Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

tourr: An R package for exploring multivariate data with projections

Hadley Wickham
Rice University

Dianne Cook
Iowa State University

Heike Hofmann
Iowa State University

Andreas Buja
Wharton School, University of Pennsylvania

Abstract

This paper describes an R package which produces tours of multivariate data. The package includes functions for creating different types of tours, including grand, guided, and little tours, which project multivariate data (p-D) down to 1, 2, 3, or, more generally, d ($\leq p$) dimensions. The projected data can be rendered as densities or histograms, scatterplots, anaglyphs, glyphs, scatterplot matrices, parallel coordinate plots, time series or images, and viewed using an R graphics device, passed to ggobi, or saved to disk. A tour path can be stored for visualisation or replay. With this package it is possible to quickly experiment with different, and new, approaches to tours of data. This paper contains animations that can be viewed using the Adobe Acrobat pdf viewer.

The **tourr** package is available on CRAN.

Keywords: grand tour, guided tour, projection pursuit, little tour, local tour, correlation tour, visualization, statistical graphics, visual data mining.

1. Introduction

Many of us still struggle to explore multivariate data. We would like a magic button to tell us about all of the structure in the data. The closest we have to a magic button right now, at least for real-valued data, is the tour, which shows a smooth sequence of projections of high-dimensional data. The tour is most useful when looking for clusters, outliers, non-linear dependence, and to get an overview of the types of structures present in multivariate data.

The tour gets us beyond the single static data projection produced by many statistical meth-

ods like principal component analysis, linear discriminant analysis, multidimensional scaling, projection pursuit or independent components analysis (Johnson and Wichern 2002). With the tour the data analyst sees many data projections, including ones revealing many different aspects of the data and how these are related to each other.

There have been numerous implementations of tours, although few are accessible to the casual user, and none of them readily encourages experimentation. The earliest implementation is described in Buja, Hurley, and McDonald (1986), from which our work mostly derives. Tierney (1991) coded a grand tour in XLispStat. Carr, Wegman, and Luo (1996)'s ExplorN has a grand tour in which k-D data projections can be displayed as a scatterplot matrix or parallel coordinates. A grand tour is programmed in DAVIS (Huh and Kim 2002). GGobi (Swayne, Lang, Buja, and Cook 2003; Swayne, Temple Lang, Cook, Buja, Wickham, Lawrence, and Hofmann 2001) has a grand, guided, and correlation tour, as well as a tour with manual controls. Different variations of the grand tour algorithm are used in each implementation. The one used in ggobi, and also the **tourr** package, is called the geodesic random walk method. It is documented in several places: originally in Buja and Asimov (1986), and more recently in Buja, Cook, Asimov, and Hurley (2005); Cook, Lee, Buja, and Wickham (2008). Articles describing the application of tours to particular types of problems include Wegman and Luo (1997), Symanzik, Wegman, Braverman, and Luo (2002), Wilhelm, Wegman, and Symanzik (1999), Cook and Swayne (2007), and Cook (2009).

This paper presents the **tourr** package, which provides a tool-kit of methods that allow an assembly of all the tours described in the literature to date, and facilitates development of new tour methods specialized for the needs of a particular problem. Currently, the focus of the package is on providing a testbed for tour research, but it also provides a user-friendly layer, and prototype GUI for using tour methods on your data.

Section 2 defines the tour and the three components that define a given tour. The three sections describe each of these components in turn: Section 2.1, the tour path; Section 2.2 display methods; and Section 2.3 the data. As well as displaying the results of a tour dynamically, tours can also be saved, replayed and visualized. Section 3.3 shows what you can do, and other options are described in Section 3. Section 4 explains how to extend the package. Finally, we conclude with our plans for future work and suggest interesting avenues of research into tours in Section 5.

2. The tour method

We define a tour to consist of the following three components:

- a data matrix ($n \times p$), with real-valued elements,
- a tour path that produces a smooth sequence of projection matrices ($p \times d$), and
- a display method that renders the projected data.

This allows us to recombine tours to produce new ones. It also allows us to better understand how existing tours relate to one another, and to see where there are gaps in the current design that could be filled with new methods.

The structure of the **tourr** package reflects this construction of the tour: to create a tour we need to combine a dataset with a type of tour path and display method. The code style follows these components also:

```
tour_function(data, tour_path, display_method)
```

The `tour_function` packages the tour. It is, currently, either a real-time animation, `animate`, or output to file, `render`. The first argument to the function is the dataset, the second argument selects a tour path, and the third sets the display type.

Figure 1 shows three different tours of the flea dataset (Lubischew 1962): (1) a 2-D tour displayed in a scatterplot, (2) a 3-D tour displayed with simulated depth, and (3) a 4-D tour displayed in a parallel coordinates plot. (If you are viewing this paper in Adobe Acrobat, you can click on each image to see the first few seconds of each tour.) The tours were generated using this code:

```
animate(flea[, 1:6], grand_tour(d = 2), display = display_xy())
animate(flea[, 1:6], grand_tour(d = 3), display = display_depth())
animate(flea[, 1:6], grand_tour(d = 4), display = display_pcp())
```

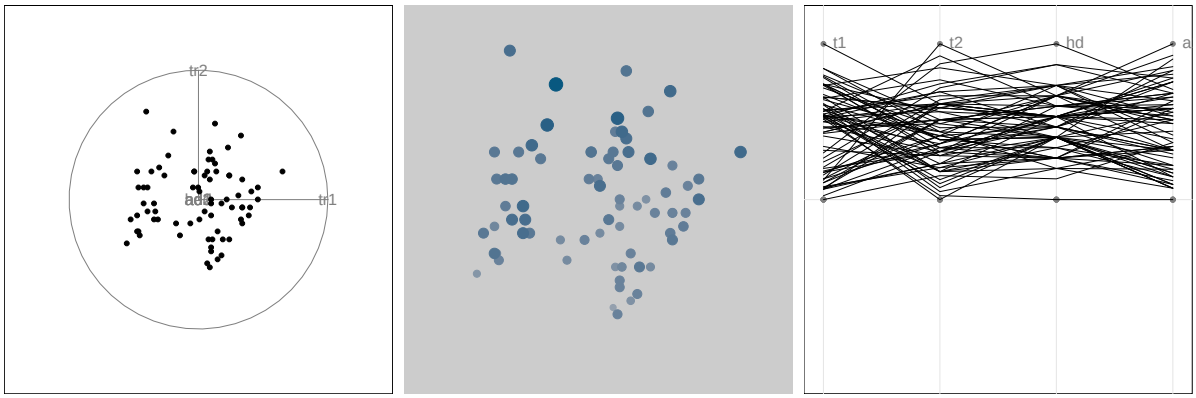

Figure 1: Three tours (from left to right): a 2-D tour displayed with a scatterplot, a 3-D tour displayed with simulated depth, and a 4-D tour displayed with a parallel coordinates plot. (If you are viewing this in Acrobat, click on each image to see a few seconds of the tour.)

The different display methods are described in detail in Section 2.2. Shortcuts to the full syntax are typically used, e.g. using `animate_xy` instead of `animate` allows the display argument to be dropped.

A tour path is constructed by a function, with the arguments controlling the movement through the space. The `grand_tour`, used above, takes a random walk on the space of projections. Here are two more examples:

```
animate_xy(flea[, 1:6], guided_tour(index = holes))
```

guides the tour towards projections where there is a "hole" in the center of the distribution, and

```
animate_xy(flea[, 1:6], little_tour(d = 2))
```

steers the views through every pair of variables. More explanation of tour paths is given in the Section 2.1.

2.1. Tour path

The tour path is made up of two parts: the **interpolator** smoothly interpolates between pairs of projections produced by the basis **generator**. The smooth interpolation is an important part of the tour as it ensures that the data appears to move smoothly from one projection to the next.

Checks in the code ensure that subsequent bases in the sequence are at least a small distance apart, and terminates the tour when it is done or in the case of a guided tour, has reached a (local) maximum.

Generators

The **tourr** package includes five generators: the grand tour, the guided tour, the planned tour, the dependence tour and the local tour. Internally, each generator consists of a function with two arguments, the `current` projection matrix (which is `NULL` for the first projection) and the `data`.

1. The `grand_tour` (Asimov 1985; Buja and Asimov 1986) picks a new $p \times d$ projection matrix at random.

   The grand tour provides a curve filling the space of projections, ensuring thereby to (eventually) show every possible projection of the data. It is useful for getting a comprehensive overview of a dataset, but even for a moderate number of dimensions it can take a long time to see everything.

   A variant on the grand tour is the `frozen_tour`: it picks a new target projection at random, while holding some variables constant.

2. In the `guided_tour` (Cook, Buja, Cabrera, and Hurley 1995), instead of picking a new projection completely at random, we pick one that is more interesting. Over time, this leads to picking projections that are closer to the current projection, so that we eventually converge to a single maximally interesting projection, in a spirit similar to simulated annealing (Kirkpatrick, Gelatt, and Vecchi 1983).

   The guided tour is a dynamic form of projection pursuit (Huber 1985; Friedman and Tukey 1974) with the difference that instead of just seeing the final "best" result, we see all of the interesting local maxima on the way. Like projection pursuit, we need to define what we mean by interesting, by describing a mathematical index (e.g. Lee, Cook, Klinke, and Lumley (2005)) that quantifies the interestingness of a data projection. The **tourr** package comes with four indices:

   - holes (`holes`)
   - central mass (`cm`)
   - lda (`lda_pp`)
   - pda (`pda_pp`)

Each of these indices takes a $n \times d$ data matrix as input and returns a single number as output. Section 4.3 describes how to write new indices.

Like the grand tour, the guided tour can also be frozen to produce the `frozen_guided_tour`.

3. The `planned_tour` is the most constrained tour: we already know where we want to go and simply cycle through a pre-specified set of frames. The planned tour is most useful if you have a sequence of projections saved by an earlier tour for later replay: see Section 3.3 for details.

   This idea is also the basis for the `little_tour`. The little tour is a special case of a planned tour that cycles through all axis parallel projections of dimension d, i.e. it provides a smooth sequence between views of all d-dimensional sets of variables in the data.

4. The `dependence_tour` combines n independent 1-D tours. This generator has a single argument, a numeric vector that specifies which 1-D tour each variable should be assigned to. For example, `c(1, 1, 2, 2)` specifies that the first two variables will be displayed with a 1-D tour on the first axis, and the second two with a 1-D tour on the second axis.

   The correlation tour (Buja *et al.* 1986) is the two dimensional special case of this method. It corresponds to canonical correlation analysis in the same way as the grand tour is analogous to PCA. Similarly, the dependence tour corresponds to generalized canonical correlation analysis.

5. The `local_tour` alternates between a specified starting position and nearby random projections. This allows us to inspect the local neighborhood of a projection.

Interpolator

All of the generators currently rely on *geodesic interpolation* as the means for a smooth interpolation between planes. This method was first described by Asimov (1985) and Buja and Asimov (1986), and is discussed in more detail by Cook *et al.* (2008) and Buja, Cook, Asimov, and Hurley (2004); Buja *et al.* (2005). Other methods of interpolation are Givens and Householder rotations, but these methods interpolate between frames rather than planes, resulting in within-plane rotation, which makes them useful only for special purposes. They were available in XGobi (Swayne, Cook, and Buja 1992) but have not yet been implemented in **tourr**.

Some background: Because a projection of p-D data onto d ($< p$) dimensions must ultimately be rendered in terms of d axes (Cartesian or parallel), a tour is internally represented by a path of so-called "d-frames" in p-space, that is, orthonormal $p \times d$ matrices where the columns represent the projection vectors. Each d-frame spans a unique "d-plane" (we use "plane" also when $d > 2$), but each d-plane can be spanned by many d frames, all of which are rotations and reflections of each other within the d-plane. The space of d-frames is called a Stiefel manifold, whereas the space of d-planes is called a Grassmann manifold.

An important goal of any tour must be to avoid unnecessary rotation within the plane because any two frames that span the same plane will generate d-D projections of the p-D data with equivalent information. In order to get a truly new view of the data "from a different side",

we should move the plane, not just rotate it within itself. Motion that completely avoids such within-plane rotation is indeed generated by the above mentioned geodesic interpolations. In this technical sense geodesic interpolation is optimal.

2.2. Display methods

The display methods produce a visual rendering of the tour: a visualisation of the projected data. There are nine methods for displaying the tour, which can be classified based on the dimensionality of the projection:

- 1-D:
  - `animate_dist` (histogram, average shifted histogram, density plot)
  - `animate_image` (image plot)
  - `animate_ts` (time series)
- 2-D:
  - `animate_xy` (scatterplot)
- 3-D:
  - `animate_stereo` (anaglyphs)
  - `animate_depth` (3-D depth cues)
- k-D:
  - `animate_andrews` (Andrews curves)
  - `animate_faces` (Chernoff faces)
  - `animate_pcp` (parallel coordinates)
  - `animate_scatmat` (scatterplot matrix)
  - `animate_stars` (star glyphs)

The tour path described earlier has no sense of time, it just provides paths over the Grassmann manifold. The animation methods provide temporal control, using two parameters: frames per second, `fps`, and angular velocity, `aps`. Increasing `fps` will produce smoother motion (up to a limit imposed by the processing and drawing speed on your computer). Changing `aps` changes the speed of the tour along the tour path.

1-D

A 1-D projection of the data can be thought of like the first principal component in principal component analysis, or even a the linear combination of variables forming a regression equation. 1-D data is typically displayed as a histogram, average shifted histogram (Scott 1985) or kernel density estimate (Scott 1995). The **tourr** package provides the three displays within the `animate_dist` function and produces displays like those in Figure 2. These displays are centered by default, so that the density does not wander to the left and right.

Two special applications also fall in to the 1-D category, multivariate spatial data, using `animate_image`, and multivariate time series using `animate_ts`. Wegman, Poston, and Solka

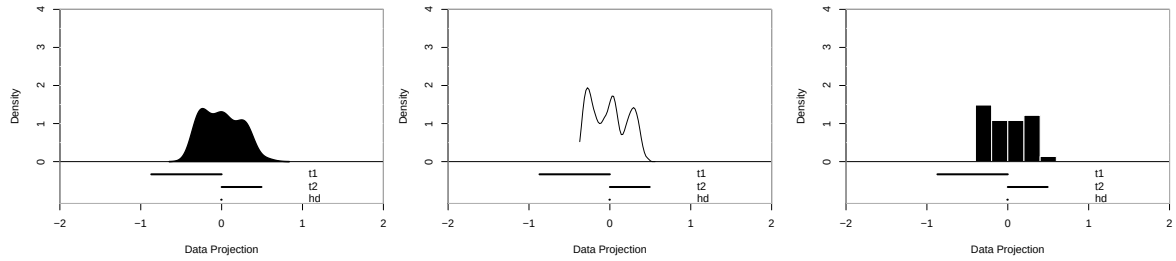

Figure 2: Three visualizations of a 1-D projection. From left to right: a density plot, an average shifted histogram and a histogram. In each display the projection coefficients, that range between -1 and 1, are displayed as line segments underneath the plot.

(2001) uses the term pixel tour for the tour on spatial data. This tour assumes the data has two spatial dimensions, which remain fixed and do not enter into the tour, and multiple measurements at each spatial location. A 1-D tour is generated by taking projections of the multivariate measurements. The value of the projected data at each spatial location is converted to a color code, which displays as an image. Wegman *et al.* (2001) used this to examine photograms of petroglyphs and satellite imagery of suspected land mine sites. The time series tour is similarly done, but it is not yet implemented in the package.

2-D

For 2-D projections a scatterplot is used. This was the original approach used when the tour was defined (Buja and Asimov 1986; Buja *et al.* 1986). Point colors and symbols can be set using the standard `pch` and `col` parameters.

3-D

It is difficult to show a 3-D object on a 2-D display. The **tourr** package provides two approaches:

- Anaglyphs, with `display_stereo`. You'll need red-blue glasses to use this tour. Anaglyphs were used first used to visualize results of the tour by Carr, Nicholson, Littlefield, and Hall (1987).
- Simulated depth cues, with `display_depth`. This display uses depth cues of occlusion (closer points occlude further away points), size (closer points are bigger) and saturation (distant points are hazier and less saturated). The illusion of depth is less convincing than with anaglyphs, but it does not require any special equipment.

It is possible to do better if specialized hardware is available. For an example see Nelson, Cook, and Cruz-Neira (1999).

k-D methods

When we have k-D projections, we can use any of the following methods to display the k-d projected data:

- Parallel coordinates (Wegman 1990; Inselberg 1985) with `display_pcp`. Parallel coordinates were used to display tour results in CrystalView (Wegman and Luo 1997).
- Andrews curves (Andrews 1972) with `display_andrews`.
- Scatterplot matrices, `display_scatmat`. Also used in CrystalView.
- Glyph based displays: stars (`display_stars`) and Chernoff faces (`faces`). We have not seen tours used with the glyph based methods (stars and faces), but it seems like a natural way to overcome the ordering problem implicit with a choice of single projection.

It is very easy to hook up any additional method of data display, particularly if it is already implemented in R.

Animations

The `animate_**` functions use `animate()` to produce the tour animations. `render()` has three arguments to control the tour, the `data`, the `tour_path` and `display`, and and further arguments to control the speed, frame rate, length of the tour and data scaling.

The `render` function, alternatively, can be used to save the plots to files, as was done for the animations included in this paper. It takes similar arguments to the `animate` function with the additional specification of the graphics device to be used, for example `pdf`. We created the animations in this paper by rendering the plots to pdf files and using the Latex **animate** package to glue these together. This is also useful if you want to use high-quality graphics to create the movie for presentation-quality display, for example using the **ggplot2** graphics (Wickham 2009) package.

2.3. Data

Most of the tour methods assume that you are working on a matrix or data frame of p continuous variables. By default these variables are scaled to each have range $[0, 1]$, but if your variables are measured on a common scale already, you can turn this off by setting `rescale = FALSE`. Optionally, the data can also be sphered prior to display with the tour. Sphering rotates and scales the data so that it has a diagonal variance-covariance matrix - this is useful because it removes typically obvious correlation effects and makes it easier to see subtler non-linear patterns.

3. Options

3.1. Frozen geodesics

The interpolation has an additional component that allows the coefficient of some variables in a projection to be "frozen". These were used in XGobi (Swayne, Cook, and Buja 1998), but have not previously been described in the literature.

In a frozen interpolation, one or more of the variables have a fixed coefficient in the corresponding row of the projection matrix. Interpolation then occurs within the subspace generated by this restriction. In practice, this is a simple modification of the interpolation algorithm, given by the following:

1. A matrix of frozen values, the *freezer* matrix. This is a matrix of the same shape as the projection matrix, containing numbers for frozen variables, and missing values for warm variables (variables that can vary freely). The columns of the freezer matrix must have norm less than 1.

2. A freeze operation which zeroes out the frozen variable values in the input projection matrix.

3. A thaw operation, which thaws the input projection matrix by replacing the 0's of the frozen variables with the values from the freezer matrix, appropriately scaling the columns to have norm 1.

The frozen geodesic interpolation is then just the regular geodesic interpolation plus:

1. Freeze the current basis, and freeze the target basis.

2. Perform the geodesic interpolation between the two frozen bases.

3. Thaw each resulting interpolated basis.

For example:

$$\mathbf{F} = \begin{bmatrix} — & — \\ — & — \\ 0.5 & 0.5 \\ — & — \end{bmatrix} \quad \mathbf{A} = \begin{bmatrix} 0.91 & -0.08 \\ -0.12 & 0.75 \\ 0.06 & -0.44 \\ -0.39 & -0.49 \end{bmatrix} \quad \text{freeze}(\mathbf{A}, \mathbf{F}) = \begin{bmatrix} 0.91 & -0.08 \\ -0.12 & 0.75 \\ 0 & 0 \\ -0.39 & -0.49 \end{bmatrix}$$

$$\mathbf{F} = \begin{bmatrix} — & — \\ 0.6 & 0.6 \\ — & — \\ — & — \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 0.91 & -0.08 \\ 0 & 0 \\ -0.12 & 0.75 \\ -0.39 & -0.49 \end{bmatrix} \quad \text{thaw}(\mathbf{B}, \mathbf{F}) = \begin{bmatrix} 0.73 & -0.07 \\ 0.60 & 0.60 \\ -0.10 & 0.67 \\ -0.31 & -0.44 \end{bmatrix}$$

3.2. Optimization

The **tourr** package provides three methods for searching projection space for more interesting projections:

- `search_better, search_better_random`: inspired by simulated annealing, these methods have been modified for better behavior in the interactive case.
- `search_geodesic`: a new method for stochastic coordinate-wise search.

Properties of an optimization algorithm suitable for visual monitoring include:

- *Monotonicity*: Index values for projections in the interpolation between starting and target bases increase monotonically. If not, the optimization will stop and search for a new target.
- *Variable Step-Size*: Two mechanisms for actually attaining the maximum are needed: (1) step size of the tour is reduced as a maximum is approached, to avoid overshooting of peak. (2) Similar to simulated annealing, the search neighborhood is decreased over time, so that there is a better chance of reaching the nearest maximum.

- *Local Stopping Criterion*: A mechanism exists that allows to jump out of a local maximum, if desired. When this happens the search neighborhood is expanded again.

3.3. History

Tour paths can be saved and replayed. The function `save_history()` works like `animate` and `render` except that it returns a 3-D array of projection matrices, instead of displaying data projections on screen. The planned tour can then be used to replay the tour, and even investigate the tour path. Figure 3 shows an example. The code:

```
f1 <- save_history(flea[, 1:6],grand_tour(d = 1), max_bases = 10)
render(flea[, 1:6], planned_tour(f1), display_dist(), frames = 50,
  "pdf", "tour1d-animation.pdf", width = 4, height = 4)
f1interp <- interpolate(f1)
```

generates a tour path, and saves it in the data structure `f1interp`. It is used to produce the 1-D tour in the left plot of Figure 3. The tour path can be viewed in the full 6-D space using a tour in ggobi, using:

```
x <- path1d_ggobi(f1)
ggobi(x)
# Brush path points large, and bright, and supporting sphere small, dull
```

This is displayed in the middle plot of Figure 3. The light grey points are points on the surface of a 6-D unit-radius sphere, which have been added to give some context to the tour path (orange). The tour path is a track on the surface of the sphere. The example used here is a short tour path, but key components can be seen: each time the path takes a sharp turn a new geodesic interpolation is used to move to a new target basis. The path makes reasonably rapid progress over the surface of the sphere. With a long tour path we would expect to see that the surface of the sphere is covered by this path.

The tour path can also be visualized in a low-dimensional representation produced by multidimensional scaling (right plot of Figure 3). The distance between two planes can be measured in an almost unique way (up to a choice of units) by forming the Euclidean length of the d "principal angles" between the two planes: $D(P_1, P_2) = (\theta_1^2 + ... + \theta_d^2)^{1/2}$. There is, therefore, a sense of "angular distance" between two planes, and such distances can be used to visualize the path taken by a tour, for example, by mapping the path with multidimensional scaling. The code used to produce this is:

```
d <- history_dist(f1interp)
ord <- as.data.frame(MASS::isoMDS(d)$points)
qplot(V1, V2, data = ord, geom="path") +
  coord_equal() + labs(x = NULL, y = NULL)
```

Tour paths can be saved as variables, but re-using these variables will not replay the same tour as the path is stochastic. To re-construct a path, either set the random seed, or, better, save the sequence of projections generated by tour path and then replay it with the *planned tour*:

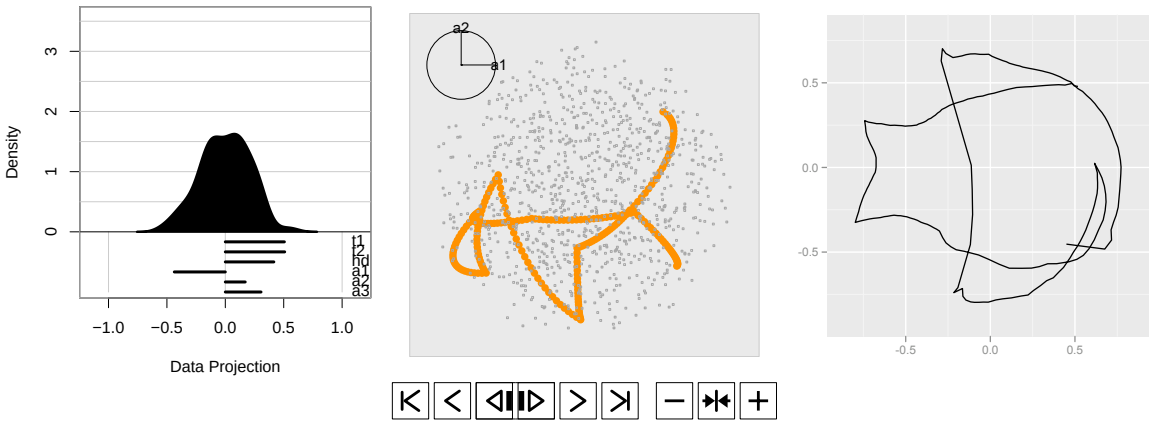

Figure 3: Visualizing a tour path to examine the tour's coverage of the projection space. The left plot shows the 1-D tour, animated upon clicking. The middle plot shows the sequence of 1-D projection vectors (orange) on a 6-D sphere (grey) that defines the tour path, itself viewed in a tour. The right plot shows a 2-D multidimensional scaling representation of the tour path.

```
gt <- grand_tour(d = 4)
animate_pcp(flea[, 1:6], gt)
animate_pcp(flea[, 1:6], gt) # Will take a different path!

set.seed(1410)
animate_pcp(flea[, 1:6], gt)
set.seed(1410)
animate_pcp(flea[, 1:6], gt) # Will take the same path.

path <- save_history(flea[, 1:6], gt, 10)
animate_pcp(flea[, 1:6], planned_tour(path))
animate_pcp(flea[, 1:6], planned_tour(path)) # The same path again
```

Other methods for visualizing a sequence of projections as part of a tour path are described in Section 3.3.

4. Extending the package

The **tourr** package uses a layered design to make it possible to customize almost every aspect of the tour. This means you need to learn about the existing abstractions, for example, closures. An explanation follows, along with examples of possible extensions.

In **tourr**, closures are used for three purposes:

- To allow functions to maintain the state between subsequent calls.
- To create functions that meet the requirements for basis generators (two arguments: current projection and data matrix) and index functions (one argument: a matrix of

the projected data) while still being able to control aspects of their behavior with other parameters.

- To create simple "objects" to describe the different components of the display of projections.

4.1. Closures

A closure is a function written by another function. Closures are so called because they *enclose* the environment of the parent function, and can access all variables and parameters in that function. This is useful because it allows us to have two levels of parameters. One level of parameters (the parent) controls how the function works. The other level (the child) does the work. The following example shows how can use this idea to generate a family of power functions. The parent function (`power`) creates child functions (square and cube) that actually do the hard work.

```
power <- function(exponent) {
  function(x) x ^ exponent
}

square <- power(2)
square(2) # -> [1] 4
square(4) # -> [1] 8

cube <- power(3)
cube(2) # -> [1] 8
cube(4) # -> [1] 64
```

The ability to manage variables at two levels also makes it possible to maintain the state across function invocations by allowing a function to modify variables in the environment of its parent. Key to managing variables at different levels is the double arrow assignment operator `<<-`. Unlike the usual single arrow assignment (`<-`) that always works on the current level, the double arrow operator can modify variables in parent levels.

This makes it possible to maintain a counter that records how many times a function has been called, as the following example shows. Each time `new_counter` is run, it creates an environment, initializes the counter `i` in this environment, and then creates a new function.

```
new_counter <- function() {
  i <- 0
  function() {
    # do something useful, then ...
    i <<- i + 1
    i
  }
}
```

The new function is a closure, and its environment is the enclosing environment. When the closures `counter_one` and `counter_two` are run, each one modifies the counter in its enclosing environment and then returns the current count.

```
counter_one <- new_counter()
counter_two <- new_counter()

counter_one() # -> [1] 1
counter_one() # -> [1] 2
counter_two() # -> [1] 1
```

For more details about environments in R, see Fox (2002).

4.2. Writing a new tour path

A tour path is composed of a basis generator and an interpolation algorithm. The **tourr** package provides geodesic interpolation. Writing a new basis generator is all that is required to produce a new tour. We will use the grand tour generator as an example:

```
grand_tour <- function(d = 2) {
  generator <- function(current, data) {
    if (is.null(current)) return(basis_init(ncol(data), d))

    basis_random(ncol(data), d)
  }
  new_geodesic_path(generator, name = "grand")
}

basis_random <- function(n, d = 2) {
  mvn <- matrix(rnorm(n * d), ncol = d)
  orthonormalise(mvn)
}

basis_init <- function(n, d) {
  start <- matrix(0, nrow = n, ncol = d)
  diag(start) <- 1
  start
}
```

There are three functions, the first defines the tour, and the other two define the basis generation functions. The key components are:

- The basis **generator** is defined inside the tour path, and passed to a function, `new_geodesic_path`, that wraps the generator in the geodesic interpolator to supply a series of piece-wise geodesics.
- The generator takes two arguments: the `current` projection and the `data` to be projected. The `data` argument is currently only used by the `guided_tour` in order to compute the projection pursuit index.

- The generator deals with a special case, when the current projection is `NULL`. This occurs when the tour is started for the first time, and there is no current projection. In this case it uses the special basis generator, `basic_init`, which assigns the first d variables to the first d output dimensions.
- The basis generators, `basis_random` and `basis_init`, are separate functions, which allows their use in other types of tours.

4.3. Writing a new index for the guided tour

The guided tour requires a method for calculating how interesting a particular data projection is. If the index depends only on the projected data, then the code is simple, for example, the code for the holes index:

```
holes <- function(mat) {
  n <- nrow(mat)
  d <- ncol(mat)

  num <- 1 - 1/n * sum(exp(-0.5 * rowSums(mat ^ 2)))
  den <- 1 - exp(-d / 2)

  num / den
}
```

If the index requires additional arguments, the function needs to take those arguments and generates the index function using the closure style discussed above. The following code shows how the implementation of the LDA index. Like linear discriminant analysis, the LDA index is maximized by a projection where the classes (two or more) are the most separated. Thus, it needs a vector of class information, a class label for each row of the data matrix. These are passed to `lda_pp` which then generates the actual index function. The index function needs a single argument, `mat`, and is implemented as a closure, which means that when `cl` is referred to, it looks in the environment of the top-level function.

```
lda_pp <- function(cl) {
  if (length(unique(cl)) == 0)
    stop("ERROR: You need to select the class variable!")
  if (length(unique(cl)) == 1)
    stop("LDA index needs at least two classes!")

  function(mat) {
    fit <- manova(mat ~ cl)

    1 - summary(fit, test = "Wilks")$stats[[3]]
  }
}
```

4.4. Writing a new display method

A display method is a list of four functions:

- `init(data)`. This function is called once, before all the other functions, and is typically used to determine scales.
- `render_frame()`. Create a new plot device with appropriate dimensions.
- `render_transition()`. On the Windows graphics device, it's much faster to reuse the existing plot, rather than creating a new one from scratch. This method should be used to clear the existing plot, typically by drawing a white rectangle over everything.
- `render_data(data, proj, geodesic)`: render the projected data. It is the responsibility of this method to project the data, typically by computing `data %*% proj`. This method should also draw axes, if possible

We only need these four functions because other infrastructure takes care of rendering to screen (using the most efficient method for that platform) and saving to disk.

5. Conclusions

In summary, this package is designed to make it easier to use and develop tour methods for high-dimensional data analysis. Natural extensions of the package would include new display types, interpolation algorithms, projection pursuit indices and optimization methods, and tours that help analyze large p, small n data. By treating the tour path like data, we can readily examine the coverage and patterns in the way that the algorithm operates on the space of all projections. This package provides exciting new possibilities for tour research.

Acknowledgements

This work has been partly supported by the National Science Foundation grant DMS0706949. Documentation for the package was written in **roxygen** by our undergraduate research assistant Barret Schloerke. The **tourr** package has extensive documentation and examples, generated with the roxygen package (Danenberg and Eugster 2008). **Roxygen** allows documentation to be placed inline, which reduces the distance between code and documentation, and makes it easier to keep the documentation. Bei Huang helped us to find and fix errors in the projection pursuit code.

References

Andrews DF (1972). "Plots of High-dimensional Data." *Biometrics*, **28**, 125–136.

Asimov D (1985). "The Grand Tour: A Tool for Viewing Multidimensional Data." *SIAM Journal of Scientific Statistical Computing*, **6**(128-143).

Buja A, Asimov D (1986). "Grand Tour Methods: An Outline." In DM Allen (ed.), "Proceedings of the 17th Symposium on the Interface between Computing Science and Statistics," pp. 63–67. Elsevier, Lexington, KY.

Buja A, Cook D, Asimov D, Hurley C (2004). "Theory of Dynamic Projections in High-Dimensional Data Visualization." *Electronic Journal of Statistics*. To appear, URL http://www-stat.wharton.upenn.edu/~buja/paper-dyn-proj-math.pdf.

Buja A, Cook D, Asimov D, Hurley C (2005). "Computational Methods for High-Dimensional Rotations in Data Visualization." In CR Rao, EJ Wegman, JL Solka (eds.), "Handbook of Statistics: Data Mining and Visualization," pp. 391–413. Elsevier/North Holland, http://www.elsevier.com.

Buja A, Hurley C, McDonald JA (1986). "A Data Viewer for Multivariate Data." In TJ Boardman, IM Stefanski (eds.), "Proceedings of the 18th Symposium on the Interface between Comput. Sci. and Statist.", pp. 171–174. Elsevier, Fort Collins, CO.

Carr DB, Nicholson WL, Littlefield RJ, Hall DL (1987). "Interactive Color Display Methods for Multivariate Data." In EJ Wegman, DJ DePriest (eds.), "Statistical Image Processing and Graphics," pp. 215–249. Marcel Dekker, Inc.

Carr DB, Wegman EJ, Luo Q (1996). "ExplorN: Design Considerations Past and Present." *Technical Report 129*, Center for Computational Statistics, George Mason University.

Cook D (2009). "Incorporating Exploratory Methods using Dynamic Graphics into Multivariate Statistics Classes: Curriculum Development." In MC Shelley II, LD Yore, B Hand (eds.), "Quality Research in Literacy and Science Education: International Perspectives and Gold Standards," pp. 339–358. Springer, Dordrecht, The Netherlands.

Cook D, Buja A, Cabrera J, Hurley C (1995). "Grand Tour and Projection Pursuit." *Journal of Computational and Graphical Statistics*, **4**(3), 155–172.

Cook D, Lee EK, Buja A, Wickham H (2008). "Grand Tours, Projection Pursuit Guided Tours and Manual Controls." In C Chen, W Härdle, A Unwin (eds.), "Handbook of Data Visualization," Springer Handbooks of Computational Statistics, chapter III.2, pp. 295–314. Springer.

Cook D, Swayne D (2007). *Interactive and Dynamic Graphics for Data Analysis with examples using R and GGobi*. Springer, New York. With contributions from Buja, A., Temple Lang, D., Hofmann, H., Wickham, H. and Lawrence, M. and additional data, R code and demo movies at http://www.ggobi.org.

Danenberg P, Eugster M (2008). *roxygen: Literate Programming in R*. R package version 0.1, http://roxygen.org.

Fox J (2002). "Frames, Environments, and Scope in R and S-Plus." http://cran.r-project.org/doc/contrib/Fox-Companion/appendix-scope.pdf.

Friedman J, Tukey J (1974). "A Projection Pursuit Algorithm for Exploratory Data Analysis." *IEEE Transactions on Computers C*, **23**, 881–889.

Huber P (1985). "Projection Pursuit." *The Annals of Statistics*, **13**, 435–475.

Huh MY, Kim K (2002). "Visualization of Multidimensional Data Using Modifications of the Grand Tour." *Journal of Applied Statistics*, **29**(5), 721–728.

Inselberg A (1985). "The Plane with Parallel Coordinates." *The Visual Computer*, **1**, 69–91.

Johnson RA, Wichern DW (2002). *Applied Multivariate Statistical Analysis (5th ed)*. Prentice-Hall, Englewood Cliffs, NJ.

Kirkpatrick S, Gelatt CD, Vecchi MP (1983). "Optimization by Simulated Annealing." *Science*, **220**, 671–680.

Lee EK, Cook D, Klinke S, Lumley T (2005). "Projection Pursuit for Exploratory Supervised Classification." *Journal of Computational and Graphical Statistics*, **14**(4), 831–846.

Lubischew AA (1962). "On the Use of Discriminant Functions in Taxonomy." *Biometrics*, **18**, 455–477.

Nelson L, Cook D, Cruz-Neira C (1999). "XGobi vs the C2: Results of an Experiment Comparing Data Visualization in a 3-D Immersive Virtual Reality Environment with a 2-D Workstation Display." *Computational Statistics*, **14**, 39–51.

Scott DW (1985). "Averaged Shifted Histograms: Effective Nonparametric Density Estimators in Several Dimensions." *The Annals of Statistics*, **13**, 1024–1040.

Scott DW (1995). "Incorporating Density Estimation into other Exploratory Tools." In "ASA Proceedings of the Section on Statistical Graphics," pp. 28–35. American Statistical Association., Alexandria, VA.

Swayne D, Temple Lang D, Cook D, Buja A, Wickham H, Lawrence M, Hofmann H (2001). "GGobi: Software for Exploratory Graphical Analysis of High-dimensional Data." *[Jul/01] Available publicly from* `www.ggobi.org`.

Swayne DF, Cook D, Buja A (1992). "XGobi: Interactive Dynamic Graphics in the X Window System with a Link to S." In "American Statistical Association 1991 Proceedings of the Section on Statistical Graphics," pp. 1–8. American Statistical Association, Alexandria, VA.

Swayne DF, Cook D, Buja A (1998). "XGobi: Interactive Dynamic Data Visualization in the X Window System." *Journal of Computational and Graphical Statistics*, **7**(1), 113–130.

Swayne DF, Lang DT, Buja A, Cook D (2003). "GGobi: Evolving from XGobi into an Extensible Framework for Interactive Data Visualization." *Journal of Computational Statistics and Data Analysis*, **43**, 423–444. `http://authors.elsevier.com/sd/article/S0167947302002864`.

Symanzik J, Wegman EJ, Braverman A, Luo Q (2002). "New Applications of the Image Grand Tour." *Computing Science and Statistics*, **34**, 500–512.

Tierney L (1991). *LispStat: An Object-Orientated Environment for Statistical Computing and Dynamic Graphics*. Wiley, New York, NY.

Wegman E, Luo Q (1997). "High Dimensional Clustering using Parallel Coordinates and the Grand Tour." *Computing Science and Statistics*, **28**, 352–360.

Wegman EJ (1990). "Hyperdimensional Data Analysis Using Parallel Coordinates." *Journal of the American Statistical Association*, **85**(411), 664–675.

Wegman EJ, Poston WL, Solka JL (2001). "Pixel Tours." In "IMA Workshop on Geophysics and Statistics," http://ima.umn.edu/talks/workshops/11-12-16.2001/.

Wickham H (2009). *ggplot2: Elegant Graphics for Data Analysis*. useR. Springer.

Wilhelm A, Wegman EJ, Symanzik J (1999). "Visual Clustering and Classification: The Oronsay Particle Size Data Set Revisited." *Computational Statistics*, **14**(1), 109–146.

Affiliation:

Hadley Wickham
Rice University
E-mail: hadley@rice.edu
URL: http://had.co.nz/

Dianne Cook
Iowa State University
E-mail: dicook@iastate.edu
URL: http://www.public.iastate.edu/~dicook/

Heike Hofmann
Iowa State University
E-mail: hofmann@iastate.edu
URL: http://www.public.iastate.edu/~hofmann/

Andreas Buja
Wharton School, University of Pennsylvania
E-mail: buja-at-wharton@gmail.com
URL: http://www-stat.wharton.upenn.edu/~buja/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd