Short Tutorials for Metagenomic Analysis

Introduction

These tutorials provide an introduction to metagenomic analysis using matR (Metagenomic Analysis Tools for R). Each tutorial is short, about 10-15 minutes. They form a progressive set, but each is pretty separate from the others, too. This document is written at a moderate (neither low nor high) technical level.

Contact us: `mg-rast@mcs.anl.gov`.

Contents

1 Preliminaries

1.1 Obtaining and Installing R

- R is free software, easily downloaded from the R Project Homepage: `http://www.r-project.org`. Binary versions are available for Mac and Windows systems, and source code for Linux. Download and install the version appropriate for your system.

  Users who already have R should *update their version.* R and its extensions are frequently updated. Keeping current is important to avoid nuisance errors.

- Add-on packages for many purposes, contributed by many people, are a great strength of R. For example, see this list of packages, organized by application area: `http://cran.r-project.org/web/views/`.

  For a repository dedicated entirely to biological functionality, see:
  `http://www.bioconductor.org`.

- Now install `matR`, the MG-RAST interface add-on package. For this, use:

```
> install.packages("matR", repo =
+         "http://dunkirk.mcs.anl.gov/~braithwaite/matR",
+         type = "source")
```

- Open an R session. Use the following command to load the matR package:

```
> library(matR)
```

  You would use a similar command to load any other package.

- matR relies on various other packages. To install these, follow the instructions provided by running this function:

```
> dependencies()
```

  At the time of this writing, the packages relied on by matR are: `RJSONIO`, `ecodist`, `gplots`, `scatterplot3d`. If the `dependencies` function doesn't complete successfully, these need to be installed one at a time, as follows:

```
> install.packages("RJSONIO")
> install.packages("ecodist")
> # ...etc
```

- Now your R environment is ready to go!

1.2 Easy Lessons in R

- Here we will learn some basics of working with data in R.
- For us, two kinds of data objects are essential in R: `matrix` and `data.frame`. First, we create a `matrix`. The function `sample` just creates a random permutation, as shown.

```
> sample(1:200)

  [1] 137  71 167 107 109  36  65 104  57 102  35  39  17
 [14] 120 142  68 130 162  54  44  34 182  20   4 170  74
 [27] 116 175  32  99  80  78  91  21  19 135 128   7 143
 [40]  64 185   3  82  37 139  77   6 108 136  93 189  10
 [53]  62 195  38 145  33 103  96  25 113  61  92 126 114
 [66] 111  28  58  86 151 148 187 178   1 140 174 192 156
 [79] 153  31  98  66 121  97 119 115  15 163 184  79  16
 [92] 166 159  30 101 133  59 155   5  60 110 149  50  11
[105] 160 164 180  49  69 134 172  48 112  51  26 125 176
[118]  43 147  83  70  75  72  12  94  47 123  81 194 144
[131]  29  41 173 117  55 138  40  84  52  22 100 199  18
[144] 193 161  88 132  56 118 127 169 198  73 154 181 152
[157] 183 157  95  63  46 158 124 171 146   2  53 105 196
[170] 177 197  89  90  14  45 186 122 179  23 129  27 191
[183]  76 168 200 165   8  87 106 190  24  67  42  85  13
[196] 131 150   9 188 141

> m <- matrix(sample (1:200), nrow=20, ncol=10)
> m

      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]  117  148  194   30  168  143  113  107  145   176
 [2,]  198   76  112  192  111  195  190   24   41   125
 [3,]   31   62    5  146   84  139  151  189   92    57
 [4,]   20   13  128   94    8   28  100  142  147   110
 [5,]    3    4   80   89   56  193  141  127   60    85
 [6,]  183   35   45   19  173   98   93  197    7   153
 [7,]  134   86  179   64   75   68   77  103   58   163
 [8,]  154  159  115  196   73  105   65  161   33    18
 [9,]  121  109  104   88   53  165  132  108   91     9
[10,]   40  119   37   32   29   97  188   49   50   164
[11,]   27  137  169  182   26  133   74  191  144   129
[12,]   61  186  200   69  199  106   72  172   96    95
[13,]    1  116  152  150  170  122   11  167   23    90
[14,]  187   67  123    2   44   46   71  130   43   140
[15,]  135  185   42  136   82   78  131   79   34   177
[16,]   51  184  166   81   99   39  155  126   55    52
[17,]  124  138  178   87   10  120  181   63   12   114
[18,]   66  156    6   48   16   83   36  102   38    22
[19,]   14   15   21  149  157   17  101  160   54    47
[20,]  158   25  174   70  175   59  180  162  171   118
```

- The `apply` function, below, applies the function specified by its last argument (in this case, `mean`) along the dimension of `m` specified by the second argument. So here we calculate the row means and then the column means of `m`.

```
> apply(m,1,mean)

 [1] 134.1 126.4  95.6  79.0  83.8 100.3 100.7 107.9  98.0
[10]  80.5 121.2 125.6 100.2  85.3 107.9 100.8 102.7  57.3
[19]  73.5 129.2

> apply(m,2,mean)

 [1]  91.25 101.00 111.50  96.20  90.40 101.70 113.10 127.95
 [9]  69.70 102.20
```

- Generally speaking, a `data.frame` is different from a `matrix` because it may contain non-numeric data. So, now we create a `data.frame` consisting of the *column means* and *column standard deviations* of `m`, but also containing a third, descriptive column.

```
> df <- data.frame(mu=apply(m,2,mean), sigma=apply(m,2,sd))
> df$sample <- paste("sample", LETTERS[1:10], sep = "-")
> df

       mu    sigma   sample
1   91.25 66.59846 sample-A
2  101.00 61.23896 sample-B
3  111.50 65.84311 sample-C
4   96.20 58.40386 sample-D
5   90.40 62.89457 sample-E
6  101.70 51.01403 sample-F
7  113.10 51.78691 sample-G
8  127.95 48.77820 sample-H
9   69.70 48.53442 sample-I
10 102.20 53.59163 sample-J
```

- Suppose we wanted to reorder the columns. Flexible indexing of objects is a great strength of R. Here we *replace* the first and third columns of `df` with (respectively) its own third and first columns — effectively, reordering them.

```
> df [c(1,3)] <- df [c(3,1)]
> df

         mu    sigma sample
1  sample-A 66.59846  91.25
2  sample-B 61.23896 101.00
3  sample-C 65.84311 111.50
4  sample-D 58.40386  96.20
5  sample-E 62.89457  90.40
6  sample-F 51.01403 101.70
7  sample-G 51.78691 113.10
8  sample-H 48.77820 127.95
9  sample-I 48.53442  69.70
10 sample-J 53.59163 102.20
```

- That almost worked, but notice that while the data moved, the column *labels* did not. It is possible to refer directly to the row and column labels of a `matrix` or `data.frame`, as follows.

```
> rownames(df)
```

```
 [1] "1"  "2"  "3"  "4"  "5"  "6"  "7"  "8"  "9"  "10"
```

```
> colnames(df)
```

```
[1] "mu"     "sigma"  "sample"
```

Now we finish by correcting the column labels.

```
> colnames(df) [c(1,3)] <- colnames(df) [c(3,1)]
> df
```

```
     sample    sigma     mu
1  sample-A 66.59846  91.25
2  sample-B 61.23896 101.00
3  sample-C 65.84311 111.50
4  sample-D 58.40386  96.20
5  sample-E 62.89457  90.40
6  sample-F 51.01403 101.70
7  sample-G 51.78691 113.10
8  sample-H 48.77820 127.95
9  sample-I 48.53442  69.70
10 sample-J 53.59163 102.20
```

- Here are some commands for viewing the first elements, last elements, and overall structure of large objects.

```
> head(m)
```

```
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]  117  148  194   30  168  143  113  107  145   176
[2,]  198   76  112  192  111  195  190   24   41   125
[3,]   31   62    5  146   84  139  151  189   92    57
[4,]   20   13  128   94    8   28  100  142  147   110
[5,]    3    4   80   89   56  193  141  127   60    85
[6,]  183   35   45   19  173   98   93  197    7   153
```

```
> tail(m)
```

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[15,]  135  185   42  136   82   78  131   79   34   177
[16,]   51  184  166   81   99   39  155  126   55    52
[17,]  124  138  178   87   10  120  181   63   12   114
[18,]   66  156    6   48   16   83   36  102   38    22
[19,]   14   15   21  149  157   17  101  160   54    47
[20,]  158   25  174   70  175   59  180  162  171   118
```

```
> str(m)
```

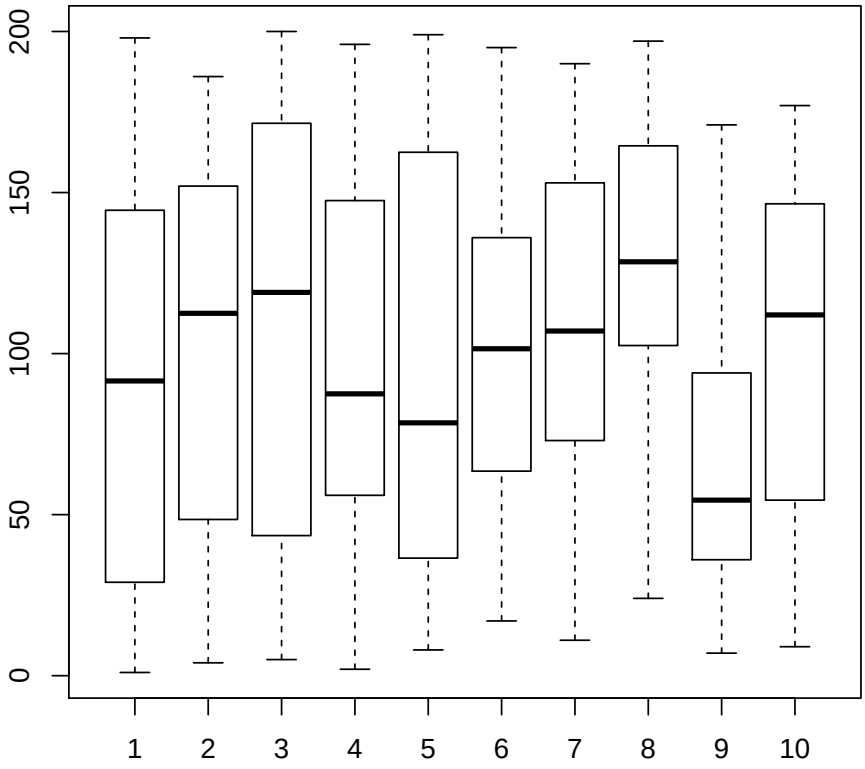
```
 int [1:20, 1:10] 117 198 31 20 3 183 134 154 121 40 ...
```

```
> str(df)
```

```
'data.frame':        10 obs. of  3 variables:
 $ sample: chr  "sample-A" "sample-B" "sample-C" "sample-D" ...
 $ sigma : num  66.6 61.2 65.8 58.4 62.9 ...
 $ mu    : num  91.2 101 111.5 96.2 90.4 ...
```

- Finally, any introduction to R should show how it easily renders statistical graphics, as with this boxplot of the columns of `m`.

```
> boxplot(m)
```



- There is a lot more to R, but the subset of commands shown here, together with the help tutorial (which is next), already enable many things!

1.3 Using R Help

- In R, as with any system, it's important to know how to use the help.
- First, locate the one-page quick reference for all matR commands:

  ```
  > vignette("matR-quick-reference")
  ```

  If that doesn't work, the quick reference is also available at:
  `http://dunkirk.mcs.anl.gov/~braithwaite/R/*******`.
  It may be handy to print a copy.

- Help on any R command is available with:

  ```
  > ?command
  ```

  For example, try:

  ```
  > ?mean
  > ?sample
  > ?apply
  ```

- For keyword-based help, use the double question mark, as in these examples:

  ```
  > ??foo
  > ??bar
  > ??baz
  ```

- Finally, to retrieve an index of all help topics *for a specific package*, use this command, replacing `matR` with the name of the relevant package:

  ```
  > library(help="matR")
  ```

- matR is updated regularly. For a summary of the latest changes, see:

  ```
  > vignette("matR-change-log")
  ```

  The same document is also available at:
  `http://dunkirk.mcs.anl.gov/~braithwaite/R/*******`.

1.4 Five More Lessons in R

-

1.5 Exporting and Importing Data; Saving Images

- This tutorial explains how to get images out of R for publications, how to bring data into R from formats such as csv, tsv, or biom; and how to save data for use in future R sessions, in Excel, or with other programs.

- `write.table()` and `read.table()` are the workhorse commands for exporting and importing any kind of tabular data. They have many options, as well as variants such as `read.csv()`. The following examples show the most common options. These functions are very flexible, though, so consult the help system to learn more.

```
> cc <- collection("4441679.3 4441680.3 4441682.3")
> write.table(cc$raw, file="data.txt", sep="\t")
> x <- read.table(file="data.txt")
> x
```

- Additionally, matR provides a function, `asFile()`, that conveniently exports several kinds of object in a default format. It's not flexible but may be adequate for many purposes.

- The functions `save()` and `load()` store R objects in a binary format for use in later R sessions. (By convention, these files end with `.Rda`.) This is helpful, for example, to store a metagenome collection or the result of an analysis that is computation-intensive. Here are some examples:

```
> cc <- collection("4441679.3 4441680.3 4441682.3")
> p <- pco(cc)
> ls()
> save(cc, p, file="saved_data.Rda")
> rm(cc, p)
> ls()
> load(file="saved_data.Rda")
> ls()
```

- There is an easy method to export images from an R session. First develop the exact commands to produce the desired image interactively. For instance, suppose we want to export the following PCoA.

```
> pco(Waters, main="functional level 3",
+                 col=c(rep("red",12),rep("blue",12)))
```

  To produce a pdf file, simply amend the code in this way.

```
> pdf(filename="my_pco.pdf", width=5, height=5)
> pco(Waters, main="functional level 3",
+                 col=c(rep("red",12),rep("blue",12)))
> dev.off()
```

  The function `pdf()` can be replaced with others, such as `png()`. For more detail, consult the help system.

2 Examples

2.1 Simple Functional Comparison of Lean and Obese Mouse

- This tutorial shows a basic analysis, just for demonstration, so the syntax does not need to be completely understood. The point is to motivate the remaining tutorials.
- First, open an R session and load matR.

```
> library(matR)
```

- We name and identify two metagenomes of interest, create a list of correctly formatted *views* to specify the exact data we want, and retrieve a metagenome collection.

```
> mice <- c(lean="4440463.3", obese="4440464.3")
> v <- default.views$raw
> views <- list()
> length(views) <- 8
> names(views) <- c("L1", "L2", "L3", "L4", "L1n", "L2n", "L3n", "L4n")
> views [1:8] <- v
> views[["L1"]]["level"] <- "level1"
> views[["L2"]]["level"] <- "level2"
> views[["L3"]]["level"] <- "level3"
> views[["L4"]]["level"] <- "function"
> views[["L1n"]]["entry"] <- "ns.normed.counts"
> views[["L2n"]]["entry"] <- "ns.normed.counts"
> views[["L3n"]]["entry"] <- "ns.normed.counts"
> views[["L4n"]]["entry"] <- "ns.normed.counts"
> views[["L1n"]]["level"] <- "level1"
> views[["L2n"]]["level"] <- "level2"
> views[["L3n"]]["level"] <- "level3"
> views[["L4n"]]["level"] <- "function"
> mice <- collection(mice, views)
```

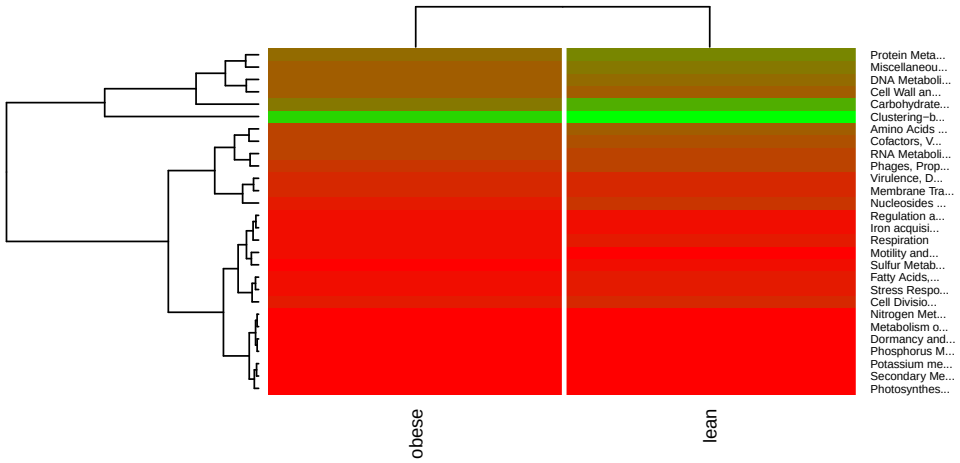
- Next, we inspect the data cursorily.

```
> dim(mice$L1)

[1] 28  2

> dim(mice$L4)

[1] 3171    2

> head(rownames(mice, "L1"))

[1] "Amino Acids and Derivatives"
[2] "Carbohydrates"
[3] "Cell Division and Cell Cycle"
[4] "Cell Wall and Capsule"
[5] "Clustering-based subsystems"
[6] "Cofactors, Vitamins, Prosthetic Groups, Pigments"

> head(rownames(mice, "L4", sep = "; "))
```

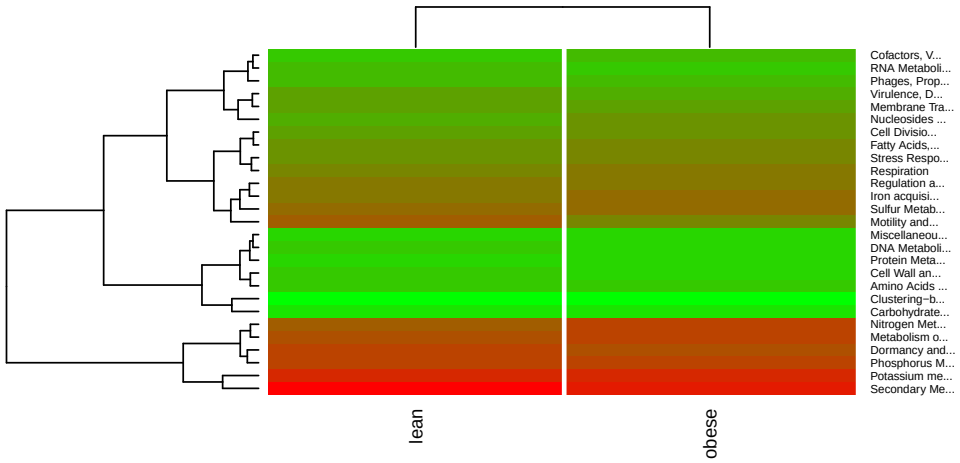
```
[1] "RNA Metabolism; RNA processing and modification; 16S_rRNA_modification_within_P_site_of_ribosc
[2] "RNA Metabolism; RNA processing and modification; 16S_rRNA_modification_within_P_site_of_ribosc
[3] "RNA Metabolism; RNA processing and modification; 16S_rRNA_modification_within_P_site_of_ribosc
[4] "RNA Metabolism; RNA processing and modification; 16S_rRNA_modification_within_P_site_of_ribosc
[5] "RNA Metabolism; RNA processing and modification; 16S_rRNA_modification_within_P_site_of_ribosc
[6] "DNA Metabolism; DNA repair; 2-phosphoglycolate_salvage; Phosphoglycolate phosphatase (EC 3.1.3
```

- Heatmaps of raw and normalized counts at functional level 1 are different, but neither contrasts the samples well, as shown on the next page.

```
> heatmap(mice, "L1", main="", cexCol=0.8, cexRow=0.5,
+   labRow=abbrev(rownames(mice$L1), 15))
```

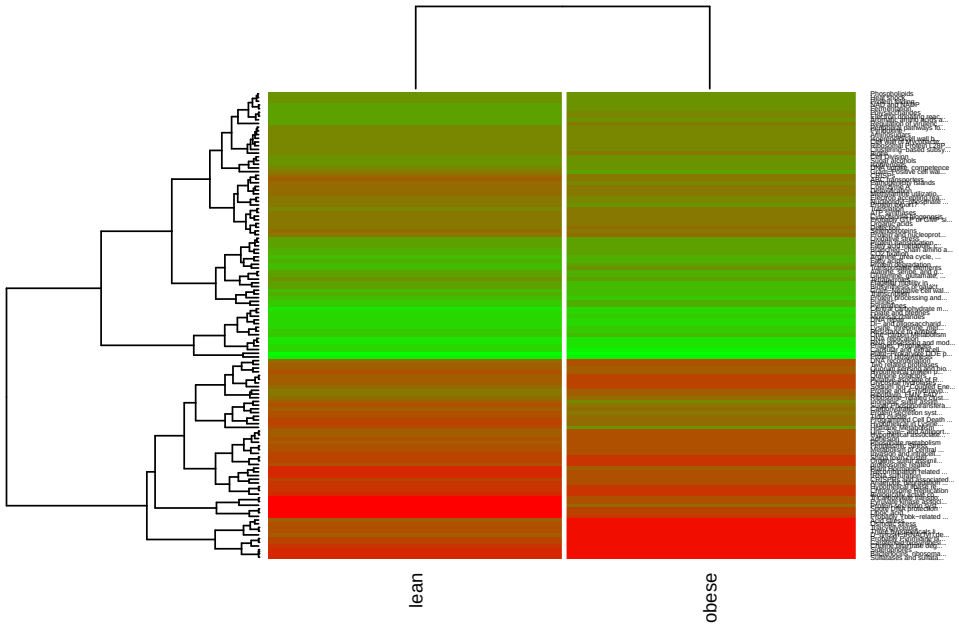



```
> heatmap(mice, "L1n", main="", cexCol=0.8, cexRow=0.5,
+   labRow=abbrev(rownames(mice$L1n), 15))
```



- At functional level 2, the heatmap (of normalized counts) shows slightly more contrast, but still not much.

```
> heatmap(mice, view="L2n", main="", cexCol=0.8, cexRow=0.3,
+   labRow=abbrev(rownames (mice$L2n), 25))
```

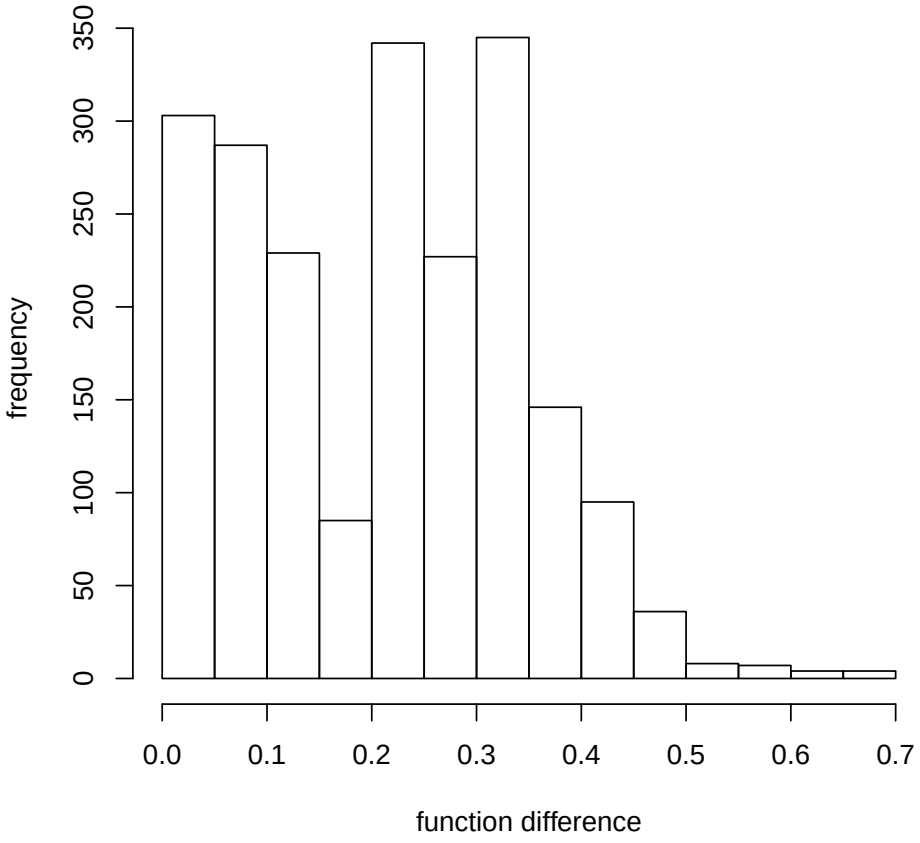


- At functional level 4, there is too much detail for a heatmap of all functions to be clear, but we can inspect the data numerically. We compute the difference of normalized counts between the two samples, for each function.

```
> differ <- abs(mice$L4n[,1] - mice$L4n[,2])
> L4n.names <- rownames(mice, "L4", sep = "; ") [
+   rownames(mice$L4) %in% rownames(mice$L4n)]
> names(differ) <- L4n.names
> hist(differ, plot=FALSE) [c("breaks","counts")]

$breaks
 [1] 0.00 0.05 0.10 0.15 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55
[13] 0.60 0.65 0.70

$counts
 [1] 303 287 229  85 342 227 345 146  95  36   8   7   4   4
```

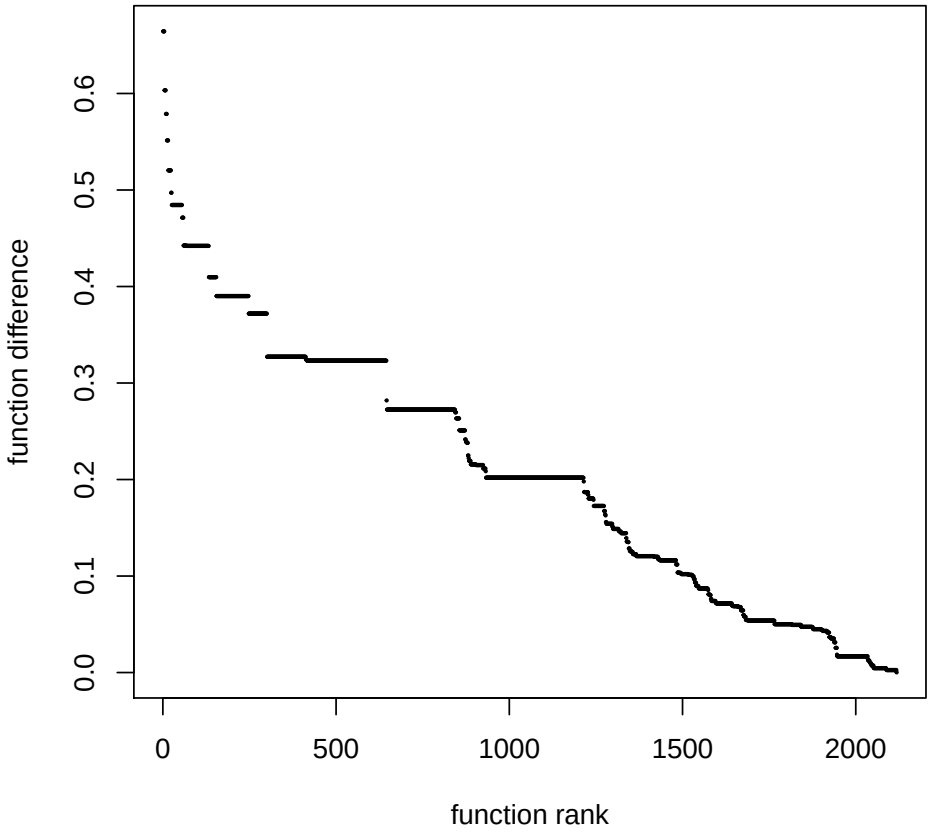
- A histogram of differences is of interest.

```
> hist(differ, plot=TRUE, main="",
+   xlab="function difference", ylab="frequency", )
```



- A plot of the differences, sorted, is also informative.

```
> sort.differ <- differ[order(differ, decreasing=TRUE)]
> plot(sort.differ, pch=19, cex=0.2,
+   xlab="function rank", ylab="function difference")
```

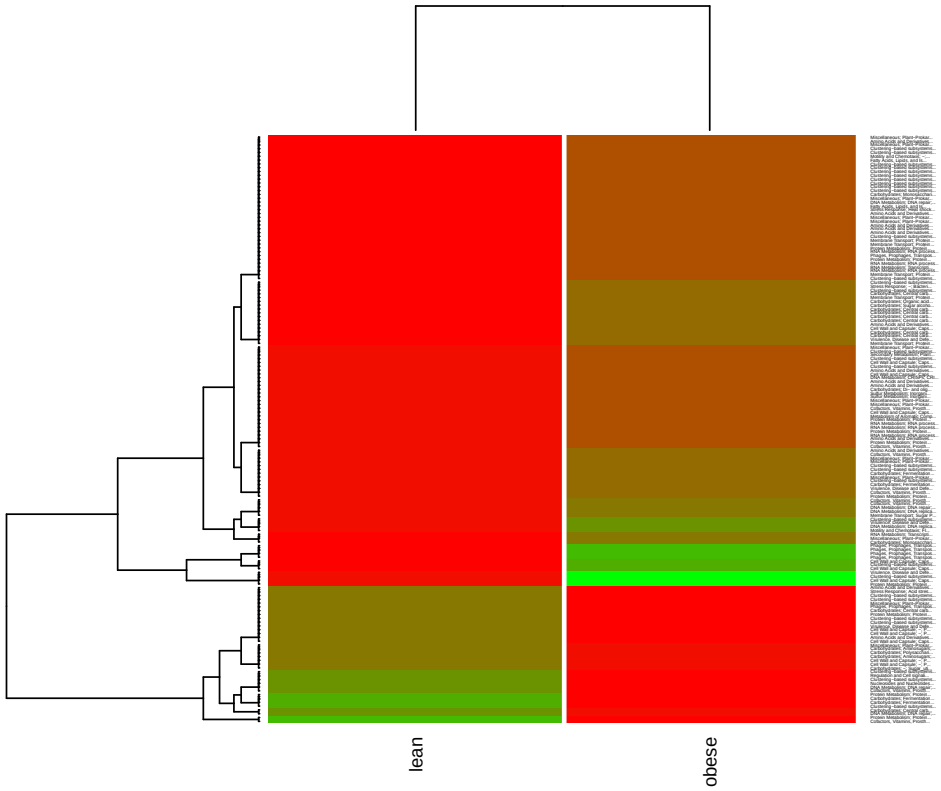


- Based that plot, we might heuristically choose a cutoff to look at the most significant functions.

```
> sum(differ>0.4)

[1] 154

> head(names(sort.differ))

[1] "Clustering-based subsystems; -; CBSS-296591.1.peg.2330; UDP-N-acetylglucosamine 4,6-dehydratas
[2] "Virulence, Disease and Defense; -; C_jejuni_colonization_of_chick_caeca; UDP-N-acetylglucosami
[3] "Cell Wall and Capsule; Capsular and extracellular polysacchrides; Legionaminic_Acid_Biosynthes
[4] "Protein Metabolism; Protein processing and modification; N-linked_Glycosylation_in_Bacteria; U
[5] "Phages, Prophages, Transposable elements, Plasmids; Phages, Prophages; Phage_entry_and_exit; P
[6] "Phages, Prophages, Transposable elements, Plasmids; Phages, Prophages; Phage_packaging_machine
```

- Now we can sharpen the level 4 heatmap by restricting to significant annotations.

```
> which.rows <- rownames(mice, "L4", sep = "; ") %in% (L4n.names[differ>0.4])
> heatmap(mice, view="L4", rows=which.rows, main="",
+   labRow=abbrev(rownames(mice, "L4", sep = "; ") [which.rows], 30),
+   cexCol=0.8, cexRow=0.2)
```



Of course, it is easy to save a record of the annotations included in this heatmap, along with their abundances.

```
> m <- mice$L4
> rownames(m) <- rownames(mice, "L4", sep = "; ")
> asFile(m[which.rows,], file="topfunctions.tsv")

[1] "topfunctions.tsv"
```

2.2 Grouping of Brazilian Coastal Water Samples

•

2.3 A Longer Analysis Example

-

3 Basics

3.1 The Annotation Matrix

- Usually, we study a matrix in which columns are labeled by samples and rows are labeled by annotation. The annotations may be taxonomic or functional. Also, they may be at different hierarchy levels.
- The matrix entries may be simply the raw numbers of observations of each annotation per sample — but they may represent other quantities, too, or be qualified in various ways. For instance, we might be interested in the average evalue of instances of each annotation per sample. Or, we might want to limit the annotation counts by source.
- The process of building a metagenome collection, described in the next section, relies heavily on the matR object `view.parameters`, which makes explicit all the ways an annotation matrix may vary. It is essentially a set of possible key-value pairs. Choosing a compatible set of values defines a *view* of the collection data.

```
> view.params

$entry
[1] "counts"           "normed.counts"    "ns.counts"
[4] "ns.normed.counts" "evalue"           "length"
[7] "percentid"

$annot
[1] "function" "organism"

$level
$level$organism
[1] "domain"  "phylum"  "class"   "order"   "family"  "genus"
[7] "species" "strain"

$level$`function`
[1] "level1"   "level2"   "level3"   "function"


$source
$source$rna
[1] "M5RNA"      "RDP"        "Greengenes" "LSU"
[5] "SSU"

$source$ontology
[1] "NOG"        "COG"        "KO"         "Subsystems"

$source$protein
 [1] "M5NR"      "SwissProt" "GenBank"   "IMG"       "SEED"
 [6] "TrEMBL"    "RefSeq"    "PATRIC"    "eggNOG"    "KEGG"


$hit
$hit$organism
[1] "all"    "single" "lca"
```

```
$hit$`function`
[1] "na"
```

- The object `view.descriptions` contains information about the meaning of each element of `view.pararmeters`. `view.defaults` shows what views are included in a collection by default.
- The parameter values `entry="ns.counts"` and `entry="normed.counts"` describe a matrix with the functions `remove.singletons()` and `normalize()`, respectively, applied to the values of a matrix with `entry="count"`. `entry="ns.normed.counts"` means both functions applied.

3.2 Metagenome Collections

- Metagenome collections are constructed using metagenome IDs. For example:

```
> IDs <- c (gut.1 = "4441695.3", gut.2 = "4441696.3")
> cc <- collection (IDs)
> dd <- collection ("4441679.3 4441680.3 4441682.3 4441695.3 4441696.3 4440463.3 4440464.3")
> ee <- collection (file = "test-IDs.txt")
```

  IDs in files should be whitespace-separated. They can also be named... ...by project

- Specifying metagenome IDs is only half the story. Specific data can be requested, pertaining to the identified metagenomes. The required syntax is slightly complicated, but enables careful analyses.

```
> collection (guts,
+   raw = c(entry = "count"),
+   nrm = c(entry = "normed.counts"))
> collection (guts,
+   L1 = c(level = "level1"),
+   L2 = c(level = "level2"),
+   L3 = c(level = "level3"),
+   L4 = c(level = "function"))
> collection (guts,
+   nog = c(source = "NOG"),
+   cog = c(source = "COG"),
+   ko = c(source = "KO"))
> collection (guts,
+   lca = c(annot = "organism", hit = "lca"),
+   repr = c(annot = "organism", hit = "single"),
+   all = c(annot = "organism", hit = "all"))
```

  In each case, a separate annotation matrix exists in the collection for each view on the data that was specified. These matrices are called "views". For complete information about all the options for views, inspect the contents of these objects.

```
> view.parameters
> view.descriptions
> view.defaults
```

- A handy approach is to create lists of data views for easy reuse, as follows.

```
> top.levels <- list (
+   L1 = c(level = "level1"),
+   L2 = c(level = "level2"))
> all.ontologies <- list (
+   nog = c(source = "NOG"),
+   cog = c(source = "COG"),
+   ko = c(source = "KO"),
+   sub = c(source = "Subsystems"))
> all.count.methods <- list (
+   lca = c(annot = "organism", hit = "lca"),
+   repr = c(annot = "organism", hit = "single"),
+   all = c(annot = "organism", hit = "all"))
> collection (guts, top.levels)
> collection (guts, all.ontologies)
> collection (guts, all.count.methods)
```

- Various functions apply to collections.

```
> samples(cc)      # show metagenomes in the collection
> projects(cc)     # show projects in the collection
> names(cc)        # show names of metagenomes
> views(cc)        # show the data views in the collection
> viewnames(cc)    # show just the names of the views
> metadata(cc)     # access metadata
```

- Views in collections are accessed with "$" by the name the view was given.

```
> cc$count
> cc$normed
```

  Generally, the purpose of views is to show different aspects of the same selection of metagenomes. Views can be specified when a collection is constructed as we saw, and can also be added to an existing collection:

```
> dd$cog <- c (source = "COG")
```

- Names of annotations within a view are accessed with `rownames()`.

```
> rownames(Guts, view = "raw", sep = FALSE)
> rownames(Guts, view = "raw", sep = TRUE)
> rownames(Guts, view = "raw", sep = "\t")
```

  The `sep` parameter ... hierarchy

- Subsets can be taken of collections, as of other objects. For instance, here we extract the first three metgenomes of `dd` into a new collection.

```
> ff <- dd [1:3]
```

3.3 Using Metadata

- Usually there is metadata associated with a metagenome collection. Metadata fields are named in a way that reflects their hierarchical organization. This tutorial shows how to access metadata, by several examples.

  The following command lists all metadata of the `Guts` example collection.

  ```
  > metadata(Guts)
  ```

- Usually there is some reason to pick out specific metadata elements. For that purpose, metadata can be indexed. To select elements, an arbitrary number of index vectors may be specified. For instance, we can use one index (of length one) to get all metadata from one metagenome:

  ```
  > metadata(Guts) ["4440464.3"]
  ```

- Here is another example of metadata indexing: two indices (each of length one) to get certain elements from all metagenomes.

  ```
  > metadata(Guts) ["latitude", "longitude"]

  [[1]]
  mgm4441679.3.metad... 40.5109
  mgm4441680.3.metad... 40.5109
  mgm4441682.3.metad... 40.5109
  mgm4441695.3.metad... 33.537594
  mgm4441696.3.metad... 33.537594
  mgm4440463.3.metad... 38.6480
  mgm4440464.3.metad... 38.6480

  [[2]]
  mgm4441679.3.metad... -88.9916
  mgm4441680.3.metad... -88.9916
  mgm4441682.3.metad... -88.9916
  mgm4441695.3.metad... -116.097751
  mgm4441696.3.metad... -116.097751
  mgm4440463.3.metad... -90.3045
  mgm4440464.3.metad... 90.3045
  ```

  An alternative form returns the same output in a more convenient form.

  ```
  > metadata(Guts) ["latitude", "longitude", bygroup=TRUE]

                       V1          V2
  mgm4440463.3   38.6480    -90.3045
  mgm4440464.3   38.6480     90.3045
  mgm4441679.3   40.5109    -88.9916
  mgm4441680.3   40.5109    -88.9916
  mgm4441682.3   40.5109    -88.9916
  mgm4441695.3 33.537594 -116.097751
  mgm4441696.3 33.537594 -116.097751
  ```

  In this variant NA is placed when a field is missing, as in the next example.

  ```
  > metadata(Guts) ["host_common_name", "disease", ".age", bygroup=TRUE]
  ```

```
                        V1      V2                   V3
mgm4440463.3        Mouse    <NA> 8 or 14 weeks of age
mgm4440464.3        Mouse    <NA> 8 or 14 weeks of age
mgm4441679.3          cow    <NA>                5 yrs
mgm4441680.3          cow    <NA>                5 yrs
mgm4441682.3          cow    <NA>                5 yrs
mgm4441695.3 striped bass healthy                <NA>
mgm4441696.3 striped bass    sick                <NA>
```

- Here we obtain the entire environmental package from one metagenome, using *one* index of *length* two: only metadata fields matching both strings are selected.

```
> metadata(Guts) [c ("4440464.3", "env_package.data")]
```

- Finally, this example uses three indices (all of length two) to select various elements:

```
> metadata(Guts) [c ("env","temp"), c ("0464", "PI_organization"),
+   c ("0464","biome")]

[[1]]
mgm4441679.3.metad... -80
mgm4441680.3.metad... -80
mgm4441682.3.metad... -80

[[2]]
mgm4440464.3.metad... Washington University in St. Louis
mgm4440464.3.metad... http://wustl.edu/
mgm4440464.3.metad... St. Louis, MO
mgm4440464.3.metad...  USA

[[3]]
mgm4440464.3.metad... animal-associated habitat
```

- Metadata can be handled independently of annotation data. This can save time in situations where annotation data is not needed. A metadata object can be built in exactly the same way as a collection, as shown.

```
> mm <- metadata("4441679.3 4441680.3 4441682.3 4441695.3 4441696.3")
```

Now `mm` can be used just as `metadata(Guts)` was used, above.

4 Analysis

4.1 Analysis Functions in Detail

- At present matR provides these customized analysis functions: `boxplot()`, `pco()`, `heatmap()`, `parcoord()`, and `sigtest()`.
- Each reimplements basic functions with added features and helpful default settings. In most cases, options to the basic functions also apply to the matR versions. More experienced users can use the basic functions directly as needed. They are: `base::boxplot()`, `ecodist::pco()`, `graphics::points()`, `graphics::text()`, `gplots2::heatmap.2()`, `stat::parcoord()`, plus several individual statistical test functions from the `stat` package.

  options to the matR function options to base function
- Boxplot Annotations Boxplots can be useful to gain a general idea of the distribution of annotation counts in each sample of a collection.
- Heatmap Dendrograms

4.2 Other Useful R Packages

-

5 Miscellaneous

5.1 Calling the MG-RAST API Directly

- The full functionality of the MG-RAST API is available through matR. For details on the API, visit its homepage, `http://api.metagenomics.anl.gov`.
- Many API resources are available through the mid-level interface function, `mGet()`. The following examples establish the general pattern.

  ```
  > mGet("sequenceSet/blah...")
  ```

- When you need more control, use the low-level function `callRaw()`, which only prepends the API server name and appends the session authorization key (if set) to its argument.

  ```
  > callRaw("matrix/....blah")
  ```

- Most API resources are returned as JSON objects and automatically parsed by `mGet` (or `callRaw`). JSON parsing can be forestalled.

5.2 Using R within an iPython Notebook

-