Appendix

The tableplots I have described in my dissertation are created in `R` 2.1.1. I use two primary functions to create a tableplot: `table.plot` and `cellgram`. These functions make use of functions in the `R` packages `grid` (Sarkar, 2005) and `lattice` (Murrell, 2005). There is also a small function, `gap.list`, that helps create partitions in tableplots; `gap.list` is used by `table.plot`.

The `cellgram` function

A tableplot is made up of *cellgrams* that appear in each cell of the tableplot. A cellgram is a square display that shows a symbol (or symbols) with area proportional to a value (or values). (See Section 3.1 for a detailed description.) For example, given values 0.9, 0.8, -0.7, 0.6, 0.5, and 0.4, the cellgram in Figure A.1 is produced by

```
cellgram(cell=c(0.9,0.8,-0.7,0.6,0.5,0.4),label.size=3,label=4).
```

A description of the parameters of `cellgram` appears in Table A.1.

Figure A.1. Example of a cellgram.

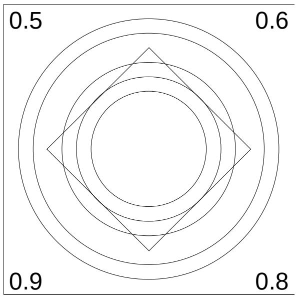


Table A.1. Description of parameters for cellgram.

Parameter	Description
`cell`	Value(s) to be plotted in each cell.
`shape`	Shape(s) of the symbol(s); 0 for circle, 1 for diamond, 2 for square. [default: `shape=0`]

`shape.col`	Color(s) of the symbol outline(s). [default: `shape.col="black"`]
`shape.lty`	Line type(s) of the symbol(s); see Murrell (2006, p. 61). [default: `shape.lty=1`]
`scale.max`	Denominator used to divide the value(s) of `cell`. [default: `scale.max=1`]
`cell.fill`	Color used to fill the interior of the symbol for the smallest (absolute) value in cell. [default: `cell.fill="white"`]
`back.fill`	Color used to fill the interior of the cell. [default: `back.fill="white"`]
`label`	Number of labels to appear in each cell; maximum is 4. If `cell` has more than four values and `label=4`, only the four largest values will be labeled. [default: `label=0`]
`label.size`	Size of text used for the labels in a cell; see Murrell (2006, p. 64) [default: `label.size=0.7`]
`ref.col`	Color of reference lines in a cell. [default: `ref.col="grey80"`]
`ref.grid`	To draw reference lines or not. [default: `ref.grid="no"`]
`shape.name`	To uniquely name the symbols in a cell. [default: `shape.name=""`]

The `table.plot` function

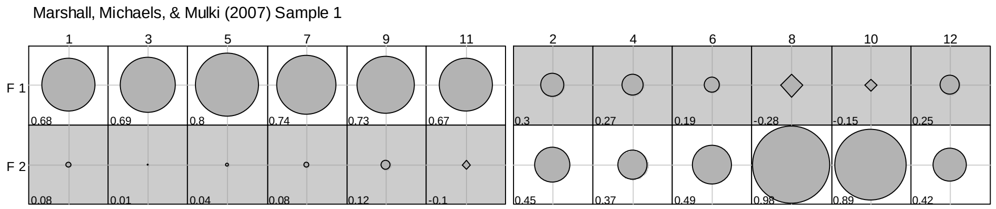
The `table.plot` function creates tableplots by drawing an arrangement of cellgrams. The appearance of each cellgram in a tableplot can be different. That is, one can designate a different set of parameters for each cellgram in a tableplot. This feature uses the parameters `matrix.2` and `patterns` of `table.plot`. For example, if the cellgrams are to have five different settings of parameters (i.e., five types of differently parameterized cellgrams), `patterns` should be assigned a list of five sublists. Each sublist entails values for the `cellgram` parameters `shape`, `shape.col`, `shape.lty`, `cell.fill`, `back.fill`, `label`, `label.size`, `ref.col`, `ref.grid`, `scale.max` as defined in Table A.1. As each cellgram is constructed, `table.plot` determines what parameter setting to use based on values of `matrix.2`, which is assigned a matrix with dimensions equal to that of the tableplot. Given five types of parameter settings, for example, the elements of `matrix.2` could be 1, 2, 3, 4, or 5, corresponding to the five types of parameter settings. If element (*i*, *j*) of `matrix.2` is 4, then the cellgram of cell (*i*, *j*) is drawn with parameter settings type 4.

In Section 3.3 I described the study by Marshall, Michaels, & Mulki (2007); I reanalyzed their sample 1 data using maximum likelihood EFA in CEFA followed by a target rotation to the hypothesized perfect cluster configuration. The resulting 12×2 factor pattern appears in Table A.2. The corresponding tableplot of this pattern (transposed) appears in Figure A.2.

Table A.2. Factor pattern of sample 1 from Marshall, Michaels, & Mulki (2007).

item	Factor 1	Factor 2
1	0.6835	0.0771
3	0.6913	0.0056
5	0.8037	0.0437
7	0.7435	0.0807
9	0.7334	0.1236
11	0.6698	-0.0996
2	0.2959	0.4506
4	0.2654	0.3741
6	0.1941	0.4879
8	-0.2835	0.9830
10	-0.1482	0.8863
12	0.2454	0.4164

Figure A.2. Tableplot of the sample 1 pattern from Marshall, Michaels, & Mulki (2007).



Scaled to maximum of |0.98|.

The `R` code that generates the tableplot in Figure A.2 appears as follows:

```
MMM1 = matrix(c(
     0.6835, 0.0771,
     0.6913, 0.0056,
     0.8037, 0.0437,
     0.7435, 0.0807,
     0.7334, 0.1236,
     0.6698,-0.0996,
     0.2959, 0.4506,
     0.2654, 0.3741,
     0.1941, 0.4879,
    -0.2835, 0.9830,
    -0.1482, 0.8863,
     0.2454, 0.4164), 12, 2, byrow=T)

B = matrix(1,2,12)
B[2,1:6]  = 2
B[1,7:12] = 2

table.plot(
      matrix.1 = t(round(MMM1,2)),
      title = "Marshall, Michaels, & Mulki (2007) Sample 1",
      top.label = ("1","3","5","7","9","11","2","4","6","8","10","12"),
      side.label = "F",
      v.parts = c(6,6),
      gap = 2,
      matrix.2 = B,
      patterns=list(
         list(0,"black",1,"grey70","white",1,0.7,"grey80","yes",0.98),
         list(0,"black",1,"grey70","grey80",1,0.7,"grey70","yes",0.98)
         )
      )
```

A description of the parameters of `table.plot` appears in Table A.3. The code for `cellgram` and `table.plot` appears after Table A.3.

Table A.3. Description of parameters for cellgram.

Parameter	Description
`matrix.1`	Values to be plotted. `matrix.1` can be a matrix or an array of three dimensions. For example, to plot three 12×2 factor patterns, `matrix.1` is assigned a 12×2×3 array of the factor pattern coefficients.
`matrix.2`	Matrix that designates the type of `cellgram` parameter settings to be used in each cell. The dimensions of `matrix.2` are same as the dimensions of the matrix assigned to `matrix.1` (or same as the first two dimensions of the array assigned to `matrix.1`).
`patterns`	A list of sublists; each sublist specifies the values for all parameters of `cellgram` other than `cell` and `shape.name` (see Table A.1). [default: `patterns=list(list(0, "black",1,"white","white",0,0.5,"grey80","no",1))`]
`title`	Title of tableplot. [default: `title="Tableplot"`]
`side.label`	Prefix used to create labels for each row; or a list of labels. [default:

	`side.label="row"`]
`top.label`	Prefix used to create labels for each column; or a list of labels. [default: `top.label="col"`]
`table.label`	To label rows and columns or not. [default: `table.label="yes"`]
`label.size`	Size of text used for the row and column labels; see Murrell (2006, p. 64). [default: `label.size=0.8`]
`gap`	Size (in millimeters; see Murrell, 2006, p. 99) of the gap(s) that appear between submatrices of the tableplot if a partitioned tableplot is selected. [default: `gap=2`]
`v.parts`	List of numbers corresponding to the subsets of columns if partitions between columns are desired. For example, to partition a tableplot with 12 columns into four sets of three columns, specify `v.parts=c(3,3,3,3)`. [default: `v.parts=0` creates no column partitions]
`h.parts`	Analogous to `v.parts`, but for creating partitions between rows of the tableplot. [default: `v.parts=0` creates no row partitions]
`cor.matrix`	If the tableplot is a correlation matrix or not. If `cor.matrix="yes"` then diagonal elements of the tableplot are filled with labels (specified in `var.names`), not cellgrams. [default: `cor.matrix="no"`]
`var.names`	If `cor.matrix="yes"` then `var.names` specifies the prefix used to create labels for the diagonal elements of the tableplot; or a list of labels. [default: `var.names="var"`]

```
cellgram = function(

          ## Arguments that may be vectorized:

          cell,                 # actual cell value(s)
          shape=0,              # shape of cell value(s); 0=circle, 1=diamond
          shape.col="black",    # color of shape(s), outline only
          shape.lty=1,          # line type used for shape(s); see page 61

          ## Arguments that will never be vectorized:

          scale.max=1,
          cell.fill="white",    # fill color of smallest cell value
          back.fill="white",    # back fill color
          label=0,              # how many cell values will be printed; max is 4
          label.size=0.7,
          ref.col="grey80",
          ref.grid="no",
          shape.name="")        # uniquely identify shapes to help fill in smallest one

          {
          grid.rect(gp=gpar(fill=back.fill, lwd=0.2))
          if (length(cell)>length(shape))     shape=rep(shape, length(cell))
          if (length(cell)>length(shape.col)) shape.col=rep(shape.col, length(cell))
          if (length(cell)>length(shape.lty)) shape.lty=rep(shape.lty, length(cell))

          ## Draw grid reference lines:

          if (ref.grid=="yes") {
                    grid.segments(x0=0,y0=.5,x1=1,y1=.5, gp=gpar(col=ref.col, lwd=0.2))
                    grid.segments(x0=.5,y0=0,x1=.5,y1=1, gp=gpar(col=ref.col, lwd=0.2))
                    }

          ## Rescale cell values:

          s.cell = cell / scale.max

          ## Draw cell values:

          for (k in 1:length(cell)){

                    if (!is.na(cell[k])) { ## This is to allow missing values; but if all missing, then error ensues!

                              #if (cell[k] < 0) this.col="red" else this.col=shape.col[k]
                              this.col = shape.col[k]

                              #if (cell[k] < 0) this.lty=3 else this.lty=shape.lty[k]
                              this.lty = shape.lty[k]

                              if (cell[k] < 0) this.shape = 1 else this.shape=shape[k]

                              if (this.shape==0)
                                        grid.circle(name=paste(shape.name,k,sep=""),
                                                  r=abs(s.cell[k]/2),
                                                  gp=gpar(col=this.col, lty=this.lty, lwd=0.1))

                              if (this.shape==1) {
                                        r1 = 0.5 - 0.5*abs(s.cell[k])
                                        r2 = 0.5*abs(s.cell[k]) + 0.5
                                        grid.polygon(name=paste(shape.name,k,sep=""),
                                                  x=c(r1, .5, r2, .5), y=c(.5, r2, .5, r1),
                                                  gp=gpar(col=this.col, lty=this.lty, lwd=0.1)) }

                              if (this.shape==2)
                                        grid.rect(name=paste(shape.name,k,sep=""), height=abs(s.cell[k]),
                                                  width=abs(s.cell[k]), gp=gpar(col=this.col, lty=this.lty))
                    }}

          grid.edit(paste(shape.name,which.min(abs(cell)),sep=""), gp=gpar(fill=cell.fill))

          ## Labels
          if (label > 0){
                    cell = sort(cell,decreasing=T)
                    d = matrix(c(0,1,1,0,0,0,1,1),4,2)
                    for (k in 1:min(label,4,length(cell))){
                              grid.text(cell[k], gp=gpar(cex=label.size),
                                  x=unit(c(.02,.98,.98,.02)[k],"npc"),
                                  y=unit(c(.02,.02,.98,.98)[k],"npc"), just=d[k,])
                              }
                    }
          }
```

```
table.plot = function(

          matrix.1,         # Matrix of values; can be a matrix, or an array of 3 dimensions.
          matrix.2,         # Matrix of pattern designations.
          patterns = list(list(0, "black", 1, "white", "white", 0, 0.5, "grey80", "no", 1)),

          title="Tableplot",

          side.label="row", # Or provide actual list of labels for each row.
          top.label="col",  # Or provide actual list of labels for each column.
          table.label="yes",# To have labels around the table or not.
          label.size=0.8,   # Size of side/top labels.

          gap=2,            # Width of gaps in partition, if there are partitions.
          v.parts=0,        # Column clusters; if provided, sum must equal number of columns.
          h.parts=0,        # Row clusters; if provided, sum must equal number of rows.

          cor.matrix="no",  # For a correlation matrix.
          var.names="var",  # Or provide a list of variable names.

          ...){

          grid.newpage()

          #---Create labels for a correlation matrix, if no names provided.

          if ((cor.matrix=="yes") && (length(var.names)==1)) var.names = paste(var.names,1:dim(matrix.1)[1])

          #---Create the pattern designation matrix when there is just one pattern.
          #---Otherwise, the pattern designation matrix is supplied by user.

          if (length(patterns)==1) matrix.2 = matrix(1, dim(matrix.1)[1], dim(matrix.1)[2])

          #---Add on extra dimension to matrix.1 if matrix.1 only has two dimensions.
          if (length(dim(matrix.1))==2) {dim(matrix.1) = c(dim(matrix.1)[1], dim(matrix.1)[2], 1)}

          #---Constructing vectors of gaps if partitions provided.

          v.gaps = gap.list(partitions=v.parts, x=dim(matrix.1)[2])
          h.gaps = gap.list(partitions=h.parts, x=dim(matrix.1)[1])

          #---Constructing labels, if no specific labels provided for each row/column.

          if (length(side.label)==1) side.label = paste(side.label, 1:dim(matrix.1)[1])
          if (length(top.label)==1)  top.label  = paste(top.label,  1:dim(matrix.1)[2])

          #---Create Layout 1 and write main title.

          L1 = grid.layout(2,1,heights=unit(c(3,1),c("lines","null")))
          pushViewport(viewport(layout=L1, width=0.95, height=0.98))

          pushViewport(viewport(layout.pos.row=1))                              ## Push row 1 of Layout 1.
          grid.text(title, x=0.02, just=c("left", "bottom"))
          upViewport()

          #---Create Layout 2.

          L2 = grid.layout(1,2,widths=unit(c(1,1),c("char","null")))
          pushViewport(viewport(layout.pos.row=2, layout=L2))                   ## Push row 2 of Layout 1.

          #---Create Layout 3.

          L3 = grid.layout(dim(matrix.1)[1],dim(matrix.1)[2],respect=T,just=c("left","top"))
          pushViewport(viewport(layout.pos.col=2))                              ## Push col 2 of Layout 2;

          #---Push Layout 3, but with adjustments to accomodate possible partitions.

          pushViewport(viewport(layout=L3, x=0, y=1,
                                    just=c(0,1),
                                    width =unit(1,"npc")-unit(gap,"mm")*(length(v.parts)-1),
                                    height=unit(1,"npc")-unit(gap,"mm")*(length(h.parts)-1)))

          #---Draw cellgrams.

          for (i in 1:dim(matrix.1)[1]){
                    for (j in 1:dim(matrix.1)[2]){

                              pushViewport(viewport(layout.pos.row=i, layout.pos.col=j))
                              pushViewport(viewport(just=c(0,1), height=1, width=1,
                                                                      x=unit(gap,"mm")*v.gaps[j],
                                                                      y=unit(1,"npc")-unit(gap,"mm")*h.gaps[i]))

                              pattern = patterns[[matrix.2[i,j]]]

                              if ((cor.matrix=="yes") && (i==j))

                              {
                               grid.rect(gp=gpar(fill="grey90"))
                               grid.text(var.names[i],gp=gpar(cex=0.5))
                              } else

                                    cellgram(cell    = matrix.1[i,j,],
                                           shape     = pattern[[1]],
                                           shape.col = pattern[[2]],
                                           shape.lty = pattern[[3]],
                                           cell.fill = pattern[[4]],
                                           back.fill = pattern[[5]],
                                           label     = pattern[[6]],
                                           label.size= pattern[[7]],
                                           ref.col   = pattern[[8]],
                                           ref.grid  = pattern[[9]],
                                           scale.max = pattern[[10]],
                                           shape.name= paste(i,j))

                              ##grid.rect()

                              if ((j==1) && (table.label=="yes")) {grid.text(side.label[i], x=-0.04, just=1,
                              gp=gpar(cex=label.size))}

                              if ((i==1) && (table.label=="yes")) {grid.text(top.label[j],  y=1.05, vjust=0,
                              gp=gpar(cex=label.size))}

                              upViewport()
                              upViewport()
                              }
                    }
          popViewport(0)
          }

## A function to construct a gap list.

gap.list = function(partitions=0,x){
          if (length(partitions)==1) rep(0,x) else {
          rep(1:length(partitions), partitions)-1}
          }
```