R / Bioconductor: Curso Intensivo

Leonardo Collado Torres
Licenciatura en Ciencias Genómicas, UNAM
www.lcg.unam.mx/~lcollado/index.php

Cuernavaca, México
Oct-Nov, 2008

Bioconductor

Motivación

- `Bioconductor` surgió por la gran cantidad de datos experimentales que estaban siendo generados. Son tal cantidad, que requerimos de la estadística y de la bioinformática para analizarlos.
- Por otro lado, muchos laboratorios pierden tiempo re-escribiendo programas que ya otros habían hecho.
- El otro problema es tener acceso a los programas que otros hacen. Por eso, `Bioconductor` es de distribución libre.

De su funcionamiento

- `Bioconductor` empezó en el otoño del 2001.
- Las personas principales que mantienen a `Bioconductor` son del `Fred Hutchinson Cancer Research Center` en Seattle, USA.
- Salen dos versiones nuevas por año.
- En sí, muchos de los paquetes de `Bioconductor` están basados en la programación orientada a objetos y usan la clase S4.
- Acuérdense de que `R` puede ser útil para programar aunque muchos de los que mantienen a `Bioconductor` tienen una formación fuerte en estadística.

Visitemos la página

- Entren a la página de Bioconductor y navegen un poco.
- Notarán que hay:
  - Información sobre cursos.
  - Un FAQ.
  - Una sección donde te puedes registrar a las "mailing lists" por si quieres pedir ayuda y/o ayudar a otros.
  - Información sobre como instalar los paquetes.
  - Tal vez la sección más importante para nosotros es la parte de software. Generalmente cada paquete tiene PDFs donde lo explican y/o ponen ejercicios demo.

CRAN

- Mientras que `Bioconductor` es el repositorio de información y paquetes más relacionado a la genómica, acuérdense de CRAN.
- Allí pueden encontrar paquetes útiles para ustedes, como es el RMySQL que veremos adelante.
- En fin, en `Bioconductor` pueden bajar datos experimentales, software, etc.

limma: intro

- Vamos a empezar con el paquete limma cuya información se encuentra aquí.
- `limma` fue principalmente creado para analizar microarreglos, relaciones lineales y encontrar genes diferencialmente expresados.
- Sus derivados gráficos son `limmaGUI` y `affylmGUI` mientras que de cierta forma su competencia es `marray`.
- En sí, solo vamos a ver una parte del paquete ya que es bastante extenso. Incluso su guía oficial no cubre todo.
- Para ejemplificar este paquete y sus funciones, vamos a hacer una de las prácticas básicas.

Problema

- En el caso básico de `limma` trabajamos con 4 mediciones por gene en un microarreglo. Se usan dos colores: Cy3 y Cy5. Primero se hace una medición: WT Cy3, Experimento Cy5. Luego cambian los colores.
- Tenemos datos de *zebrafish*, ya que lo usan para estudiar el desarrollo temprano en vertebrados. Swirl es una mutante puntal del gene BMP2 que afecta al eje "dorsal/ventral" del cuerpo. Nuestro objetivo es usar los datos de este experimento para encontrar los genes con un nivel de expresión alterado en esta mutante comparado con el WT.
- Las hibridaciones del experimento se hicieron en dos sets con tintes intercambiados, por lo que tenemos cuatro mediciones por gene. En cada microarreglo, compararon el RNA del pez mutante swirl con el del pez normal.

Datos

- Para empezar, bajen los siguientes archivos a un directorio:
  - fish.gal
  - swirl.1.spot
  - swirl.2.spot
  - swirl.3.spot
  - swirl.4.spot
  - SpotTypes.txt
  - SwirlSample.txt
- Abran `R` y cambien su directorio[1] a donde bajaron los archivos. Podríamos hacer todo vía Internet, pero esto nos va a ahorrar líneas de código.

[1] setwd

readTargets

- Ahora, cargen el paquete `limma` que ya debería estar instalado en los servidores[2].
- Usen la función readTargets para leer una tabla con información de nuestro experimento.

```
> library(limma)
> targets <- readTargets("SwirlSample.txt")
> targets
```

```
  SlideNumber     FileName       Cy3
1          81 swirl.1.spot     swirl
2          82 swirl.2.spot wild type
3          93 swirl.3.spot     swirl
4          94 swirl.4.spot wild type
       Cy5      Date
```

readTargets


```
1 wild type 2001/9/20
2     swirl 2001/9/20
3 wild type 2001/11/8
4     swirl 2001/11/8
```

- Claro, siempre podrían haber llenado esto a mano :P Pero bueno, ahora ya tenemos información sobre como fue el experimento.
- En sí, nuestros archivos no son los más basicos, ya que los microarreglos fueron leídos por un escáner Axon para producir una imagen `TIFF` que después fue analizada con el software `SPOT`.

[2] Si no lo tienen en su lap, siempren pueden instalarlo con las funciones que vimos la 1era clase

read.maimages

- Vamos a usar la función read.maimages para leer los archivos generados por SPOT.
- Nos lee la intensidad *foreground* y *background* para los colores verde y rojo.

```
> RG <- read.maimages(targets$FileName,
+      source = "spot")
```

```
Read swirl.1.spot
Read swirl.2.spot
Read swirl.3.spot
Read swirl.4.spot
```

- Usamos nuestro objeto `targets` para obtener los nombres de los archivos. Ahora chequen su objeto `RG`.

```
> RG
```

readGAL

- De una vez nos podemos dar cuenta que los microarreglos de este experimento tienen 8448 puntos.
- Como info aparte, los experimentalisatas usaron 768 puntos de control. Además, la impresora del arreglo funcionó con una cabeza de impresión de arreglo 4x4. Cada cuadrícula consiste de 22x24 puntos que se imprimieron con una punta de impresión.
- En el archivo `GAL` tenemos el nombre del gene asociado con dada punto. Usamos la función readGAL para leer esta info:

```
> RG$genes <- readGAL("fish.gal")
```

getLayout

- Ahora, como se dieron cuenta, tenemos mucha información previa de como hicieron el microarreglo.
- Podríamos pasar la información (las dimensiones) de la impresora del arreglo manualmente. Claro, es posible que nos equivoquemos :P
- La función getLayout nos recupera esta información.

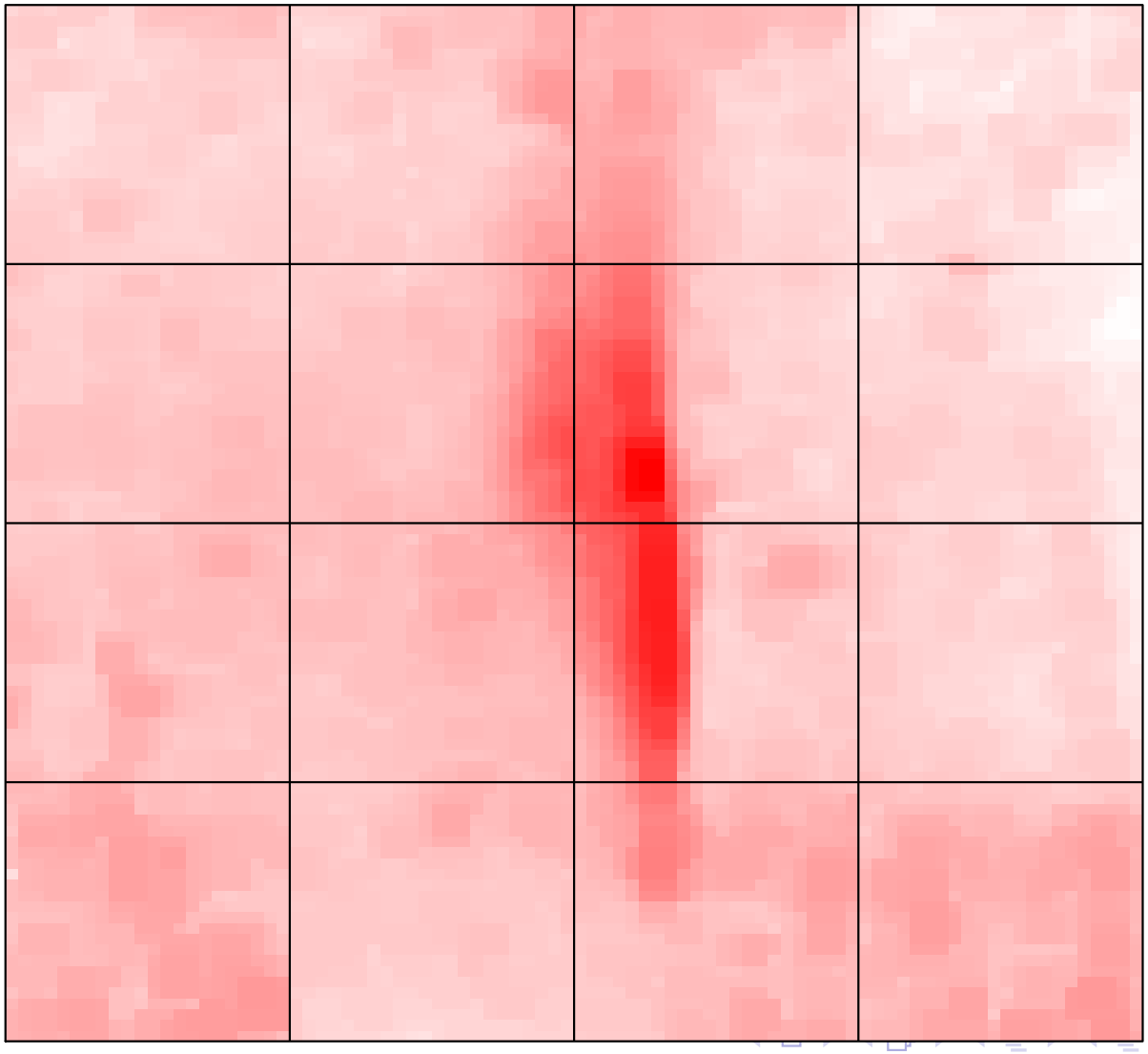
```
> RG$printer <- getLayout(RG$genes)
```

imageplot

- Una función del R básico que nos ayuda a visualizar una matriz es image.
- Para esto de los microarreglos hay una función similar (pero especializada) llamada imageplot.
- Vamos a usar esa función para ver si hay variación en el *background* de nuestros microarreglos.

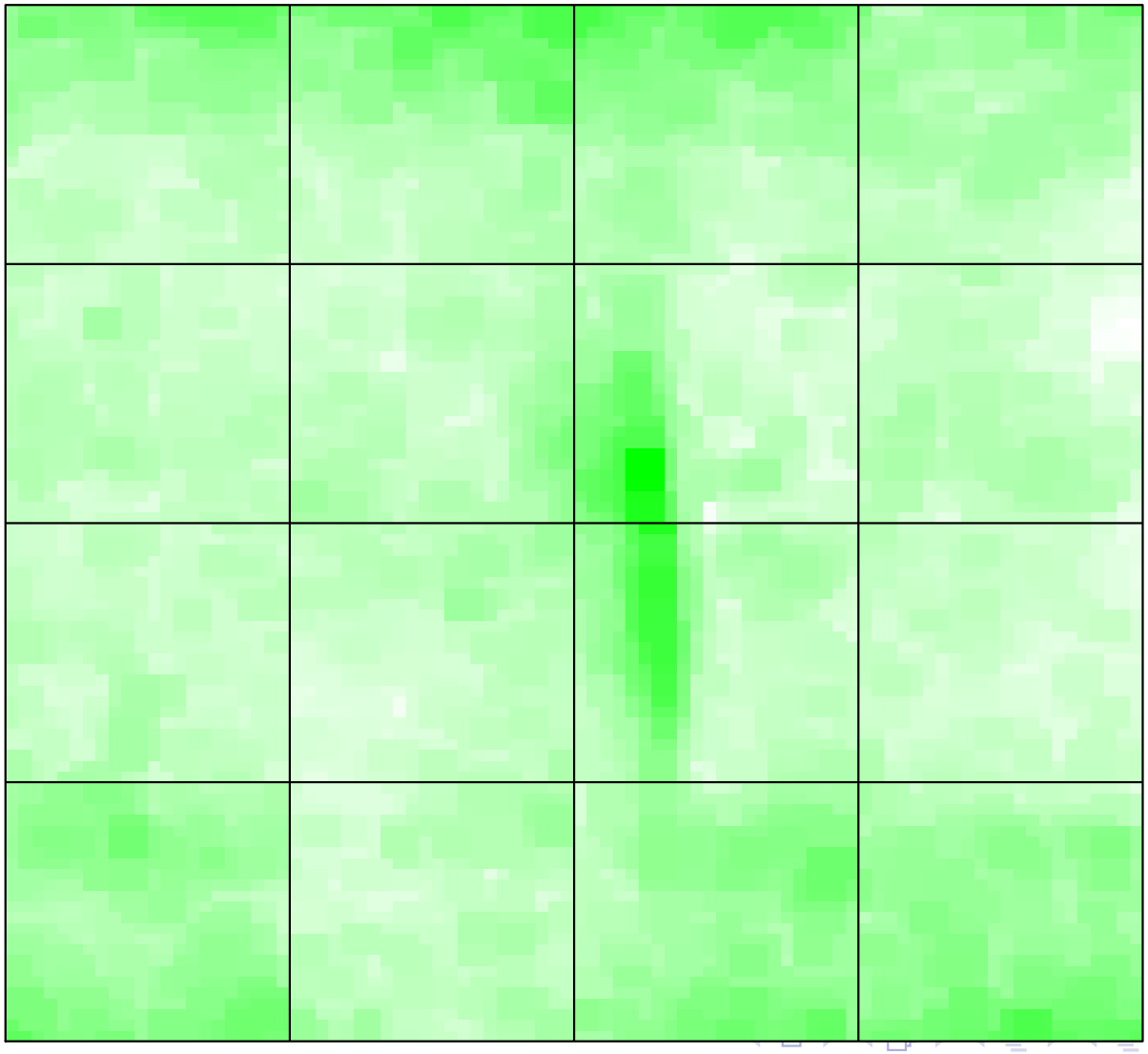
```
> imageplot(log2(RG$Rb[, 1]), RG$printer,
+     low = "white", high = "red")
> imageplot(log2(RG$Gb[, 1]), RG$printer,
+     low = "white", high = "green")
```

imageplot: rojo



z−range 5.9 to 11.1 (saturation 5.9, 11.1)

imageplot: verde



z−range 6.2 to 8.2 (saturation 6.2, 8.2)

normalizeWithinArrays

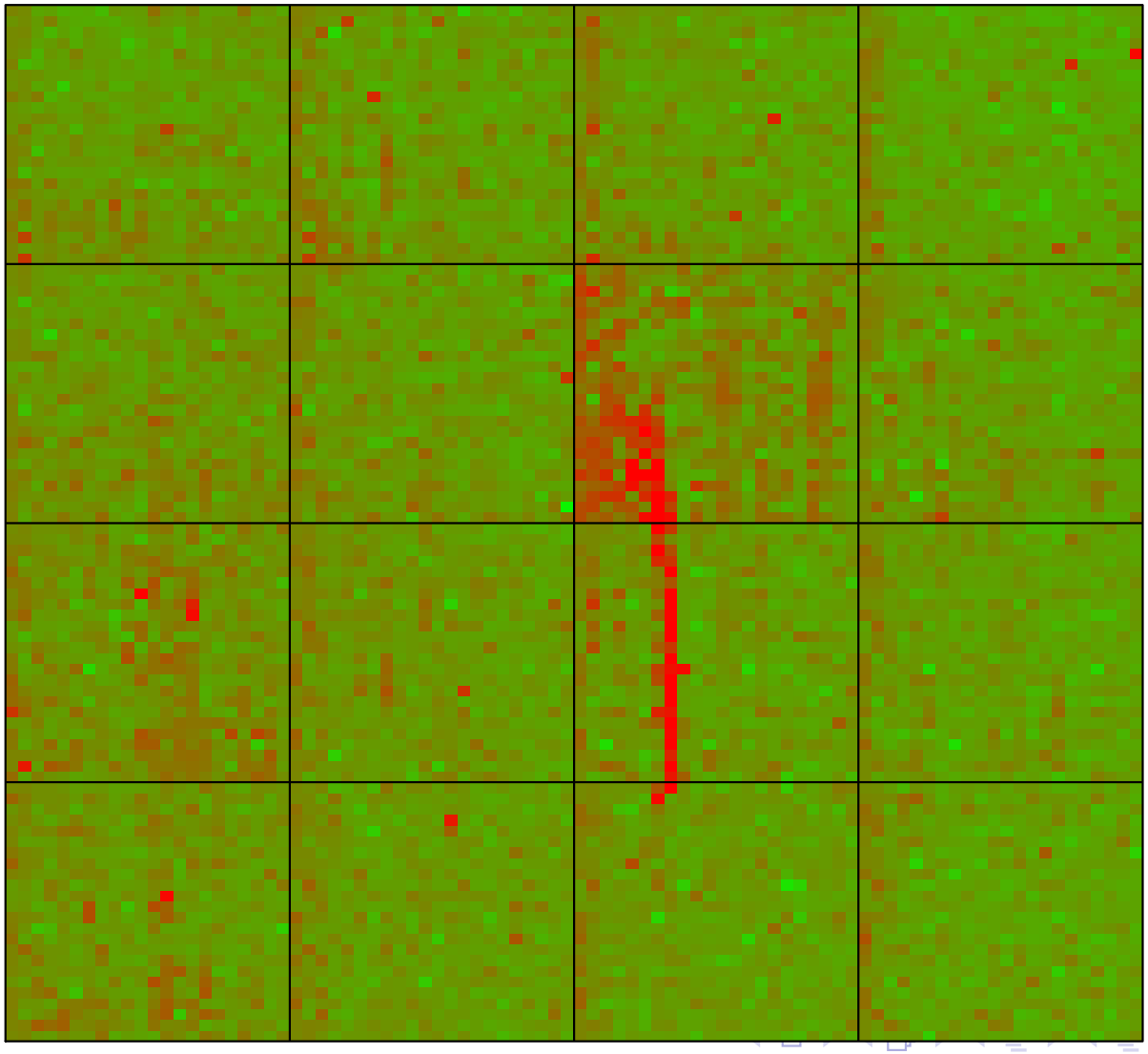
- Solo vimos la info del primer arreglo, pero nos dimos cuenta de que hay que normalizar los datos.
- Usemos la función normalizeWithinArrays, la cual nos normaliza los datos con sus *log-ratio* con tal de que la media de estos sea 0 en cada arreglo.

```
> MA <- normalizeWithinArrays(RG,
+      method = "none")
```

- Ahora veamos una imagen del primer arreglo ya normalizado:

```
> imageplot(MA$M[, 1], RG$printer,
+      zlim = c(-3, 3))
```

imageplot: normalizado



z−range −2.7 to 4.4 (saturation −3, 3)

o_O

- La función `imageplot` nos rota al arreglo, por lo que el grupo de hasta abajo a la izquierda es el primero.
- En la última gráfica podemos ver que hay una raya roja. Esto nos dice que había polvo o que el microarreglo estaba rayado.
- Los puntos que están en esa zona del artefacto, pues tendrán valores sospechosos.