High-Level Software for Networks and Data Analysis

Duncan Temple Lang
Statistics

Outline

- Introduction to R.
- Comparison of high-level and lower-level software systems.
- R & Graphs and visualization.
- Dynamic, interactive visualization on graphs.

Introduction to R

- Intepreted language and environment for data analysis, simulation, and general computing.
- S language developed over many years at Bell Labs
- Licensed to S-Plus/Insightful and commercially marketed as S-Plus.
- R is an Open Source project which is not "unlike" S, but quite different internally.
- In 1998, John Chambers (Bell Labs) won the ACM Software Award for S.

- R is freely available - no cost, open source.
- Works on most platforms, including Unix, OS X, Windows.
- R is
  - a language
  - an interpreter
  - a collection of packages providing extensive statistical functionality.
- R provides a vast collection of add-on packages contributed by the user community.

Repositories

- About 800 contributed packages.
- http://cran.r-project.org - CRAN
- http://www.bioconductor.org
- http://www.omegahat.org
- Several others.

Functional Language

- Functional language - "no" side effects.
  - Easier to understand code and debug
  - No references, pass by value => copies.
  - This is an issue when dealing with graphs and nodes.
  - When we modify an object, we must reassign it.
  - Can use lexical scoping/mutable state via environments, ...

- Basic data structure is a vector
  No scalar values <=> vector of length 1
- Vector is an ordered set of homogeneous element types.
- Create using c() for concatenate
- c(1.2, 3.6, c(1, 2))
  c(TRUE, FALSE,
- Basic data types are
  numeric, integer, logical, character.
- R coerces to common type.

- Can index a vector in various convenient ways:
  - by position: x[2], x[c(3, 5, 7)], x[1:3]
  - by omission/negation: x[-2], x[-c(3, 5, 7)]
    (can't mix negation and inclusion)
  - logical mask: x[c(TRUE, FALSE, FALSE, TRUE)]
  - by name: x = c(a = 1, b = 2, y = 3.4)
    x[c("a", "b")], x["y"]

- Can also assign to subsets using the same notation
  x[c("a", "b")] <- c(10, 15.3)
  x[c(1, 2)] <- c(10, 15.3)
  x[c(TRUE, TRUE, FALSE)] = c(10, 15.3)
- Matrices and multi-dimensional arrays are basically vectors with an attribute giving the dimensions.
- So matrices have homogeneous types of elements.
- Sparse matrix support provided by SparseM package available on CRAN.

Lists

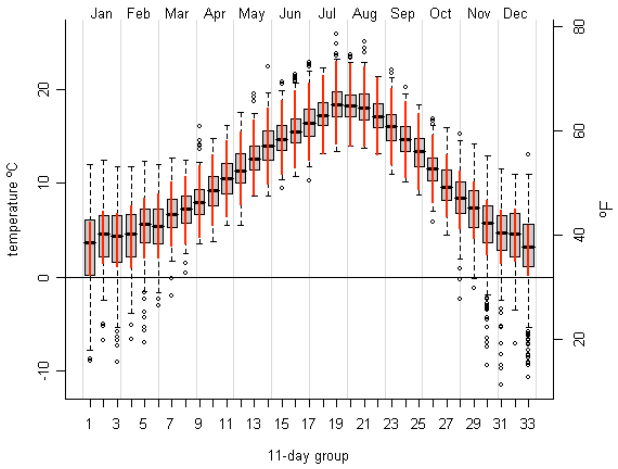
- For collecting objects that are not of the same type, need an additional data structure - a list.
- list(a = 1, b = c("x", "y"), matrix(1:9, 3, 3))
- Ordered and can have names.
- Subsetting using [works in the same way as for vectors,
- BUT returns an object of the same type as being subsetted, i.e. a list.
  l[c(1, 2)]
- To get individual element, use [[, i.e. l[[1]], l[["a"]]

Objects & Search Path

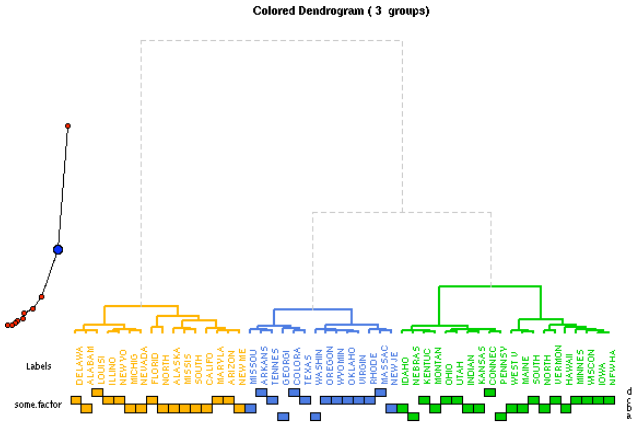
- R has a workspace for your session.
- Assignments made at the top-level are stored by name in this environment.
- Can discover current variables using objects().
- Can remove objects via, e.g. rm(x, y)
- When we bring in addtional packages, we use the command library().
- This ads the workspace for the package to the search path in which R looks for the variables mentioned in a calculation.
- search()

R Graphics

- R has rich graphics functionality.
- 2 basic systems
  - grz - regular graphics
  - grid graphics for ultra fine control.
- Usual types of plots built-in, and many more specialized plots available in add on packages.
- Static, presentation quality graphics with many, many controls.




```
require(seas)
data(mscdata)
par(cex=0.8)
plot.seas.temp(mscdata,id="1108447",add.alt=TRUE,style=c(0,1))
```


```
require(fpc)
require(A2R)

d.usa <- dist(USArrests, "euc")
h.usa <- hclust(d.usa, method="ward")

set.seed(1)
some.factor <- letters[1:4][rbinom(50, prob=0.5, size=3)+1]

hubertgamma <- sapply(1:10,
                      function(i)
                       cluster.stats(d.usa, cutree(h.usa, k=i+1),
                              G2 = FALSE, G3 = FALSE,
                               silhouette = FALSE)$hubertgamma
}
A2Rplot(h.usa, k=3,  fact.sup=some.factor, criteria=hubertgamma,
       boxes = FALSE,
       col.up = "gray",
       col.down = c("orange","royalblue","green3"))
```

Debugging - recover()

- Add
  options(error = recover)
  to your session
  e.g. via the file ~/.Rprofile (read on startup)
- When an error is encountered, you are placed in an interactive debugging environment.
- Can move around the different call frames by selecting a number,
- view the available objects() and their values using the usual R commands
- Exit call frame with empty command

Profiling

- R is interpreted and so not as fast as compiled code. (Although a lot of commands rapidly use native routines in R or BLAS, ATLAS, etc.)
- When a script or function is slow, we can profile it to determine where the bottlenecks are.
- Rprof("filename")
  Run our commands
  Rprof(NULL)
- After ending the collection of profile information, can examine it via summaryRprof("filename")
- Returns a data structure that we can manipulate directly in R

High-level Languages

- Compiled languages: C, C++, Objective C, Fortran.
- Java is compiled and runs on a Virtual Machine (VM)
- Both types of compiled languages require
  - explicit type specification.
  - application to be completed before running
  - recompilation before re-testing.

Higher Level Languages

- Perl & Python
- General languages with an interpreter.
- Both are general purpose languages, but with no particular focus on
  - numerical computation,
  - graphics.
- Numerous add on modules, some of which are for numerical computation.
- Graphics is brought in via various different types of extensions.

Perl & Python

- Perl is best suited for text manipulation.
  Regular expressions builtin to language.
- Language is succinct and "cute", but resulting code can be difficult to maintain.
- Object-oriented system somewhat ad hoc.
- Python is very general purpose, less focused on text manipulation.
- Object system at the core of the language.

R & Matlab

- S (R & S-Plus) is focused on statistics and data analysis.
- Matlab is focused more on engineering.
- The common intersection is linear algebra (matrices) and graphics.
- Both systems are general purpose programming languages and so can be "used" to do just about anything any other language can do.
- Both are vectorized and there is a benefit to using this.
- Both can readily integrate native/compiled code such as C & Fortran to make use of existing software and improve speed.

R & Matlab

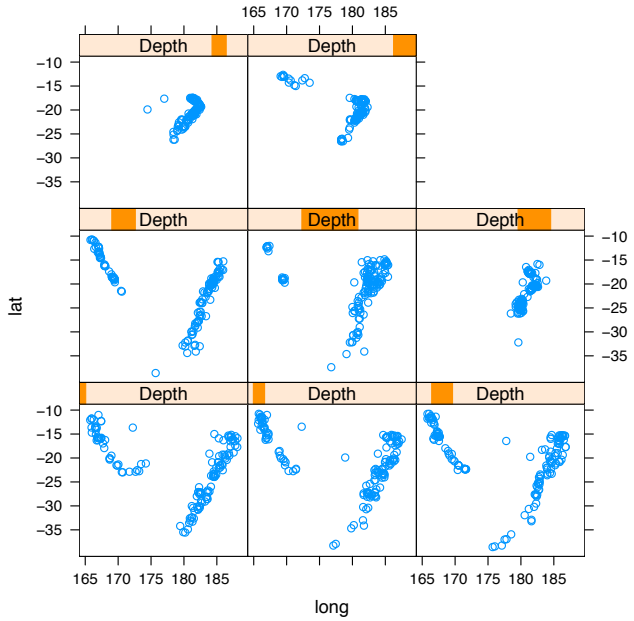
- In Matlab, one tends to write the calculations for doing a computation, e.g. $(X'X)^{-1}X'Y$
- In R, we make extensive use of symbolic representation of models via formula.

  response ~ log(age) + height + gender
- Then, we can fit this model using different methods without having to expose the underlying calculations to the user.

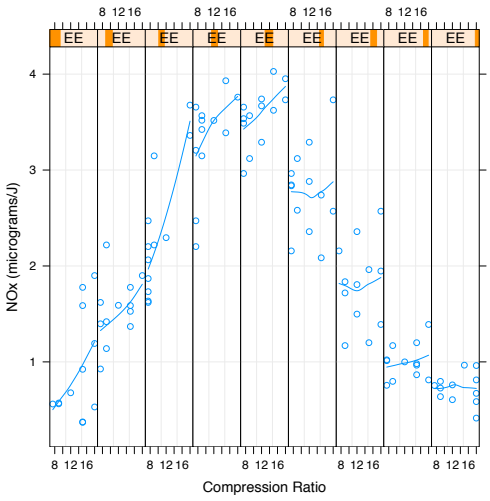
```
lm( f, data = data1)
lm( f, data = data2)
glm(f, data = data1)
```


Formula Language

- Note that the formula is an object in R, not just syntax.
- Formula is symbolic and independent of the data.
- Details about expanding categorical data in constructing the design matrix (X) (i.e. contrasts) are orthogonal to the formula, and specified when fitting the model.
- Additionally, we use the formula language for specifying plots.
- plot(y ~ x)
  histogram(x | gender)




```
Depth <- equal.count(quakes$depth, number=8, overlap=.1)
xyplot(lat ~ long | Depth, data = quakes)
```


```
EE <- equal.count(ethanol$E,
number=9, overlap=1/4)
xyplot(NOx ~ C | EE, data = ethanol,
       prepanel = function(x, y
        prepanel.loess(x, y, span=1),
       xlab = "Compression Ratio",
       ylab = "NOx (micrograms/J)",
       panel = function(x, y) {
           panel.grid(h=-1, v= 2)
           panel.xyplot(x, y)
           panel.loess(x,y, span=1)
       },
       aspect = "xy")
```

R & Matlab

- Matlab's object system is like S's older OOP system dynamic and loosely structured.
- R has both S3 and S4 class systems
- S3 is a dynamic system with no formal definition of a class.
  class(x) <- c("A", "B") - is an A and inherits from B.
- Get single dispatch, i.e. foo(x, y) finds appropriate method for foo based on type of x.
- S4 system uses explicit declarations for classes and methods. Quite different from C++ or Java or Python.

R

- R has a rich and powerful package mechanism.
- Used to easily publish software, and easily install on user's own machine.
- install.packages()
- Searches repositories on Web (or local) and can fetch all dependencies as part of the download and installation.
- Relatively easy to go from writing code interactively to writing functions to putting them in a package.

- If you are using R to do your class project, an R package is a convenient packaging medium.
- DESCRIPTION file
- R code in R/ directory
- Help pages for functions and data in Rd/ directory, written using prompt() and LaTeX-like markup.
- R CMD INSTALL myPackage
  or install.packages("path/to/package", repos = NULL)
- R CMD check myPackage
  to verify that it is "correct"

- Additionally, Sweave is an authoring tool for creating "dynamic" reproducible documents.
- One puts the code in the document, not the output of the R commands.
- One can then generate different views by processing the workflow.
- Can parameterize the computations with new inputs, e.g. data sets, parameters for simulations.

Stat. Software for Graphs

- We'll look at software for working with graphs.
- R for creating, manipulating and applying algorithms to graphs
  graph and RBGL packages from BioConductor
  www.bioconductor.org
- Rgraphviz and GGobi and rggobi for displaying graphs.
- Rgraphviz is for "static" displays of graph structures.
- GGobi is for dynamic, interactive displays of data and graph structures, as well as data associated with graphs (i.e. on nodes and edges)

More Information

- BioConductor monograph
  Bioinformatics and Computational Biology Solutions with R and BioConductor.
  Gentleman, Carey, Huber, Irizary, Dudoit
  Chapters 19, 20, 21.
- Vignettes
  e.g. vignette("Rgraphviz")
- R News

Graphs in R

- install.packages("graph", depend = TRUE,
  repos = "http://www.bioconductor.org")
  library(graph)
- Now we can create graphs.
- g = randomGraph(letters[1:10], 1:4, .5)
- class(g)
  [1] "graphNEL"

- Given a graph, can display it using R's graphics and Rgraphviz's layout

  plot(g)

- Different layout algorithms available to us
  - dot - hierarchy
  - neato, fdp - spring layout
  - twopi
  - circo
- plot(g, "neato")

graphNEL

- Nodes and Edge list graph.
- Can be directed or undirected.
- new("graphNEL", nodes = c("a", "b", "c"),

  edgeL = list(a = list(edges = "b"),

  b = list(edges = "b"),

  c = list(edges = c("a", "b"))),

  edgeMode = "directed")
- Edge mode can be "directed" or "undirected".
- Edges can also have numeric weight values.

Creating Graphs

- Adjacency matrix
  square matrix, with 1's and 0's indicating whether a pair of nodes is connected or not.
- row and column names identify the nodes.
- m = matrix(rbinom(25, 1, .2), 5, 5,
  dimnames = list(letters[1:5], letters[1:5]))
- as(m, "graphNEL") - coercion method
- Or new("graphAM", adjMat = m, edgemode = "directed")

Different Graph Classes

- From Sparse matrices to graphs via SparseM package on CRAN.
- clusterGraph - a graph made up of complete but disjoint subgraphs.
- distGraph - a complete graph, where the inter-node distances give the weights for the edges

Operations on Graphs

nodes, edges, acc, adj

- nodes(g) & edges(g)
  return the nodes and the edges respectively.
- Often need to identify edges by name, so use edgeNames()
- adj(g, c("nodeID", "otherID"))
  get a list with each element identifying which nodes are adjacent to that particular node.
- acc(g, c("nodeID", "otherID"))
  a list with each element identifying which nodes are reachable from this one and the length of the shortest path between them.

Graph Ops. degree()

- degree(g)
  returns the in and out degree values for each node for a directed graph
  and for undirected graphs, the simple degree distribution.
- Connected components of a graph are obtained via
  connComp(g)
  This returns a list of the different disjoint subcomponents.

subGraph

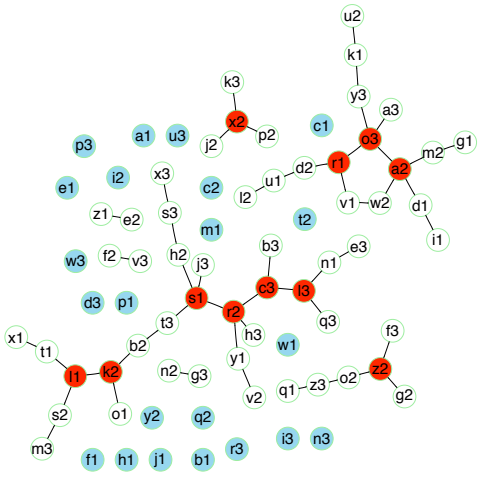
- ugraph(g) gives the undirected graph. Again, a copy.
- subGraph(c("nodeID", "otherID", "yetAnother"), g) yields the subgraph
- This is essentially a copy of the relevant nodes and edges. It is not done via references.
- union(), intersection() and complement give new graphs

Editing the Graph

- addNode(g, c("nodeID", "otherID", ...))
- addEdge(g,
- removeNode("nodeID", g)
  removeEdge("node1", "node2", g)
- combineNodes(c("node1", "node2"), g)
  amalgamates the two nodes into 1
- clearNode("nodeID", g)
  removes edges to and from that node.

Controlling the Plotting

- The simple plot(graph, "layout") command gives convenient results.
- However, we often want to control different features of the plot, e.g.
  - node labels
    shape,
    color,
    fill color
  - edge color,
    label
    line type,
    etc.




```
myNodes = as.character(outer(letters, 1:3, paste, sep = ""))
ga = randomEGraph(myNodes, edges = 50)
plot(ga, "neato")


# Create a character vector with value "red"
# and names identifying the nodes with degree >= 3
hiNodes = nodes(ga)[degree(ga) >= 3]
cols = structure(rep("red", length(hiNodes)),
                 names = hiNodes)
lowNodes = nodes(ga)[degree(ga) == 0]
lowNodes = structure(rep("skyblue", length(lowNodes)),
          names = lowNodes)

plot(ga, "neato", cex = .2,
      attrs = list(node = list(color = "lightgreen")),
      nodeAttrs = list(fillcolor = c(cols, lowNodes)))
```

- Specify graph and layout type.
- Then can specify a collection of global attributes.
- And then node and edge specific settings, i.e. for particular nodes and edges.
- The settings are merged together, with the most specific for a given node used.
- Need only specify the ones you want to change. These are merged with the values returned from getDefaultAttrs()
- Settings for graph, cluster, node and edge.

- Use getDefaultAttrs() to find out what attributes may be set for the different levels.
- For edges, need to identify the edge for which a setting is intended.

Use edge name, in form "src~dest"

e.g.

```
plot(g, edgeAttrs = list(label = c("a~b" = "ssh"),
                         col = c("a~b" = "red"))
```

Only Layout

- The simple plot() command does both layout and rendering.
- If we want to use the layout information in multiple situations, or simply do calculations on it, we can separate the two steps.
- agopen() does the layout and returns an instance of the "Ragraph" class.

Accessing the Layout

- l = agopen(g, layoutType = "neato", name = "")
plot(l)
- Can also do the plotting ourselves using R's own graphics tools.
- The layout object has lists of nodes and edges and we can access these via AgNode() and AgEdge().
- Can then get the center of each nodes, get its coordinates, etc.
- This allows us then to entirely control what is drawn, delegating graphviz to layout, and R's graphics to high quality rendering in different formats.

User Defined Node Rendering

- plot(layout, drawNode = function(node) ...)
- The function can draw whatever it wants, however it wants to using many different sources of information.
- Can even produce an Image Map using imageMap()
- Using RGtk, tcltk or soon wxWindows, we can build interactive tools for working with graphs.

RBGL

- Vincent Carey provides an interface to the Boost Graph library, a C++ collection of algorithms by Siek et al.
- Can pass a graph object from any of the types in the graph package to any of these algorithms.

RBGL functions

- Algorithms currently include
  - Traversal: Depth and Breadth first searches (dfs, bfs) return the visited nodes in order.
  - Shortest Paths: sp.between, djkstra.sp, bellman.ford.sp, dag.sp, johnson.all.pairs.sp
  - Minimal Spanning Trees: mstree.kruskal.
  - Connectvity:
  - Max. Flow Algorithms.

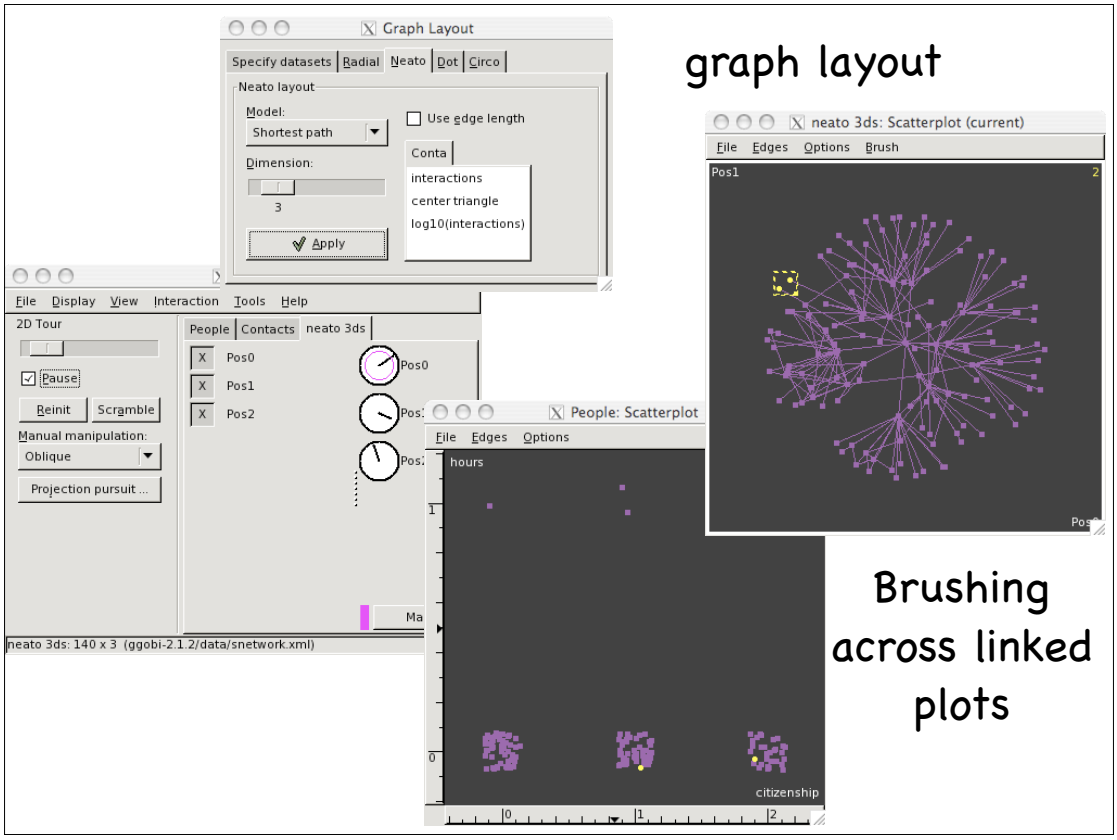
Temporal Graphs

- We can look at the structure of a graph by doing different layouts.
- We can even see how that structure changes over time.
  - layout the union of the entire collection of graphs
  - color only the edges (and nodes) that are present for a given "time" period.
  - Use animation or interactive controls to change "time".
- Can build such a GUI in R using RGtk or tcltk.

Data on Graphs

- Generally, we are not just interested in a graph and its structure only.
  Rather, we have data for each node, and potentially observations on the edges.
- E.g. computer network we have information about the operating system on each computer, its users, login sessions, files, etc.
- Edges: connections between machines on the network have information about the ports, the length of the session, number of bytes, etc.

- GGobi for interactive, dynamic graphics with spport for graphs
- Open Source, freely available from http://www.ggobi.org
- Also, connection with R via rggobi.
- And can program own link actions via RGtk and rggobi.



Statistics for Graphs

- Distributions for measures on graphs to determine whether they are stochastically similar.
- Probabilities of detecting edges
  false positives, false negatives and missing data.