Teaching Statistics using 3D Graphics in R

Duncan Murdoch[1]

[1] University of Western Ontario, London, Ontario, Canada
E-mail: murdoch@stats.uwo.ca

Keywords: Interactive graphics, singular value decomposition, Nelder-Mead algorithm

1 Introduction

The Department of Statistical and Actuarial Sciences at the University of Western Ontario offers four year honors undergraduate programs in statistics and actuarial science. At the introductory level (in the second year), there are typically 50 to 80 students in the classes. These students have a strong mathematics background, but their background in computing is quite varied, some having taken programming courses in high school or first year university, some having no programming experience at all.

The second year curriculum (counted in 78 hour, full year courses) contains 0.5 units of probability, 0.5 of introductory mathematical statistics, 1.0 of calculus, and 0.5 of computing. The actuarial science students (the majority) also take about 1.5 courses in that field. A normal course load is 5.0 courses per year, with various requirements on the elective courses that fill out the load.

Prior to 2005, our computing course emphasized problem solving using computer packages, but in that year we made a substantial revision to it to emphasize programming. Because of my involvement in the R Project (R Development Core Team (2008)) and the use of R in higher year courses, it was natural to base the course on R, and the first half of the course teaches basic programming using R as a general purpose language. The second half of the course applies this knowledge to Monte Carlo simulation, computational linear algebra, and numerical optimization. (The text Braun and Murdoch (2007) was written to match this syllabus, and the course follows it quite closely.)

Throughout the course the students are encouraged to think of programmatic solutions to problems, and to construct graphical displays to help their understanding. We teach the "classic" pen-on-paper graphics model that originated in S (Becker *et al.* (1988)); the goal is to get students to break down the steps involved in drawing a plot into smaller steps, and to encourage them to use visualization to understand their programs or the processes they are studying. In some cases the problems are naturally more than two dimensional, and I have found that the rgl package (Adler and Murdoch (2008)) allows relatively straightforward construction of interactive displays. Students construct graphical displays of data, simulations, and illustrations of geometric concepts.

In this paper I will describe two such displays used in the class. The first helps to study the singular value decomposition in computational linear algebra and the second illustrates the Nelder-Mead simplex algorithm in numerical optimization.

2 The Singular Value Decomposition

The singular value decomposition of a square $n \times n$ matrix A is

$$A = UDV^T \tag{1}$$

where U and V are $n \times n$ orthogonal matrices (i.e. $U^TU = V^TV = I$), D is an $n \times n$ diagonal matrix with non-negative entries, and the superscript T indicates matrix transposition. The students in our computing class have heard these definitions before in a first course in linear algebra, but lack intuition about their meaning. Our goal is to develop graphical methods to foster this intuition, and also to teach strategies for studying problems such as this.

We start by reminding students that matrices are representations of linear operators on vector spaces. That is, one way to look at the properties of A is to think of the effect of the multiplication $y = Ax$, where x and y are $n \times 1$ column vectors. As we vary x the vector y will also vary. The students already know how to determine A from its action on the basis vectors, so this is not new to them.

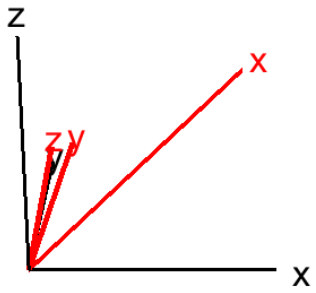

Figure 1: Illustrating the effect of a matrix on the basis vectors. The original basis vectors are shown in black, their images are shown in red.

We then use the rgl package to develop a graphical representation of 3×3 matrices. It is mathematically sufficient to show the images of the basis vectors, and we start with that display. For example, the code below produces the display shown in Figure 1.

```
> getwd()

[1] "D:/svn/papers/IASC2008"

> A <- matrix(c(1,2,0.1,0.1,1,0.1,0.1,0.1,0.5), 3,3)
> A

     [,1] [,2] [,3]
[1,]  1.0  0.1  0.1
[2,]  2.0  1.0  0.1
[3,]  0.1  0.1  0.5

> basis <- cbind(c(0,1,0,0,0,0),c(0,0,0,1,0,0), c(0,0,0,0,0,1))
> open3d()

[1] 1

> segments3d(basis, lwd=3)
> segments3d(basis %*% t(A), col="red", lwd=5)
> text3d(1.1*basis, texts=c("","x", "", "y", "", "z"), cex=2)
> text3d(1.1*basis %*% t(A), col="red", texts=c("","x", "", "y", "", "z"), cex=2)
```

(In these calculations, we multiply on the right by the transpose of A, because rgl normally works with row vectors arranged in a matrix rather than column vectors.)

In a live demonstration the figure is rotatable, but even so it is hard to visualize the overall effect of the transformation. We find that showing the images of more points gives a better impression of the display. We transform the surface of a coloured sphere[1], as shown in Figure 2.

```
> sphere <- subdivision3d(cube3d(color=rep(rainbow(6),rep(4*4^4,6))),depth=4)
> open3d()
```

[1] Actually, the shape generated below is only approximately a sphere; a more accurate description is that it is a partially inflated cube.

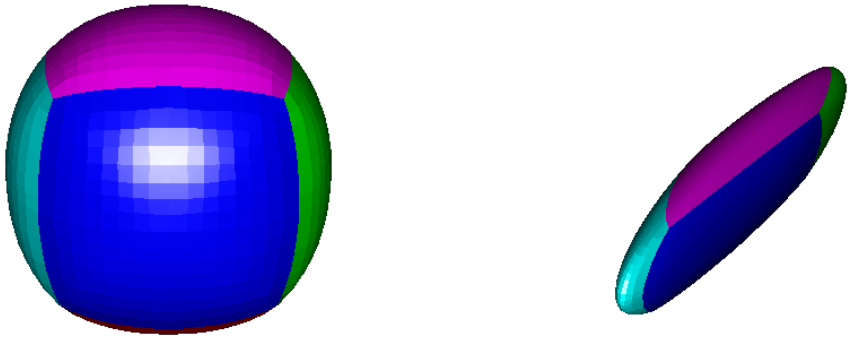

Figure 2: Illustrating the effect of a matrix acting on points on the surface of a sphere. Left: the original sphere. Right: the transformed points.

```
[1] 2
```

```
> shade3d(sphere)
> mult <- function(matrix, obj) transform3d(obj, t(matrix))
```

```
> open3d()
```

```
[1] 3
```

```
> shade3d(mult(A, sphere))
```

Even without the live rotation of the display, the effect of matrix multiplication is clearer. We use this display to understand the actions of orthogonal and diagonal matrices in the singular value decomposition (Figure 3).

```
> svd <- svd(A)
> U <- svd$u
> D <- diag(svd$d)
> V <- svd$v
> open3d()
```

```
[1] 4
```

```
> shade3d(mult(U, sphere))
```

```
> open3d()
```

```
[1] 5
```

```
> shade3d(mult(D, sphere))
```

```
> open3d()
```

```
[1] 6
```

```
> shade3d(mult(V, sphere))
```

In these static figures the student can easily see that U and V perform rotations and that D stretches the coordinate axes, but it is not so easy to tell the different roles of U and V. In class we use a TCL/TK (see http://www.tcl.tk) widget written using R's tcltk package to apply each of these transformations under the control of sliders, varying from the identity transformation smoothly to full application. The appendix contains the R code for this demonstration.

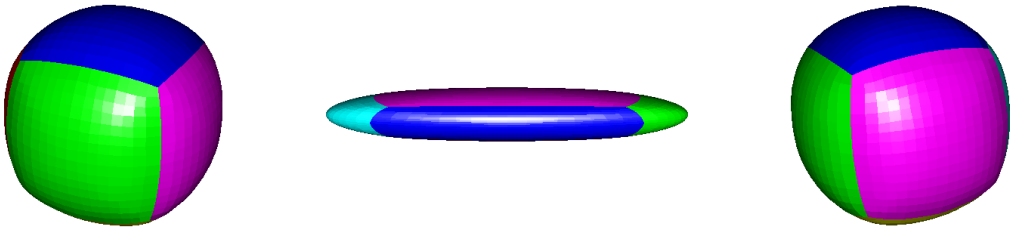

Figure 3: Illustrating the singular value decomposition. Left: the effect of U on the sphere. Middle: the effect of D. Right: the effect of V.

3 Nelder-Mead Optimization

The Nelder-Mead simplex method (Nelder and Mead (1965)) is a robust derivative-free multi-dimensional minimizer. This optimizer is easy to describe and to visualize, and implementations of it are within the reach of our introductory students. It is not very fast, and the visualizations help to illustrate why. Our demonstration is based on the implementation in R, which is adapted from Nash (1990) and more fully described in Braun and Murdoch (2007).

The algorithm works by constructing a non-degenerate simplex in the space of the arguments to the target function. It iterates through updates of the simplex until the simplex is determined to be close enough to a local minimum. (Most of the updates replace the vertex with the highest function value with a new one, either by shrinking, expanding, or reflecting the simplex through the centroid of the other vertices.)

We start with a simple function of two arguments. We have found it useful to display both a contour plot of the function and a perspective plot (Figure 4): this helps the students to recall the geometric interpretation of the contour plot.

```
> f <- function(x, y) ((x-y)^2 + (x-2)^2 + (y-3)^4)/100
> x <- seq(0,10,len=20)
> y <- seq(0,10,len=20)
> z <- outer(x, y, f)
> x <- seq(0,5,len=20)
> y <- seq(0,5,len=20)
> z <- outer(x, y, f)
> contour(x,y,z,xlab="x",ylab="y")

> showsurface <- function(x, y, z) {
+   persp3d(x,y,z, col="red", alpha=0.3, axes=F)
+   contours <- contourLines(x,y,z)
+   for (i in 1:length(contours))
+     with(contours[[i]], lines3d(x, y, level, col="darkred"))
+ }
> open3d()

[1] 7

> showsurface(x,y,z)
```

After showing a few steps of the Nelder-Mead algorithm (Figure 5), we proceed to three parameter functions. The goal in this example is to maximize the density of a three-dimensional, three-component normal mixture distribution, one of the examples in the misc3d package (Feng and Tierney (2008)). We

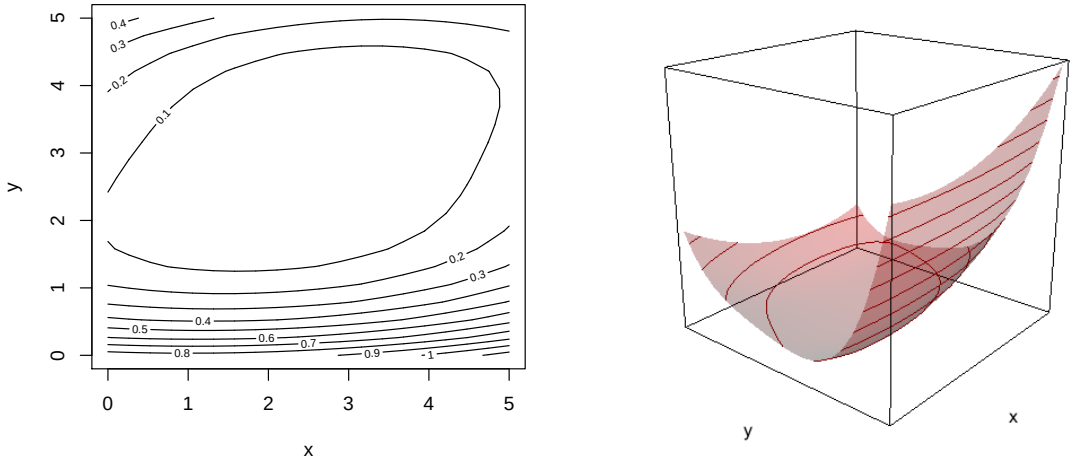

Figure 4: Two views of $f(x, y) = [(x - y)^2 + (x - 2)^2 + (y - 3)^4]/100$.

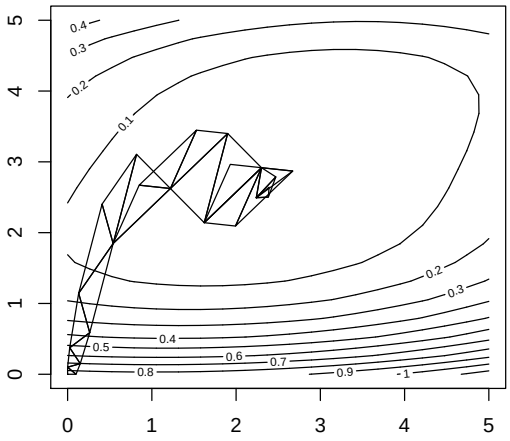

Figure 5: Twenty steps of the Nelder-Mead algorithm on $f(x, y)$.

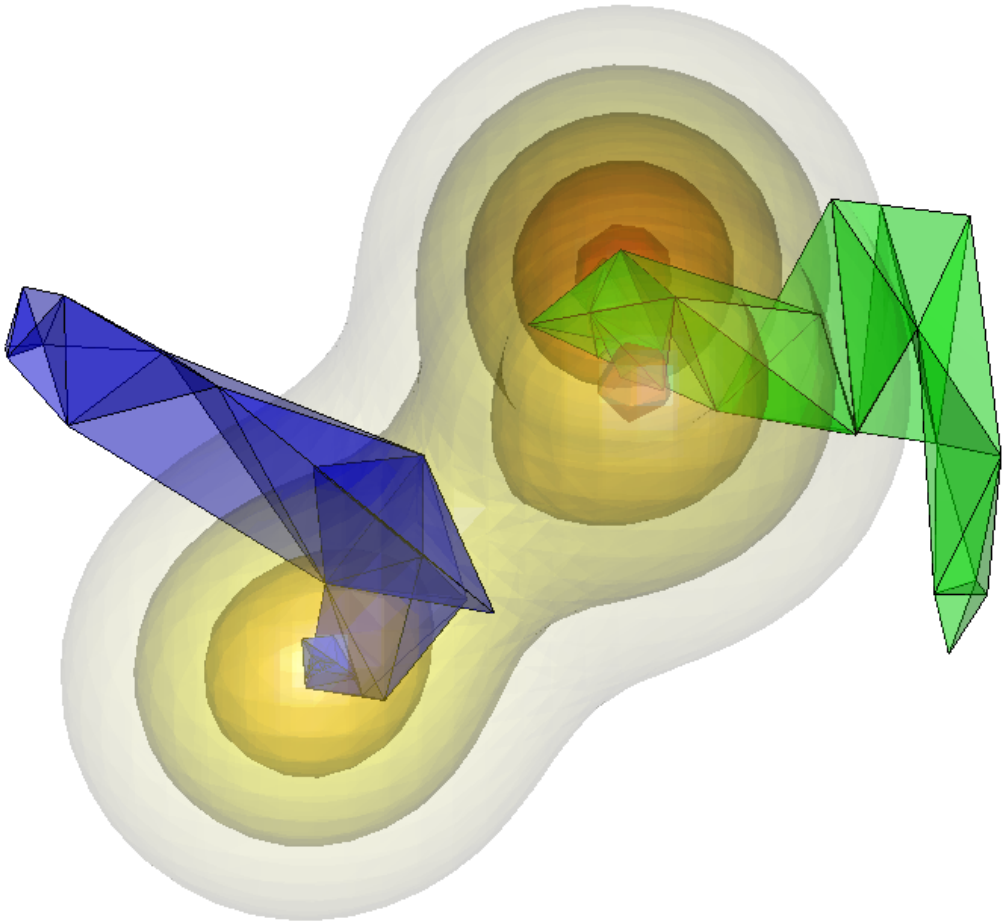

Figure 6: The Nelder-Mead algorithm in three dimensions, started from two different initial simplices (blue and green).

use that package to draw a three-dimensional contour plot, and show Nelder-Mead iterations starting from different starting points (Figure 6).

It is clear from Figures 5 and 6 that the Nelder-Mead algorithm is not particularly fast. It does not always move in the best direction, and it takes smaller steps than we (as omniscient viewers) can see it could. In addition, Figure 6 shows that it sometimes converges to a local extreme, rather than a global one. We use these limitations to motivate other optimizers.

4 Conclusion

Besides teaching about the singular value decomposition and the Nelder-Mead algorithm, we hope that these demonstrations show students that it is possible to generate relatively sophisticated graphics in a fairly easy way. The students are used to being computer users (as game players, etc.), but the idea that they can take control is one of the more valuable lessons that we hope they take away from our class.

Appendix 1: TCL/TK demonstration code for singular value decomposition

```
> library(tkrgl)
> makeSVDfn <- function(A) {
+   svd <- svd(A)
+   U <- rotationMatrix(matrix=svd$u)
+   D <- svd$d
+   Vt <- rotationMatrix(matrix=t(svd$v))
+   # A is U %*% diag(D) %*% Vt
+   interpU <- par3dinterp(c(0,1), userMatrix=list(identityMatrix(), U),
+                          extrapolate = "constant")
+   interpD <- par3dinterp(c(0,1), scale=list(c(1,1,1), D),
+                          extrapolate = "constant")
+   interpVt <- par3dinterp(c(0,1), userMatrix=list(identityMatrix(), Vt),
+                          extrapolate = "constant")
+
+   SVDfn <- function(upart, dpart, vpart) {
+     result <- interpU(upart)$userMatrix %*%
+          diag(c(interpD(dpart)$scale,1)) %*%
+          interpVt(vpart)$userMatrix
+     return(result[1:3,1:3])
+   }
+ }
> labels <- function(title="") {
+   axes3d(c('x','y','z'))
+   box3d()
+   title3d(xlab='X',ylab='Y',zlab='Z')
+   mtext3d(title,'x++',line=4)
+ }
> showSVD <- function(A) {
+   f <- makeSVDfn(A)
+   upart <- tclVar(100)
+   dpart <- tclVar(100)
+   vpart <- tclVar(100)
+
+   dev <- open3d()
+   shade3d(sphere)
+   labels("Partial")
+
+   setTransform <- function(...) {
+     upart <- as.numeric(tclObj(upart))/100
+     dpart <- as.numeric(tclObj(dpart))/100
```

```
+     vpart <- as.numeric(tclObj(vpart))/100
+
+     rgl.set(dev)
+     par3d(skipRedraw=TRUE)
+     clear3d()
+     shade3d(mult(f(upart, dpart, vpart), sphere))
+     labels("Transformed")
+     par3d(skipRedraw=FALSE)
+   }
+   tkbase <- tktoplevel()
+   tkwm.title(tkbase, "SVD")
+   spin <- spinControl(tkbase, dev)
+   uscale <- tkscale(tkbase, showvalue=FALSE, orient="horiz", from=0, to=100,
+                                  resolution=5, variable=upart, command=setTransform)
+   dscale <- tkscale(tkbase, showvalue=FALSE, orient="horiz", from=0, to=100,
+                                  resolution=5, variable=dpart, command=setTransform)
+   vscale <- tkscale(tkbase, showvalue=FALSE, orient="horiz", from=0, to=100,
+                                  resolution=5, variable=vpart, command=setTransform)
+   quit <- tkbutton(tkbase, text = "Quit", command = function() {
+       tkdestroy(tkbase)
+       rgl.set(dev)
+       rgl.close()
+       })
+   tkpack(spin, tklabel(tkbase, text="U"), uscale, tklabel(tkbase, text="D"), dscale,
+               tklabel(tkbase, text="V^t"), vscale, quit)
+ }
```

Appendix 2: One step of Nelder-Mead algorithm

```
> neldermead <- function(x, f) {
+   n <- nrow(x)
+   p <- ncol(x)
+
+   if (n != p + 1) stop(paste('Need', p + 1, 'starting points'))
+
+   fx <- rep(NA, n)
+   for (i in 1:n) fx[i] <- f(x[i,])
+
+   o <- order(fx)
+   fx <- fx[o]
+   x <- x[o,]
+   xmid <- apply(x[1:p,], 2, mean)
+   z1 <- xmid - (x[n,] - xmid)
+   fz1 <- f(z1)
+
+   if (fz1 < fx[1]) {
+     z2 <- xmid - 2*(x[n,] - xmid)
+     fz2 <- f(z2)
+     if (fz2 < fz1) {
+        cat('Accepted reflection and expansion, f(z2)=',fz2,'\n')
+        x[n,] <- z2
+     } else {
+        cat('Accepted good reflection, f(z1)=',fz1,'\n')
+        x[n,] <- z1
+     }
+   } else if (fz1 < fx[p]) {
```

```
+      cat('Accepted okay reflection, f(z1)=',fz1,'\n')
+      x[n,] <- z1
+   } else {
+     if (fz1 < fx[n]) {
+       x[n,] <- z1
+       fx[n] <- fz1
+     }
+     z3 <- xmid + (x[n,] - xmid)/2
+     fz3 <- f(z3)
+     if (fz3 < fx[n]) {
+       cat('Accepted contraction 1, f(z3)=',fz3,'\n')
+       x[n,] <- z3
+     } else {
+       cat('Accepted contraction 2,')
+       for (i in 2:n) {
+         x[i,] <- x[1,] + (x[i,] - x[1,])/2
+         cat(' f(z', i+2, ') = ', f(x[i,]), sep='')
+       }
+       cat('\n')
+     }
+   }
+   return(x)
+ }
```

References

Adler, D. and Murdoch, D.J. (2008). rgl: 3D visualization device system using OpenGL. R package version 0.81. `http://r-forge.r-project.org/projects/rgl`.

Becker, R.A., Chambers, J.M. and Wilks, A.R. (1988). *The New S Language.* Wadsworth and Brooks/Cole.

Braun, W.J. and Murdoch, D.J. (2007). *A First Course in Statistical Programming with R.* Cambridge University Press.

Feng, D. and Tierney, L. (2008). *misc3d: Miscellaneous 3D Plots.* R package version 0.6-1.

Nash, J.C. (1990) *Compact Numerical Methods for Computers. Linear Algebra and Function Minimisation.* Adam Hilger.

Nelder, J.A. and Mead, R. (1965). A simplex algorithm for function minimization. *Computer Journal* **7**, 308-313.

R Development Core Team (2008). *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing. ISBN 3-900051-07-0. `http://www.R-project.org`.