I was playing around with this today. Looks like the axes are fixed when changing slice thickness, but not when rotating?**
  - **Anyways, some observations that might be useful, but all of this seems difficult for including in a paper:**
  - **In the first example we can show how the yellow group is associated with the first variable (x1, I guess this is C9?) - when slicing through the center it only appears when x1 has a bigger component on the projection, it is associated with low values of x1. Since they are not near the average value of x1 the points do not get captured in a slice where x1 is not important in the projection. We can move the center point to -1 in the first component to then explore how the yellow group relates to the other three variables. Much less clear, maybe some indication of smaller x2 and larger x3 for this.**
  - **The second example is too busy for manual exploration I would say (too many axes to move around separately).**
  - **Third example could be similar to the first: blue region requires small values of x2, can be understood from the exploration, then move center point to -0.5 in x2 to explore further. Now interpretation becomes tricky. Sometimes blue and yellow seem just randomly everywhere, sometimes blue more contained in an ellipsoid around the center (indicating some correlation between**

variables). For example try zeroing out components 2 and 3 to see this slice in x1 vs x4.

- **For the last example x4 seems to be the one to work with. I feel like this is again very similar, maybe better to instead explore other examples (maybe something without slicing?)**