

# A package to work with *E. coli* genome (and other circular DNA entities)

Laurent

May 12, 2008

## Introduction

This package is a mixture of data (close the the data in the meta-data packages) and of code. The code was not included in other packages because:

- much is still to be done, much is likely to change.
- some features do are appear completely compatible with what is existing elsewhere

Things will settle in time...

The package contains data on *Escherichia coli* but can be used with other bacterial genomes or plamids, provided one supplies the required data.

To load the package, do:

```
R> library(ecolitk)
```

## *E. coli* data

Data related to *E. coli* are included in the package. Effort was made to follow the convention used in the meta-data packages available on bioconductor. On should refer to the help files for further details.

In the following example, we explain how to find the positions of the genes of the operon lactose on the genome. A first step is to load needed data structures:

```
> data(ecoligenomeSYMBOL2AFFY)
> data(ecoligenomeCHRLoc)
```

This done, we want to find the genes with names like `lac*`. The *base* function `grep` provides a convenient way to do it. Because the data structures are centered on Affymetrix probeset identifiers, an extra step is need to convert the gene name into the corresponding identifier. The locations of the genes can then be extracted from the environment `ecoligenomeCHRLoc`. We store them in the `beg.end`.

```

> lac.i <- grep("^lac", ls(ecoligenomeSYMBOL2AFFY))
> lac.symbol <- ls(ecoligenomeSYMBOL2AFFY)[lac.i]
> lac.affy <- unlist(lapply(lac.symbol, get, envir = ecoligenomeSYMBOL2AFFY))
> beg.end <- lapply(lac.affy, get, envir = ecoligenomeCHRLOC)
> beg.end <- matrix(unlist(beg.end), nc = 2, byrow = TRUE)

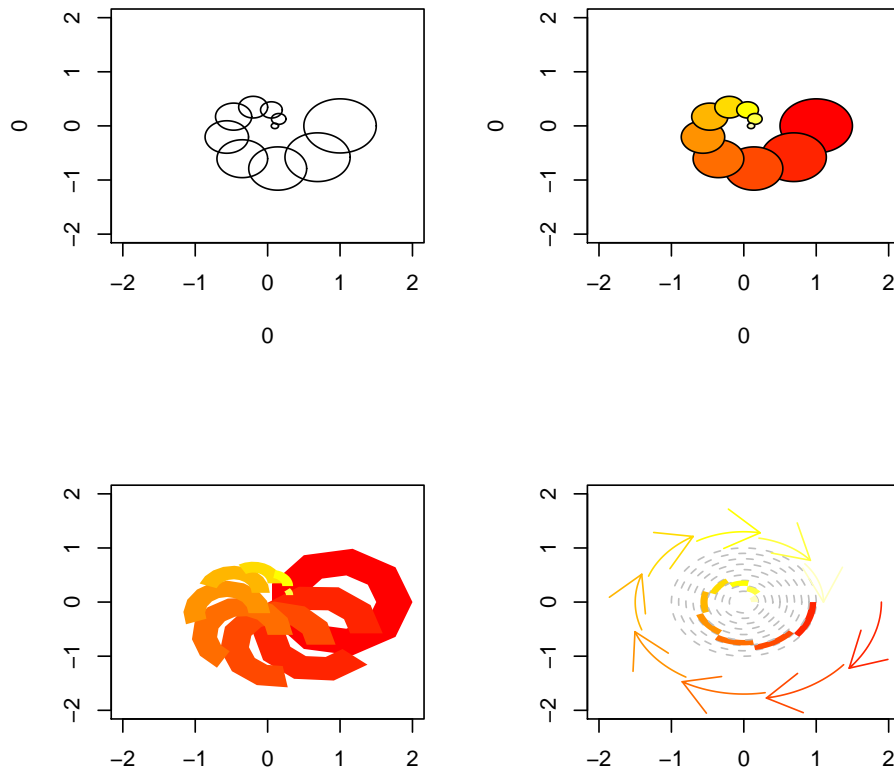
```

## Primitives for circular entities

```

> par(mfrow = c(2, 2))
> n <- 10
> thetas <- rev(seq(0, 2 * pi, length = n))
> rhos <- rev(seq(1, n)/n)
> xy <- polar2xy(rhos, thetas)
> colo <- heat.colors(n)
> plot(0, 0, xlim = c(-2, 2), ylim = c(-2, 2), type = "n")
> for (i in 1:n) linesCircle(rhos[i]/2, xy$x[i], xy$y[i])
> plot(0, 0, xlim = c(-2, 2), ylim = c(-2, 2), type = "n")
> for (i in 1:n) polygonDisk(rhos[i]/2, xy$x[i], xy$y[i], col = colo[i])
> plot(0, 0, xlim = c(-2, 2), ylim = c(-2, 2), type = "n", xlab = "",
+      ylab = "")
> for (i in 1:n) polygonArc(0, thetas[i], rhos[i]/2, rhos[i], center.x = xy$x[i],
+      center.y = xy$y[i], col = colo[i])
> plot(0, 0, xlim = c(-2, 2), ylim = c(-2, 2), type = "n", xlab = "",
+      ylab = "")
> for (i in (1:n)[-1]) {
+   linesCircle(rhos[i - 1], col = "gray", lty = 2)
+   polygonArc(thetas[i - 1], thetas[i], rhos[i - 1], rhos[i],
+     col = colo[i], edges = 20)
+   arrowsArc(thetas[i - 1], thetas[i], rhos[i] + 1, col = colo[i],
+     edges = 20)
+ }

```



## Plotting toolbox for circular genomes

### Plot a (circular) genome

The function `cPlotCircle` draws a circular chromosome.

```
> cPlotCircle(main.inside = "E. coli - K12")
```



The drawing of the chromosome is thought as a first step when plotting informations on a circular genome. The following example shows how to plot the locations of the genes for the operon lactose on the genome. (the steps leading to their respective positions were detailed above).

### **plot genes on a genome**

We sort the genes the `lac*` genes discussed in an example above according to their positions on the genome.

```
> lac.o <- order(beg.end[, 1])  
> lac.i <- lac.i[lac.o]  
> lac.symbol <- lac.symbol[lac.o]  
> lac.affy <- lac.affy[lac.o]  
> beg.end <- beg.end[lac.o, ]
```

This done, we plot them as polygons:

```
> cPlotCircle(main.inside = "E. coli - K12", main = "lac genes")
> polygonChrom(beg.end[, 1], beg.end[, 2], ecol.i.len, 1, 1.4)
```

### **lac genes**



Zoom in can be achieved by playing with `xlim` and `ylim`.

```
> l <- data.frame(x = c(0.47, 0.48), y = c(0.87, 0.9))
> cPlotCircle(xlim = range(l$x), ylim = range(l$y), main = "lac genes")
> polygonChrom(beg.end[, 1], beg.end[, 2], ecol.i.len, 1, 1.007,
+   col = rainbow(4))
> legend(0.47, 0.9, legend = lac.symbol, fill = rainbow(4))
```

## lac genes



## More plots

The following plot uses several different annotation sources.

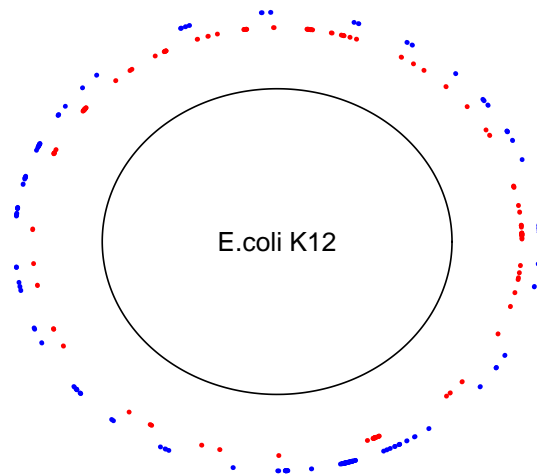
It tries to show a spatial pattern for the genes annotated as - Macromolecules (cellular constituent) biosynthesis -. In the following plot those genes are plotted according to their position on the genome and according to their respective strand.

```
> library(Biobase)
> data(ecoligenomeBNUM2STRAND)
> data(ecoligenomeBNUM)
> data(ecoligenomeBNUM2MULTIFUN)
> data(ecoligenomeCHRLOC)
> affyids <- ls(ecoligenomeCHRLOC)
> affypos <- mget(affyids, ecoligenomeCHRLOC, ifnotfound = NA)
> bnums <- unlist(mget(affyids, ecoligenomeBNUM, ifnotfound = NA))
> strands <- unlist(mget(bnums, ecoligenomeBNUM2STRAND, ifnotfound = NA))
> multifun <- mget(bnums, ecoligenomeBNUM2MULTIFUN, ifnotfound = NA)
```

```

> f <- function(x) {
+   if (all(is.na(x)))
+     return(FALSE)
+   length(grep("^1\\.6", x) > 0)
+ }
> is.selected <- unlist(lapply(multifun, f))
> cPlotCircle(main.inside = "E.coli K12")
> beg.end <- matrix(unlist(affypos), nc = 2, byrow = TRUE)
> good <- strands == ">" & is.selected
> linesChrom(beg.end[good, 1], beg.end[good, 2], ecolilen, 1.4,
+   col = "red", lwd = 3)
> good <- strands == "<" & is.selected
> linesChrom(beg.end[good, 1], beg.end[good, 2], ecolilen, 1.5,
+   col = "blue", lwd = 3)

```



## More complex plot

The next example is more complex: we want to compute and display the GC content over a fragment of the genome. The fragment is decided to be of size 1 million bases, with the origin of replication (base zero) in the middle of the fragment. We limit the size to lower the computing resources needed to build this document and to demonstrate how to perform something not so trivial.

```
cPlotCircle(main.inside = "E. coli - K12")

data(ecoli.m52.genome)
size.frag <- 1000000

fragment.r <- substring(ecoli.m52.genome, 1, size.frag/2)
fragment.l <- substring(ecoli.m52.genome, ecoli.len - size.frag / 2, ecoli.len)
fragment <- paste(fragment.l, fragment.r, sep="")
library(matchprobes)
tmp <- wstringapply(fragment, 400, 200, basecontent)
gccontent <- unlist(lapply(tmp, function(x) sum(x[3:4]) / sum(x)))

theta0 <- chromPos2angle(0 - size.frag/2, ecoli.len)
theta1 <- chromPos2angle(size.frag/2, ecoli.len)

linesCircle(1.5, col="gray", lty = 2)
linesArc(theta0, theta1, gccontent + 1)
```

Zooming can be performed by setting the `xlim` and `ylim` parameters of the function `cPlotCircle`. In our example, there seem to be a CG rich island we would like to have a closer look at:

```
par(mfrow=c(1,2))
cPlotCircle(main.inside = "E. coli - K12")
l <- data.frame(x=c(-0.7737990, -0.5286815), y=c(1.521509, 1.304151))
linesCircle(1.5, col="gray", lty = 2)
linesArc(theta0, theta1, gccontent + 1)
rect(l$x[1], l$y[1], l$x[2], l$y[2], border="red")

cPlotCircle(xlim=range(l$x), ylim=range(l$y))
box(col="red")
linesCircle(1.5, col="gray", lty = 2)
linesArc(theta0, theta1, gccontent + 1)
```

As the parameter `SLIDE` was set to 200, one can estimate from the close up that the size of the island is roughly over 1500 base pairs.



## Genes in operons

In procaryotes, some genes are *bundled* in operons. This means that a some genes are physically located near each others on the genome, and that they are transcribed<sup>1</sup> together. Complex mecanismos can regulate the translation<sup>2</sup>. Microarrays are designed to measure the relative abundance of transcripts, therefore genes of the same operon should have the same expression level.

First one has to find the Affymetrix identifiers for the genes known to be in operons:

```
> library(Biobase)
> data(ecoligenome.operon)
> data(ecoligenomeSYMBOL2AFFY)
> tmp <- mget(ecoligenome.operon$gene.name, ecoligenomeSYMBOL2AFFY,
+           ifnotfound = NA)
> ecoligenome.operon$affyid <- unname(unlist(tmp))
> ecoligenome.operon <- subset(ecoligenome.operon, !is.na(affyid))
```

For convenience, the Affymetrix probe set identifiers are stored in the *data.frame*. As an exercise, the reader can write the few lines of code needed to plot the operon on the genome (see above for plotting examples).

Once this done, grouping the Affymetrix identifiers according to the operon they belong to is done simply:

```
> attach(ecoligenome.operon)
> affyoperons <- split(affyid, operon.name)
> detach(ecoligenome.operon)
> affyoperons[18:22]
```

```
$cai
```

```
[1] "caiT_b0040_st" "caiA_b0039_st" "caiB_b0038_st" "caiC_b0037_st"
[5] "caiD_b0036_st" "caiE_b0035_st"
```

```
$carAB
```

```
[1] "carA_b0032_st" "carB_b0033_st"
```

```
$celABCDF
```

```
[1] "celA_b1738_st" "celB_b1737_st" "celC_b1736_st" "celD_b1735_st"
[5] "celF_b1734_st"
```

```
$cheRBYZ
```

```
[1] "tar_b1886_st" "tap_b1885_st" "cheR_b1884_st" "cheB_b1883_st"
```

---

<sup>1</sup>The *transcription* is the copy of DNA into RNA

<sup>2</sup>The *translation* is the making of a protein from RNA

```
[5] "cheY_b1882_st" "cheZ_b1881_st"
```

```
$chpSB
```

```
[1] "chpS_b4224_st" "chpB_b4225_st"
```

Since we are going to use Affymetrix data, loading the affy package is needed:

```
> library(affy)
```

A bioconductor package of experimental E.coli data is used:

```
> library(ecoliLeucine)
```

```
> data(ecoliLeucine)
```

First, one should normalize the data:

```
> abatch.nqt <- normalize(ecoliLeucine, method = "quantiles")
```

The summary statistics for a probe set can be obtained simply:

```
> names(affyoperons)[18]
```

```
[1] "cai"
```

```
> eset <- computeExprSet(abatch.nqt, pmcorrect.method = "pmonly",  
+   summary.method = "medianpolish", ids = affyoperons[[18]])
```

```
6 ids to be processed
```

```
|           |  
|#####|
```

```
> operons.eset <- computeExprSet(abatch.nqt, pmcorrect.method = "pmonly",  
+   summary.method = "medianpolish", ids = unlist(affyoperons))
```

```
499 ids to be processed
```

```
|           |  
|#####|
```

We assumed that genes within operons should be all differentially expressed, or all not differentially expressed.

```
> library(multtest)
```

```
> my.ttest <- function(x, i, j) {  
+   pval <- t.test(x[i], x[j])$p.value  
+   return(pval)  
+ }
```

```

> is.lrpplus <- pData(operons.eset)$strain == "lrp+"
> is.lrpmoins <- pData(operons.eset)$strain == "lrp-"
> operons.ttest <- esApply(operons.eset, 1, my.ttest, is.lrpplus,
+   is.lrpmoins)
> operons.ttest.adj <- mt.rawp2adjp(operons.ttest, "BY")$adjp
> operons.diff.expr <- operons.ttest.adj < 0.01

```

One needs a little bit of bookkeeping to know what belongs to which operon. We can build a list of indexes to know what belongs to where:

```

> operons.i <- split(seq(along = operons.ttest), ecoligenome.operon$operon.name)

```

The thrill is then to see if the results for differential expression (or non-differential expression) are homogeneous among the genes within the same operon...