Análisis de datos de Citometría de flujo con R-Bioconductor. Aplicación al estudio de marcadores de funcionalidad celular en toxicología.

Máster en Aproximaciones Moleculares en Ciencias de la Salud

Trabajo Fin de Master

RAMÓN TAMARIT AGUSTÍ

Análisis de datos de Citometría de flujo con R-Bioconductor. Aplicación al estudio de marcadores de funcionalidad celular en toxicología.

Máster en Aproximaciones Moleculares en Ciencias de la Salud

Trabajo Fin de Master

RAMÓN TAMARIT AGUSTÍ

Resumen

El alto rendimiento es una de las principales demandas para el screening de tóxicos *in vitro*. Los avances en instrumentación en citometría de flujo y su carácter multiparamétrico proporcionan un entorno óptimo para experimentación de alto rendimiento. La gran cantidad de información generada por estos medios tiene que ser almacenada, estructurada y analizada para hacerla accesible al investigador. El software estadístico de código abierto R, conjuntamente con las librerías de Bioconductor orientadas a la citometría de flujo, juega un papel importante dentro de este modelo, que proporciona una plataforma de investigación unificada a los bioinformáticos e investigadores en biomedicina para desarrollar los estándares o crear nuevos métodos.

El objetivo del presente trabajo es evaluar la idoneidad de las bibliotecas flowCore para el analizar un estudio multiparamétrico de citotoxicidad con Cd^{2+} y Hg^{2+}. Este trabajo muestra cómo utilizar el paquete flowCore junto con R para la importación, preprocesado, compensación, y el análisis de archivos de FCS. Los scripts que se presentan sirven como ejemplos para el análisis de grandes conjuntos de datos y pueden ser reutilizados, ampliados y adaptados para posterior uso. El presente trabajo también emplea las herramientas de visualización del paquete flowViz y extiende estas herramientas mediante la creación de nuevas funciones.

El presente estudio revela que los paquetes de R-biocondutor para tratamiento de datos de citometría de flujo son adecuados para analizar los ensayos multiparamétricos de citotoxicidad.

Análisis de datos de Citometría de flujo con R-Bioconductor. Aplicación al estudio de marcadores de funcionalidad celular en toxicología.

Máster en Aproximaciones Moleculares en Ciencias de la Salud

Trabajo Fin de Master

Abstract

One principal demand for in vitro screening for toxic effects is the high throughput of test methods. Flow cytometry offers the possibility to study several parameters simultaneously and technological advances in instrumentation provide the basis for high-throughput experimentation. The large amount of information generated must be stored, managed, and needs to be analysed in order to make it accessible to the researcher. The open source statistical software R in conjunction with the Bioconductor flow cytometry libraries can play an important role in this new paradigm by providing a unified research platform which bioinformaticians and biomedical scientists can use to develop standard or novel methods.

The aim of the present work is to assess the suitability of flowCore libraries for the analysis of a multi-parametric cytotoxicity assay with Cd^{2+} and Hg^{2+}. This work shows how to use the flowCore package in conjunction with R for importing, pre-processing, compensating, and analysing FCS files. The programmed scripts can serve as examples for analysing large datasets and can be reused, extended and adapted to future use. The work also employs some visualization tools of the flowViz package and extends these tools creating new functions.

The present study reveals that R-biocondutor packages for Flow Cytometry are well suited to analyse multi-parametric cytotoxicity assays.

1 Introducción

1.1 Alcance y objetivos del trabajo

El presente trabajo es el resultado del "proyecto de iniciación a la investigación" realizado por Ramón Tamarit Agustí dentro del máster en Aproximaciones Moleculares en Ciencias de la Salud. El trabajo se ha llevado a cabo en el Departamento de Citómica del Centro de Investigación Príncipe Felipe (CIPF).

El objetivo general del proyecto es el análisis de datos toxicología *in vitro* obtenidos por citometría de flujo usando R-Bioconductor. Bioconductor es un entorno de desarrollo de software libre y de código abierto para el análisis y comprensión de datos biológicos. El proyecto bioconductor se inició en el 2001 y actualmente integra librerías para el análisis estadístico de datos en el campo de la biología molecular. Está basado en el lenguaje de programación estadística R.

Las librerías para el análisis de datos de citometría de flujo de bioconductor son relativamente recientes. La primera versión operativa data de 2007 y el primer artículo descriptivo fue publicado en 2008 (Sarkar et al., 2008). Las pocas aplicaciones existentes en la bibliografía abordan el problema clásico del inmunofenotipado. Sin embargo, la problemática del análisis de datos toxicológicos es diferente. No sólo interesa separar mediante filtros las poblaciones celulares sino también cuantificar y comparar la intensidad de fluorescencia de múltiples experimentos mediante técnicas estadísticas.

El protocolo clásico de análisis de datos en citometría de flujo requiere: compensar y filtrar uno a uno los ficheros de datos, extraer la información, volcar la información en hojas de cálculo y analizarla. Para cada grupo de experimentos o paneles hay que repetir el proceso con el consiguiente coste en tiempo. Una de las ventajas de R-Bioconductor es que se puede construir un protocolo (en forma de script o programa) que realiza todas las operaciones en un solo bloque y puede ser reutilizado. Para ello son necesarios conocimientos tanto de la técnica experimental como de lenguajes de programación. La construcción de un script en R es un proceso que requiere tiempo, sin embargo, una vez construido se puede reutilizar cuando se necesite. De esta forma el tiempo invertido en el desarrollo del script se gana cuando hay que reanalizar los datos.

Este es un proyecto en bioinformática básica. Aunque se aborda un problema biológico concreto, los objetivos del proyecto están principalmente orientados a desarrollar metodología

bioinformática que se plasma en scripts en lenguaje de programación en R. Dichos objetivos se enumeran a continuación:

1. Obtener y aplicar las matrices de compensación de un grupo de experimentos de toxicología y evaluar su aplicabilidad.
2. Resolver la problemática que supone la transformación logarítmica de los datos que realiza el citómetro de flujo.
3. Ampliar las funcionalidades de visualización de R-Biocondutor para adaptarlas a los requerimientos del análisis de los datos.
4. Desarrollar scripts de código en R para analizar experimentos masivos de toxicología mediante citometría de flujo sin la intervención del usuario.

Aunque el trabajo se ha centrado en un grupo concreto de experimentos de funcionalidad celular tras el tratamiento con Cd^{2+} y Hg^{2+}, los métodos resultantes son aplicables a cualquier tipo de tratamientos.

En la segunda parte de este capitulo se encuentra una pequeña introducción a los paquetes de R-Bioconductor. La tercera parte del capítulo se centra en explicar las herramientas matemáticas básicas para entender el epígrafe de "resultados y discusión". La descripción, fundamentos (Givan, 2004) y aplicaciones de la citometría de flujo están ampliamente descritos en la bibliografía (Bashashati and Brinkman, 2009; Herrera et al., 2007; Smith et al., 2007; Valet, 2006) y se pueden consultar en las referencias proporcionadas, por lo que no se discuten en este documento.

Normalmente las aplicaciones bioinformáticas de R-Bioconductor se documentan en forma de "vignettes" (http://www.bioconductor.org/docs/vignettes.html). Una "vignette" es un documento que contiene descripción funcional del script. Está orientada al "usuario" con la intención de que a partir del código incluido pueda reproducirlo y aplicarlo a sus propios ejemplos. Parte del capitulo de "resultados y discusión" está redactado con esta filosofía. En el Anexo se han incluido parte de los scripts desarrollados en este trabajo.

1.2 Módulos de Bioconductor para el análisis de datos de citometría de flujo.

Impulsada por la FICCS (Flow Informatics and Computational Cytometry Society) (Lee et al., 2008), la implementación de la capa de análisis de datos de citometría de flujo se está realizando dentro del proyecto Bioconductor, empleando R como herramienta de programación. El resultado es un conjunto de paquetes (Hahne et al., 2009; Lee et al., 2009; Lo et al., 2009; Sarkar et al., 2008; Strain et al., 2009) que resuelven las necesidades básicas de análisis

estadístico de cualquier estudio de citometría de flujo, ya sea los tradicionales o los de alto rendimiento. Para más detalles se puede consultar http://www .bioconductor.org/docs/ workflows /flowoverview/

Módulos bajo flowCore	
flowCore	**Es el módulo principal encargado de importar y preprocesar los datos. Los objetos generados por este módulo se pueden analizar mediante las implementaciones del resto de módulos.**
flowViz	Métodos gráficos para la visualización gráfica
flowQ	Control de calidad de los datos
flowStats	Métodos estadísticos adicionales a flowCore
flowClust	Clustering mediante "t mixture models with Box-Cox transformation"
flowFP	Creación de huellas dactilares a partir de datos de Citometría de flujo.

Tabla 1.- Módulos para el análisis de datos de Citometría de Flujo con R-Bioconductor

Las ventajas de R-Bioconductor sobre otras plataformas son incuestionables: Dispone de toda la experiencia de las implementaciones de microarrays, es software libre, todas las técnicas estadísticas están implementadas a bajo nivel, es multiplataforma, es fácilmente integrable en pipelines. En definitiva, cumple todos los requisitos necesarios para integrarse como herramienta bioinformática para la investigación básica.

Los datos generados por la mayoría de los citómetros de flujo comerciales se almacenan en el formato FCS (Spidlen et al., 2010). Dentro de este formato, los valores de fluorescencia, dispersión frontal (FSC) y lateral (SSC) se almacenan en modo lista (Leary, 2001). Los ficheros de datos FCS tienen una cabecera de texto seguida de una matriz con los valores experimentales y finaliza con un bloque de texto en donde se incluyen los protocolos de análisis de datos (Spidlen et al., 2006; Spidlen et al., 2010). La tarea principal del paquete *flowCore* es la adquisición, representación y manipulación básica de los ficheros FCS. Esto se logra a través de un modelo de datos similar al adoptado por otros paquetes de bioconductor (en concreto con las técnicas de microarrays).

La unidad básica de manipulación e información en *flowCore* es el *flowFrame*, que se corresponde con un solo archivo FCS (un tubo de experimento). Un *flowFrame* se compone de los "*slots*": De "expresión" que con contienen la información a nivel de eventos (los resultados de fluorescencia de cada célula detectada), y de "parámetros" que contiene los metadatos

respectivamente. Los datos de fluorescencia se almacenan como una matriz y pueden ser fácilmente manipulados mediante los métodos comunes de bioconductor y R, como por ejemplo el método "*exprs()"*. El *slot* de parámetros contiene la información recuperada de los campos de texto del fichero FCS. Por ejemplo, los métodos "*featureNames()"* y "*colNames()"* devuelven (normalmente) los marcadores de fluorescencia y los nombres asignados a los canales respectivamente. La mayoría de los experimentos (varios tubos) consisten en varios objetos *flowFrame*, que se organizan mediante un objeto *flowSet*. Esta clase proporciona un mecanismo eficaz para garantizar que los metadatos experimentales se relacionan adecuadamente con cada flowFrame.

Gran parte de la visualización más sofisticada de los *flowFrame* y objetos *flowSet*, se lleva a cabo por el paquete *flowViz*. La lista de métodos de visualización de *flowViz* es extensa, prácticamente se pueden reproducir todos los gráficos existentes en la bibliografía.

La tarea más común en el análisis de los datos de citometría de flujo es el filtrado (o gating), ya sea para obtener estadísticas de resumen sobre el número de eventos que cumplan determinados criterios o para realizar nuevos análisis en un subconjunto de los datos. La mayoría de las operaciones de filtrado se componen de una o más operaciones. La definición de los filtros ("gates") en *flowCore* sigue la "*Gating Markup Language Candidate Recommendation*" Spidlen et al. (Spidlen et al., 2008), por lo que cualquier estrategia de filtrado de *flowCore* puede ser reproducida por cualquier otro software que cumpla el estándar.

Los filtros más simples, son los "*gates*" geométricos, que corresponden a los que se suelen encontrar en el software interactivo de la citometría de flujo, como son los: filtros marginales, rectangulares, poligonales y elipsoidales. Adicionalmente, se introduce el concepto de filtros generados por la distribución estadística de los datos o "*data-driven gates*", concepto que no se encuentra bien definido en el software comercial de citometría de flujo. En el enfoque de "*data-driven gates*" los límites de las poblaciones se calculan sobre la base de las propiedades estadísticas subyacentes. Por ejemplo, mediante un ajuste a distribución normal o por la estimación de la función de densidad: el filtro "*norm2Filter"* es un método robusto para encontrar la región que más se asemeja a una distribución normal bivariada, y el filtro "*kmeansFilter"*, identifica las poblaciones sobre la base de un agrupamiento k-dimensional.

Una vez seleccionadas las poblaciones de interés el análisis de los datos no está limitado por ninguna barrera de software. En este sentido las posibilidades de R cubren cualquier tipo de análisis estadístico o matemático necesario para interpretar los datos.

1.3 Marcadores funcionales fluorescentes. Modelo matemático para el tratamiento de la intensidad de fluorescencia

En citometría de flujo un marcador funcional fluorescente es una sustancia que modifica su emisión de fluorescencia como respuesta a la variación de la actividad funcional de celular. El marcador es una molécula formada por un fluorocromo y un grupo sensible a la funcionalidad (Johnson, 1998; Johnson, 2001). Las aplicaciones de la citometría de flujo en toxicología emplean marcadores de este tipo para evaluar el efecto tóxico sobre la funcionalidad celular (Herrera et al., 2007). Para evaluar la variación de la intensidad de fluorescencia entre tratamientos es necesario disponer de modelos matemáticos que expresen las relaciones entre ambos.

En un citómetro de flujo convencional la señal luminosa de una célula marcada con un solo fluorocromo tiene tres componentes:

1. Auto-fluorescencia celular (*A*): debida a las moléculas fluorescentes generadas por la propia célula en condiciones normales.
2. Fluorescencia intrínseca (*F*) debida a un marcador fluorescente.
3. Ruido de fondo (*R*): debido a fotones de luz que llegan al detector y que no han sido emitidos por la célula. Normalmente la electrónica del equipo elimina parte de este ruido de fondo con el llamado "baseline-restorer".

La intensidad de fluorescencia (*s*) se pude expresar como:

$$s = F + R + A \qquad \text{(Ec. 1)}$$

La intensidad de fluorescencia intrínseca del marcador es proporcional al número de moléculas (*N*) y a su rendimiento de fluorescencia (*f*, número de fotones emitidos por fotones de excitación).

$$F = f \bullet N \qquad \text{(Ec. 2)}$$

Por otra parte, la señal que se obtiene como resultado de un experimento de citometría corresponde a los fototes de un determinado rango de longitudes de onda seleccionados por un filtro. Los fotones seleccionados por el filtro llegan a la electrónica del citómetro, tubo foto-multiplicador y "*baseline- restorer*", que se encargan de amplificar electrónicamente la señal y restar el ruido de fondo ($R \approx 0$). La señal resultante *S* es linealmente proporcional a *s* según la calidad de los amplificadores del citómetro.

$$S = C \cdot s \qquad \text{(Ec. 3)}$$

$$S = C \cdot (f \cdot N + A) \qquad \text{(Ec. 4)}$$

La señal obtenida de esta forma no permite observar con facilidad la distribución de poblaciones resultante, por lo es amplificada logarítmicamente para que cubra entre 4 y 5 etapas (*E*) en una escala logarítmica. Posteriormente, la señal analógica es trasformada por el convertidor Analógico-Digital en números de canal. El rango de números de canal (*Cn*) depende de su resolución. Por ejemplo para 16 bits de resolución tendremos 4096 canales numerados del 0 al 4095. Finalmente la intensidad de fluorescencia observada en números de canal viene dada por la ecuación (Ec. 5). Los valores de fluorescencia que se almacenan en los ficheros FCS tienen unidades de "número de canal"

$$O_{log} = \text{int}[(\text{Log10}\,(C \cdot (f \cdot N + A)) \bullet E/Cn] \qquad \text{(Ec. 5)}$$

La relación entre la fluorescencia observada (O_{log}) y el número de moléculas de fluorocromo (*N*) está afectada por las siguientes aproximaciones debidas a la electrónica del equipo:

1. Los fotomultiplicadores amplifican linealmente en un determinado margen de trabajo. Por tanto, la ecuación (Ec. 4) es únicamente valida previa calibración del los fotomultiplicadores (Wood, 1998)
2. Los "baseline-restorers" pueden no funcionar linealmente si la intensidad de autofluorescencia que les llega es muy baja, entonces $r>0$ ó $r<0$
3. El amplificador logarítmico puede no hacer la conversión exactamente a un número entero de etapas o bien la base (base de logaritmo) puede nos ser exactamente 10

Las anteriores aproximaciones se pueden evitar calibrando la respuesta del citómetro frente a microesferas que adhieren cantidades controladas de fluorocromo, siendo posible construir una función de calibrado con los parámetros correctos de amplificación(Gratama et al., 1998).

Como se aprecia en la ecuación (Ec. 5) la fluorescencia observada en unidades de "numero de canal" (O_{log}) es proporcional al logaritmo del número de flourocromos (*N*). La fluorescencia observada se puede linealizar mediante la ecuación (Ec. 6).

$$O_{lin} = 10^{(O_{\log} \cdot \frac{Cn}{E})} \qquad \text{(Ec. 6)}$$

La intensidad de fluorescencia linealizada es directamente proporcional al número de moléculas de marcador que emiten fluorescencia mediante la ecuación (Ec. 7). La ecuación (Ec. 7) tiene dos sumandos, el primero es la aportación de la fluorescencia del fluorocromo y el segundo es la aportación de la fluorescencia celular.

$$O_{lin} = C \cdot f \cdot N + C \cdot A \qquad \text{(Ec. 7)}.$$

Uno de los principales problemas asociados a la linealización de la fluorescencia amplificada logarítmicamente es la pérdida de resolución en los valores que se encuentran a partir del canal 2000 (segunda década). Los valores de la Tabla 2 muestran la pérdida de resolución. En la tabla las columnas "*Lin. n.chann*" y "*Lin. n.chann + 1*" contienen el valor linealizado mediante la ecuación (Ec. 6) del canal "*Log n.chann*" y de su canal siguiente. Por ejemplo, mientras que entre el canal 1 y 2 en escala logarítmica hay una diferencia de 0.0023, en el canal 2000 es de 0.2021. Este efecto se denomina "striped-data" (Roederer, 2001).

Log n.chann	Lin n.chann.	Lin n.chann. + 1	Diferencia
1	1.0023	1.0045	0.0023
300	1.9632	1.9676	0.0044
1000	9.475	9.496	0.0213
2000	89.77	89.97	0.2021
4095	9977	10000	22.460

Tabla 2- Efecto de la linealización de la amplificación logarítmica.

1.4 La compensación de los canales de fluorescencia.

Una de las ventajas de la citometría de flujo es la posibilidad de medir simultáneamente la señal de varios fluorocromos (Herrera et al., 2007). A cada fluorocromo se le asigna un "canal" mediante un filtro que coincida aproximadamente con su pico de máxima emisión. Sin embargo, el espectro de emisión del resto de flourocromos empleados en el marcado puede estar solapado (ver Figura 1) con el pico. La señal observada en cada uno de los canales es la suma de las aportaciones individuales de los fluorocromo presentes en la muestra. Al proceso de eliminación de la señal que no corresponde al fluorocromo principal de cada canal se denomina compensación.

Para calcular la compensación se supone que el desdoblamiento de la fluorescencia intrínseca del canal primario en los secundarios tiene una relación lineal (Hahne et al., 2010; Roederer, 2001; Shackney et al., 2006; Wood, 1998). Por ejemplo para un experimento con cuatro canales, la fluorescencia total cada canal se expresa por el conjunto de ecuaciones (Ecs. 8):

$$T_1 = f_{11}\,O_1 + f_{21}O_2 + f_{31}O_3 + f_{41}O_4 \qquad \text{(Ecs. 8)}$$

$$T_2 = f_{12}\,O_1 + f_{22}O_2 + f_{32}O_3 + f_{42}O_4$$

$$T_3 = f_{13}\,O_1 + f_{23}O_2 + f_{33}O_3 + f_{43}O_4$$

$$T_4 = f_{14}\,O_1 + f_{24}O_2 + f_{34}O_3 + f_{44}O_4$$

El conjunto de ecuaciones (Ecs. 8) son una aproximación de la ecuación (Ec. 7). *T* es la señal no compensada, *f* es el factor de compensación y O es la señal compensada. El sistema de ecuaciones en formato matricial se puede expresar como

$$\mathbf{T} = \mathbf{F} \times \mathbf{O} \qquad \text{(Ec. 9)}$$

El desdoblamiento espectral de un fluorocromo (factores *f* en las Ecs. 8) se puede estimar a partir de una muestra marcada únicamente con ese fluorocromo ("single-stain") y con una muestra no marcada con ningún fluorocromo ("un-stained"). La autofluorescencia celular en cada canal se estima a partir de la mediana de la población celular en la muestra un-stained, que corresponde al segundo término de la ecuación (Ec. 7). La fluorescencia intrínseca del fluorocromo se estima a partir de la mediana de la población de la muestra SS en su canal principal restándole la autofluorescencia obtenida anteriormente. La cantidad de señal que se desdobla en cada canal secundario se calcula como diferencia entre la intensidad observada y la autofluorescencia en ese canal. Los valores así obtenidos son una estimación de la matriz **F**.

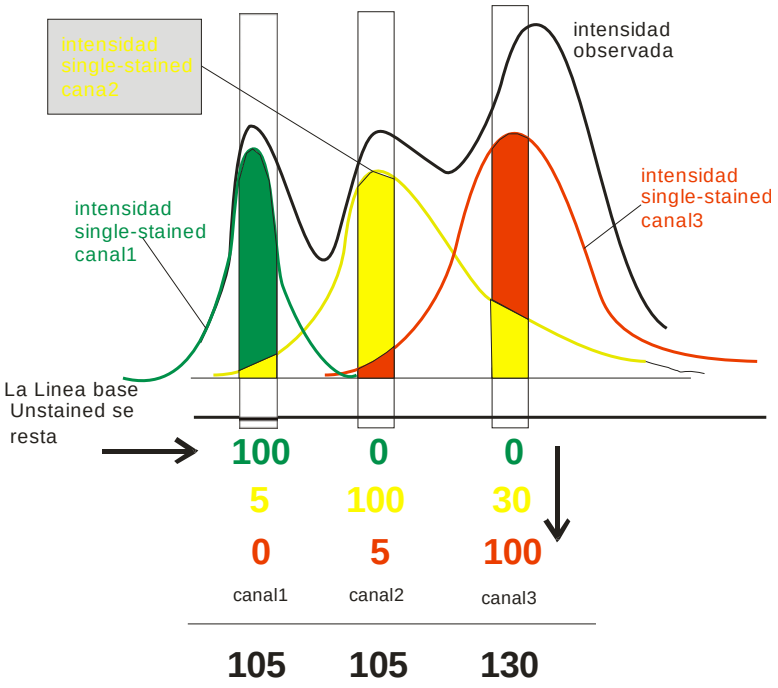

Figura 1. Solapamiento espectral y compensación de fluorescencia.

Dado que el objetivo son los valores "compensados" es necesario obtener la matriz inversa de **F**. Los valores compensados se obtienen entonces multiplicando la matriz T por $^{-1}\mathbf{F}$, ecuación (Ec. 10)

$$^{-1}\mathbf{F} \times \mathbf{T} = \mathbf{O} \qquad \text{ecuación (Ec. 10)}$$

La validez del modelo está sujeta a las suposiciones realizadas, entre las cuales las más importantes son:

1. La señal primaria se desdobla de forma lineal
2. La auto-fluorescencia celular se mantiene constante en todas las condiciones experimentales
3. Los parámetros de captura del citómetro son reproducibles durante toda la serie de experimentos. La modificación de los parámetros de adquisición invalida la matriz de compensación

Los valores compensados de esta forma no eliminan las posibles correlaciones o interferencias entre los marcadores funcionales. La medida conjunta de dos parámetros puede estar interferida por la competición química entre sustratos o por fenómenos foto-físicos entre los fluorocromos de los marcadores (*quenching*).

2 Materiales y métodos

2.1 Procedencia y características de los experimentos de Citometría de flujo

Los experimentos de citometría de flujo analizados en este trabajo fueron facilitados por la unidad de Citómica del CIPF en forma de ficheros FCS. Los experimentos forman parte de un estudio multiparamétrico de toxicidad con Cd^{2+} y Hg^{2+} en cultivos de una línea celular de neuroblastoma humano (SH-SY5Y). Los protocolos de cultivo y marcado empleados son propiedad intelectual de la unidad de Citómica del CIPF y no se facilitan en este trabajo.

Tras el tratamiento con los tóxicos, las células se marcaron con diferentes fluorocromos, con el fin de analizar distintos parámetros celulares.

Marcador	**Actividad**
DCF -Dihidrodiclorofluoresceína diacetato	Actividad peroxidasa
TMRM - Tetrametilrodamina	Potencial de membrana mitocondrial
Indo-1	Calcio intracelular.
BODIPY 675	Peroxidación lipídica
DIBAC3(4)	Potencial de membrana plasmática.
MCB – Monoclorobimano	Niveles de tioles libres, esencialmente glutation.
DAF – Diaminofluoresceína diacetato	Niveles de óxido nítrico
Mitosox	Superóxido mitocondrial

Tabla 3. Marcadores de la funcionalidad celular empleados en el estudio

Se facilitaron tres paneles de marcaje. En la Tabla 3 se muestran los marcadores de la funcionalidad celular empleados en el estudio. Cada panel combina 4 fluorocromos distintos. La asociación de los fluorocromos con los canales de fluorescencia y su agrupación en paneles se indica en la Tabla 4.

Panel	FL1 (B)	FL3 (C)	FL6 (D)	FL8 (E)
1	DCF	TMRM	Indo-1	BODIPY 675
2	$DiBAC_3(4)$	Mitosox	MCB	BODIPY 675
3	DAF	TMRM	MCB	BODIPY 675

Tabla 4. Asignación de los marcadores a los canales de fluorescencia en los paneles y agrupación en paneles.

Cada panel de experimentos con los tóxicos Cd^{2+} y Hg^{2+} se compone de dos subgrupos de experimentos: (1) Experimentos que han sido marcados con los 4 fluorocromos del panel, se denominan "full-stain" (FS) y se marcan como paneles A, y (2) Experimentos donde sólo se ha utilizado un fluorocromo, se denominan "single-stain" (SS), se marcan como B, C, D y E dependiendo del fluorocromo utilizado (Tabla 4). Adicionalmente cada panel tiene un control "blanco" que no ha añadido ningún marcador. Las concentraciones de Cd^{2+} y Hg^{2+} de cada subgrupo se pueden consultar en la Tabla S1 del anexo.

Adicionalmente, se facilitó otro grupo de experimentos realizados con tóxicos que tienen actividad conocida (controles positivos) sobre alguno de los marcadores empleados (Tabla 5). Este grupo de experimentos se realizaron con los mismos protocolos y línea celular que los de Cd^{2+} y Hg^{2+}. Del grupo de controles positivos únicamente se facilitaron experimentos full-stain y single-stain del canal en donde se espera una variación significativa.

Tóxico	**Actividad**	**Afecta**	**Respuesta esperada**
CHP-Cumene hidroperóxido	Aumenta los niveles de peróxidos intracelulares.	DCFH, FL1 panel 1	⇧
FCCP	Desacoplador de la cadena de transporte electrónico mitocondrial.	TMRM, FL3 panel 1 y 3	⇩
Ionomicina	Ionóforo de calcio, que provoca el aumento de calcio intracelular.	Indo-1, FL6 panel 1	⇧
Gramicidina	Provoca la pérdida de la polaridad de la membrana plasmática	DiBAC FL1 panel 2	⇧
Paraquat	Aumento de los niveles de superóxido intracelular	MitoSox FL3 panel 2	⇧
BSO - Buthionine sulfoximide	Hace disminuir los niveles de tioles intracelulares libres (glutation)	MCB FL6 panel 1 y 3	⇩
NOR-1	Es un dador de óxido nítrico	DAF FL1 panel 3	⇧

Tabla 5 Tóxicos con actividad conocida empleados en el estudio. La columna "Afecta" indica el marcador y canal de florescencia en donde se espera una variación significativa de la fluorescencia. Las flechas en la columna "Respuesta esperada" indican aumento o disminución de la fluorescencia.

Todos los experimentos se analizaron con un citómetro modelo MoFlo (Beckman-Coulter, CA, USA) manteniendo constantes los parámetros de adquisición (ganancias y voltajes amplificadores y potencia del láser) en todos los experimentos. Los filtros de emisión utilizados

fueron 520 nm para la FL1 (DCFH, $DiBAC_3(4)$ y DAF), 615 nm para la FL3 (TMRM y MitoSox), 405 nm para la FL6 en el caso del Indo-1, 450 nm para la FL6 en el caso del MCB y 670 nm para el BODIPY 675.

2.2 Instalación de R-bioconductor, sistemas operativos, detalles computacionales y requerimientos generales de Hardware.

El procesado de los ficheros FCS y los cálculos estadísticos se realizaron con R versión 2.10.2. Las librerías necesarias (Código 1) para ejecutar los scripts corresponden a la revisión 2.6 de Bioconductor (http://www.bioconductor.org/News/bioc_2.6_release). El proceso de instalación de R y bioconductor se puede consultar en http://www.bioconductor.org/docs/install/.

```
##################################################
### chunk: Load Libraries
##################################################
 library(flowCore)
 library(flowQ)
 library(flowViz)
 library(flowStats)
 library(flowUtils)
 library(geneplotter)
 library(colorspace)
 library(grid)
 library(MASS)
 library(flowFP)
 library(geneplotter)
 require(RColorBrewer)
 library(MASS)
 library(KernSmooth)
```

Código 1. *Conjunto de librerías necesarias para ejecutar los scripts.*

Los scripts de este trabajo se instalaron y comprobaron en los sistemas operativos Windows Vista (SP1) y XP (SP3). Los requerimientos de memoria de R para procesar los scripts que se incluyen en el anexo requieren modificar la asignación realizada por defecto durante la instalación a 2000 Gb (*memory.size(max = 2000)*). No es recomendable aumentar la memoria asignada a R más de 2000 Gb ya que se pueden producir fallos inesperados del sistema operativo. En condiciones normales cualquier PC con al menos un procesador Intel Celeron 2 Ghz y 3 Gb de RAM es capaz de ejecutar los scripts aquí presentados. No obstante, el renderizado ("generación de la imagen digital") de los ficheros con scatter-plots de todos los paneles puede llevar entre 1 y 3 horas de cómputo (dependiendo del procesador y la memoria libre disponible).

2.2.1 Advertencia sobre los puntos de depuración e inspección

Durante el diseño de un script en R es necesario incluir trozos de código intermedios necesarios para la depuración o inspección de los resultados intermedios (*"debug- snippets"*). La ejecución condicional de los *debug-snippets* se consigue introduciendo una variable global cuyo valor se inicializa al comienzo del script. Los scripts que se adjuntan en el Anexo emplean la variable "*ifDebug*" para implementar esta funcionalidad. Esta funcionalidad no debe habilitarse en los cálculos masivos (el script completo) ya que se producirá un fallo del sistema. La funcionalidad se anula modificando el valor de la variable *IfDebug* a FALSE (Código 2)

```
              ## Si ifDebug="S" se ejecutan diferentes tests necesarios para
              ## depurar el código y visualizar los resultados intermedios.
              ifDebug <- FALSE
              print(paste("### ifDebug <- " , ifDebug ) )

       if (ifDebug) {

              ## Aquí se ejecutan las funciones necesarias para la depuración

                     }
```

Código 2. Implementación de la funcionalidad *ifDebug*.

Adicionalmente, en las estructuras iterativas es posible añadir estructuras comentadas con la indicación "*for debug*" que se emplean para ejecutar parte del código durante la etapa de depuración.

```
              ##for debug panelFile <- 2
```

2.3 Descripción de los scripts del anexo.

En la versión electrónica de este trabajo se encuentran los listados de los principales scripts empleados en el epígrafe de resultados y discusión.

- *head.repair.R*: Script para la homogeneización de cabeceras para la carga conjunta de múltiples ficheros FCS en un flowSet.
- *fluo.compensation.R*: Cálculo de la compensación de fluorescencia en ficheros FCS con amplificación logarítmica y calculo de estadísticas.
- *Analisis.compensated.R:* Cálculo de la mediana de la intensidad de fluorescencia sobre la población positiva y negativa de cada marcador.
- *Analisis.positive.ctrl.R:* Cálculo comparativo de la variación en la intensidad de fluorescencia de Cd y Hg con marcadores positivos.

- Librería: *utils.funtions.R* : Librería que contiene las funciones de apoyo para visualización y tratamiento de datos
- Chunk: *positive.selection.R* Realiza una selección de poblaciones positivas y negativas mediante clustering.

3 Resultados y discusión

3.1 Descripción del problema. Flujo de trabajo.

El objetivo de los scripts que rediscuten en este capítulo es determinar la variación de los marcadores de funcionalidad celular (Tabla 3) con la concentración de cadmio (Cd) o mercurio (Hg) y comprobar si la variación de los marcadores se puede medir de forma multiparamétrica. La variación de los marcadores se mide en la población de células viables que sobreviven al tratamiento con tóxico.

El flujo de trabajo en los scripts desarrollados realiza las siguientes operaciones:

1. Preprocesar los ficheros FCS
2. Cargar los ficheros FCS por paneles.
3. Calcular la matriz de compensación de cada panel y compensar los experimentos.
4. Guardar los ficheros compensados y matrices de compensación para posterior uso.
5. Seleccionar la población de células viables y calcular las medias y medianas de la intensidad de fluorescencia.
6. Generar ficheros con los datos de medias y medianas.
7. Generar ficheros con las graficas necesarias
8. Recuperar los ficheros compensados y realizar estadísticas adicionales.
9. Añadir nuevos ficheros (controles positivos) a los paneles, compensarlos y recalcular las estadísticas

En las secciones siguientes se detalla como se han implementado cada una de las etapas. Se incluye también una valoración biológica de los resultados.

3.2 Reparación de cabeceras.

Como paso previo a cualquier tratamiento de datos con "*flowCore*" los ficheros FCS obtenidos del citómetro tienen que procesarse para eliminar los canales no utilizados y homogeneizar los nombres de los canales de fluorescencia. Este proceso es necesario para facilitar la carga conjunta de todos los ficheros en el *flowSet*. Se realiza para todo el conjunto de datos con el script "*head.repair.R*" que se adjunta en el anexo.

El script "*head.repair.R*" toma input los ficheros FCS que residen en un directorio determinado y un fichero FCS que sirve como referencia para realizar la transformación. Como output se obtiene un conjunto de ficheros FCS con idéntica cabecera. En especial todos los parámetros de fluorescencia se renombran a FL1, FL3, FL6, FL8 en los tres paneles de experimentos. De esta forma es posible tratar conjuntamente en el mismo script todos los paneles de experimentos.

3.3 Carga de los datos en flowCore

Para cargar los ficheros de datos en *flowCore* se eligió emplear ficheros de anotaciones en modo texto. Un fichero de anotación es una matriz que contiene al menos una columna con los nombres de los ficheros FCS y tantas columnas como factores de identificación de las muestras se quieran añadir. Dependiendo de las necesidades, el fichero de anotaciones se puede subdividir en ficheros más pequeños. El diseño experimental se muestra en la Figura 2.

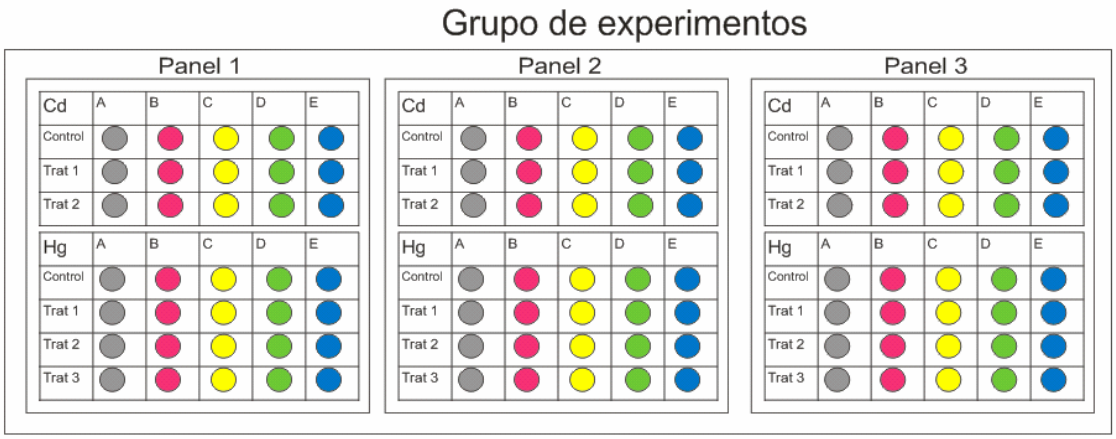

Figura 2. Diseño del grupo de experimentos

En concreto para cada panel se creó un fichero de anotaciones cuyas columnas contienen:

1. el nombre del fichero FCS
2. *Id*: Un identificador único del fichero: Es una cadena de texto que resume el tipo de experimento y que se utiliza para nombrar a los experimentos en las salidas de los scripts
3. *Date*: La fecha del experimento
4. *panel*: Es un número entre 1 y 3 que indica a qué panel corresponde el experimento

5. *StainId*: Es una letra que indica el marcado del experimento, A: todos los marcadores ("*Full-Stain"*), y B,C,D,E teñidos con un solo marcador (en adelante los nombraremos como "*single-stain"*) para FL1, FL3, FL6 y FL8 respectivamente
6. Tox: Es una cadena que identifica el tóxico empleado (Cd o Hg, los controles de fondo se marcan con la letra "N")
7. *CTox*: Es un número que indica la concentración del tóxico

La tabla XX del anexo muestra el fichero completo de anotación con todos los experimentos empleados en este trabajo.

Dado que cada subgrupo de experimentos puede necesitar diferentes variables para su ejecución, los ficheros de datos se agruparon en seis ficheros de anotación repartidos en dos matrices dependientes de un factor. Por ejemplo, en el programa "*fluo.Compensation.R*" los experimentos con Cd^{2+} y Hg^{2+} necesitan diferentes valores de las variables de visualización. La variable "*PanelToxico*" determina el panel de experimentos que se utilizará en los cálculos y los valores de las variables necesarios. Posteriormente, se puede iterar sobre la matriz que contiene los nombres de los ficheros de anotación para realizar los cálculos necesarios. El bloque constructivo que se muestra en (Código 3) se usa en los scripts para iterar sobre todo el grupo de experimentos.

```
## Matrices de anotación que contienen los experimentos con Cd y Hg separados.
## Cargar una u otra:las funciones de visualización necesitan parámetros diferentes
       PanelToxico <- "Cd"
       if (PanelToxico=="Cd"){
              matPanels <- c("cd_hg_annotation_bl_pnl1Cd.txt",
                             "cd_hg_annotation_bl_pnl2Cd.txt",
                             "cd_hg_annotation_bl_pnl3Cd.txt")
              ## Parámetros para la visualización Cd
              laycol <-c(3,5); my.layout=c(3,6); rango <- c(2:16)
              }

       if (PanelToxico=="Hg"){
              matPanels <- c("cd_hg_annotation_bl_pnl1Hg.txt",
                             "cd_hg_annotation_bl_pnl2Hg.txt",
                             "cd_hg_annotation_bl_pnl3Hg.txt")
              ### Parámetros para la visualización Hg
              laycol <-c(4,5); my.layout=c(4,6); rango <- c(2:21)
              }

              ## Inicio de la iteración a través de los paneles descritos en la matriz
              for (panelFile in 1:length(matPanels)) {
              print("################### Carga del Panel  ####################")
              ExpList <- matPanels[panelFile]
              flowData   <-   read.flowSet(path   =   ".",   phenoData   =   ExpList,
transformation=FALSE)
              ## Ejecutar funciones necesarias …..
              }
              ## Final de la iteración sobre matPanels
```

Código 3. Bloque constructivo para iterar sobre todo el grupo de experimentos.

3.4 Protocolo de compensación en R

El protocolo de compensación de los 6 grupos de experimentos se implementó en el script "*fluo.compensation.R*" que se adjunta en el anexo.

El script realiza las siguientes operaciones:

1. Lee el flowSet y linealiza los valores de fluorescencia.
2. Normaliza los canales FSC y SSC.
3. Selecciona un conjunto de células viables.
4. Calcula la matriz de compensación con los controles SS
5. Aplica la matriz de compensación al flowSet en escala lineal
6. Restablece la transformación logarítmica original del citómetro y almacena los experimentos compensados.

El script "*fluo.compensation.R*" genera diversos ficheros de resultados, entre los cuales cabe destacar:

1. Ficheros en modo texto con las matrices de compensación, valores de fluorescencia basal, medias y medianas de las poblaciones viables en escala lineal, logarítmica y lineal-logarítmica.
2. Ficheros pdf con las distribuciones normalizadas, representaciones tipo scatter plot entre los canales compensados y sin compensar para todas las combinaciones de canales.

El script tiene puntos de control y test que pueden ser desactivados mediante las opciones globales de la cabecera. A continuación y a modo de ejemplo extensible a los 6 grupos de experimentos se detalla el procedimiento de compensación del panel 3 con Hg, incidiendo en las partes más significativas del proceso.

3.4.1 Normalización de lo canales FSC y SSC

Como se aprecia en la Figura 3 las distribuciones de población en los canales FSC y SSC que definen las células viables, muertas y restos celulares son diferentes entre muestras de un mismo panel. Este hecho es normal entre muestras con tóxico, pero entre los controles de compensación (sin toxico) es un indicador de que existen variaciones técnicas entre experimentos.

Una opción implementada en *flowCore* es "normalizar" los canales que muestran variaciones técnicas. El procedimiento de normalización de *flowCore* (Hahne et al., 2010) ajusta

las distribuciones de densidad mediante clustering. Después de la normalización los máximos de las distribuciones en los canales SSC y FSC están correctamente alineados, mientras que el resto de canales se mantienen inalterados (Figura 3).

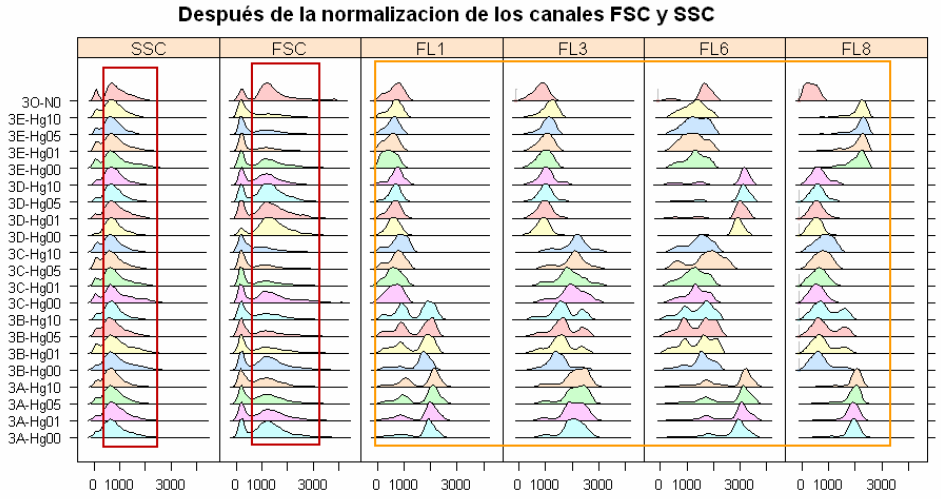

Figura 3 Efecto de la normalización en los canales FSC y SSC. Los cuadros en rojo muestran como las distribuciones de SSC y FSC quedan alineadas. Las distribuciones de los canales de fluorescencia no quedan afectadas por la normalización de FSC y SSC (recuadro naranja)

3.4.2 Selección de la población objeto de la compensación.

Uno de los problemas que se puede presentar durante la compensación es que las diferentes poblaciones morfológicas tengan distintos niveles de fluorescencia, siendo interesante seleccionar una única población bien definida en todos los controles de compensación. En este caso la población de interés son las células morfológicamente viables (Vermes et al., 2000) Para seleccionar esta población se realizan los siguientes pasos:

1. Previamente se eliminan los valores extremos del flowset original mediante un filtro ”*boundaryFilter*” como una forma de evitar que los valores extremos de las distribuciones ocasionen errores de cálculo debidos a división por cero.

Para seleccionar la población se emplean dos filtros.

2. Un primer filtro rectangular excluirá los restos celulares (“*debris*”), parte de las células muertas y los agregados celulares. Los valores fijados para este filtro son: FSC mínimo:500 , FSC máximo:2400 , SSC mínimo:200, SSC máximo: 2000 (Figura 4).
3. Un segundo filtro “*norm2filter*”, para crear una distribución normal en dos dimensiones. La distribución binormal se centra en la mediana de las

poblaciones y encierra una región que incluye el 95% de la población (es decir, 2 desviaciones estándar) El filtro se aplica en los canales FSC y SSC. El resultado de este filtro corresponde con las células viables (Figura 5).

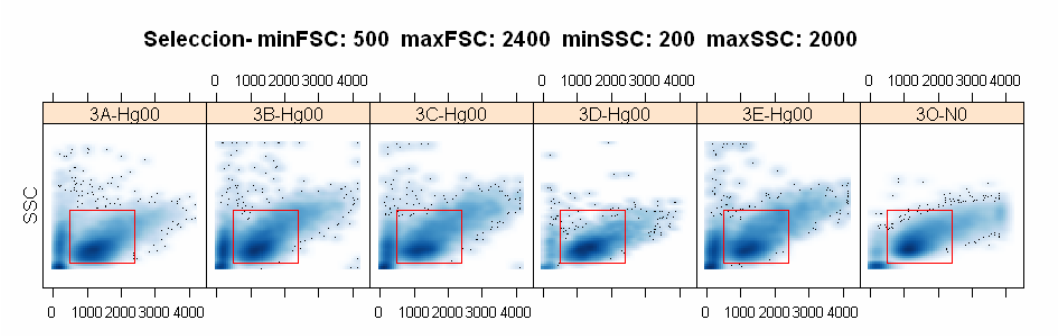

Figura 4. Eliminación de los restos celulares con un filtro rectangular en los canales FSC y SSC. Las preselección de células viables está encuadrada en rojo.

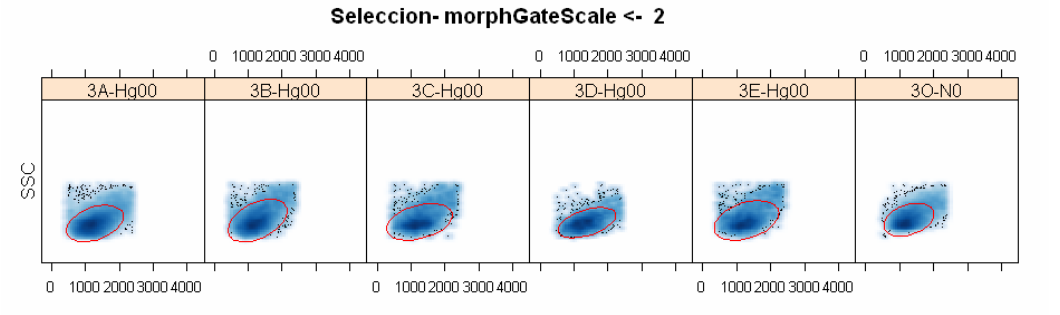

Figura 5. Selección de la población de células viables con un filtro "*norm2filter*". La población con mayor densidad queda enmarcada en rojo.

3.4.3 Revertir la amplificación logarítmica del citómetro

La amplificación logarítmica no permite calcular la matriz de compensación mediante el método tradicional, por lo que hay que transformar los datos de fluorescencia a escala lineal. Supondremos que en el rango de valores de la población de células viables el amplificador trabaja de forma "ideal", es decir, la fluorescencia en escala lineal se puede expresar según la ecuación (Ec. 6)(Gratama et al., 1998).

$$O_{lin} = 10^{(O_{\log} \cdot \frac{Cn}{E})} \qquad \text{(Ec. 6)}$$

En el caso que nos ocupa, el número de etapas logarítmicas (E = 4) y la resolución del citómetro (Cn=4096) se obtienen de las cabeceras de los ficheros FCS. La función "*AllgT*" implementada en "*utils.functions.R*" realiza la transformación de escala del flowset. Antes de

aplicar la transformación hay que modificar los parámetros de amplificación y escala de cada uno los *flowFrames*, ya que los métodos base de *flowCore* no realizan esta operación automáticamente (ver detalles de la implementación en el código de "*fluo.compensation.R"*).

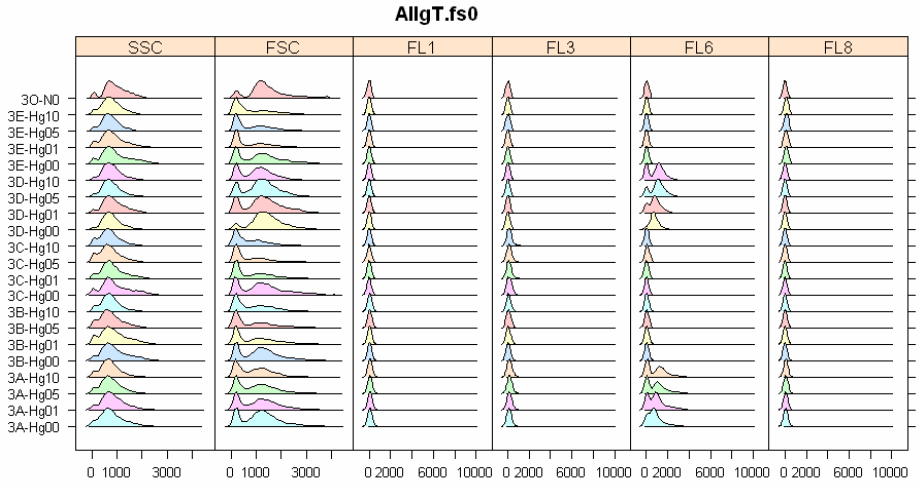

Figura 6. Distribución de densidad normalizada de los experimentos del panel 3 en escala lineal.

Como se aprecia en la Figura 6, una vez transformados a escala lineal las distribuciones de densidad normalizada de FL1, FL3 y FL8 los valores están apilados dentro del intervalo 0-500, haciendo imposible discernir a simple vista las poblaciones positivas y negativas. Para resolver este inconveniente se implementó la función *"visualizarHistogramas()"* en "*utils.functions.R*" que permite reescalar la visualización dentro de los márgenes definidos por el usuario. La implementación de esta función no hace uso de las funcionalidades de "*flowViz*" y es un ejemplo de cómo se pueden añadir funcionalidades según las necesidades. La Figura 7 muestra parte del resultado de la función.

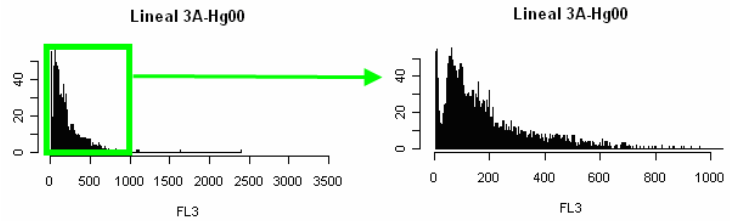

Figura 7. Ejemplo de salida de la función *"visualizarHistogramas()"*. Se observa la modificación de la escala de visualización del histograma.

3.4.4 Cálculo de la matriz de compensación

Para calcular la matriz de compensación se empleó el *flowSet* con los valores de fluorescencia en escala lineal.

El cálculo de la matriz implica los siguientes pasos:

1. Separar los *flowFrame* SS para cada canal incluyendo el control un-stained (sin marcadores).
2. Calcular las medianas de cada uno de los canales para cada uno de los controles. La matriz resultante contiene tanto los máximos de intensidad (si el *flowFrame* es un control positivo para ese canal), como los valores de la línea base (autofluorescencia + ruido)
3. El valor de la línea base se establece como el mínimo entre todos los controles sin contar con el un-stained.
4. Desplazar la mediana de cada control SS respecto de la línea base de cada canal
5. Por último, se recalculan las medianas de cada experimento SS y se estima la matriz de compensación resolviendo la ecuación matricial.

La matriz de compensación obtenida de esta forma se puede aplicar mediante los métodos base de *flowCore* únicamente a los flowSet en escala lineal y no a los que tienen la amplificación logarítmica.

3.5 Análisis de los resultados de los cálculos de las matrices de compensación.

La Tabla 6 muestra el resultado del cálculo de la matriz de compensación para cada conjunto de experimentos. Los valores obtenidos para el mismo panel con diferente tóxico son reproducibles en un 3% (los valores con una diferencia de 0.03 se muestran subrayados), exceptuando el desdoblamiento de FL6 en FL1 y FL8 en el panel 2 que es de un 6%.

A pesar de que los valores son reproducibles entre ambos paneles con un margen de error máximo del 6% hay que tener en cuenta que la compensación es únicamente valida en el entorno de las poblaciones positivas.

El cálculo de la matriz de compensación del Panel 2 es especialmente delicado debido a que el canal FL3 presenta una población de células viables positivas inferior a la del resto de parámetros y el valor de desdoblamiento queda sesgado. Mediante el "*chunk*" de código "*positive.selecction.R*" incluido en el Anexo, se puede modificar la funcionalidad del script

"*fluo.compensation.R*" de forma que se seleccione únicamente la población positiva (Figura 9). Las matrices de compensación obtenidas con esta modificación son similares en un 3% a los obtenidos con los controles de Cd (no se muestran los datos).

	Panel 1 Cd				**Panel 2 Cd**				**Panel 3 Cd**			
	FL1	FL3	FL6	FL8	FL1	FL3	FL6	FL8	FL1	FL3	FL6	FL8
FL1	**1,00**	**0,39**	**0,06**	**0,01**	**1,00**	**0,25**	**0,36**	**0,03**	**1,00**	**0,38**	**0,53**	**0,03**
FL3	**0,01**	**1,00**	**0,01**	**0,00**	**0,02**	**1,00**	**0,00**	**0,01**	**0,01**	**1,00**	**0,00**	**0,00**
FL6	**0,00**	**0,00**	**1,00**	**0,00**	**0,00**	**0,00**	**1,00**	**0,00**	**0,00**	**0,00**	**1,00**	**0,00**
FL8	**0,01**	**0,02**	**0,05**	**1,00**	**0,01**	**0,02**	**0,01**	**1,00**	**0,00**	**0,02**	**0,03**	**1,00**

	Panel 1 Hg				**Panel 2 Hg**				**Panel 3 Hg**			
	FL1	FL3	FL6	FL8	FL1	FL3	FL6	FL8	FL1	FL3	FL6	FL8
FL1	**1,00**	**0,38**	**0,07**	**0,01**	**1,00**	**0,22**	**0,42**	**0,02**	**1,00**	**0,35**	**0,51**	**0,00**
FL3	**0,01**	**1,00**	**0,00**	**0,00**	**0,03**	**1,00**	**0,00**	**0,01**	**0,05**	**1,00**	**0,00**	**0,00**
FL6	**0,00**	**0,00**	**1,00**	**0,00**	**0,00**	**0,00**	**1,00**	**0,00**	**0,00**	**0,00**	**1,00**	**0,00**
FL8	**0,01**	**0,02**	**0,05**	**1,00**	**0,01**	**0,02**	**0,06**	**1,00**	**0,00**	**0,02**	**0,04**	**1,00**

Tabla 6. Resultados del calculo de las matrices de compensación con el script "*fluo.compensation.R*". Los valores que difieren en más de un 2% para los diferentes tóxicos se muestras subrayados.

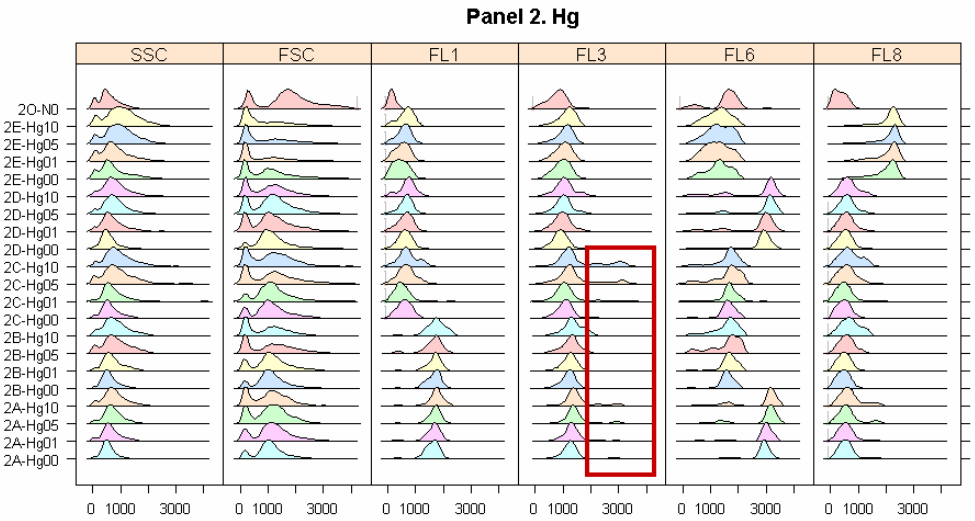

Figura 8. Gráficos de densidad normalizada de las poblaciones de panel 2 Hg.

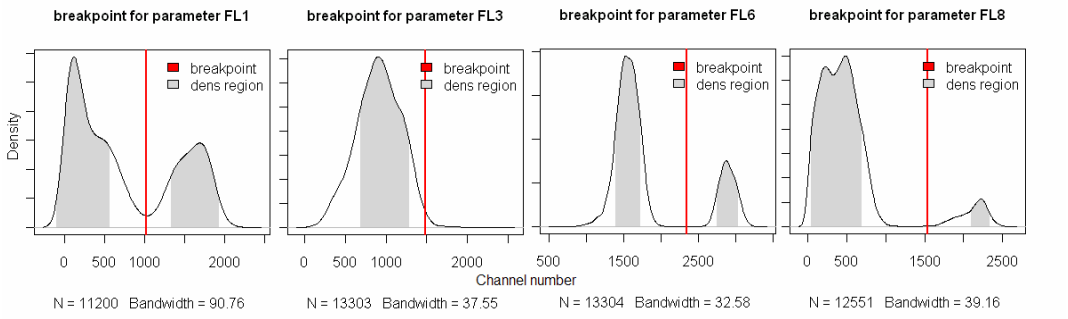

Figura 9. Selección de poblaciones positivas en panel 2 Hg. El script "*positive.selecction.R*" genera un punto de corte (linea roja) para las poblaciones positivas basándose en un análisis de cluster de densidad de las poblaciones colapsadas de los controles.

3.5.5 Restablecer la amplificación logarítmica del citómetro a los datos lineales

Para comprobar el resultado de la compensación hay que restaurar la amplificación logarítmica original. Esta operación presenta un problema técnico: Durante la compensación se han generado valores de fluorescencia negativos debidos al ajuste de las líneas base por lo que el cálculo del logaritmo producirá errores de cálculo y el apilamiento en el canal cero de las poblaciones con más baja intensidad (poblaciones negativas). Por ello es necesario reescalar los picos de las intensidades basales a los valores previos a la compensación. Este reescalado se realizó calculando el valor mínimo de la mediana de cada canal antes y después de la compensación. La diferencia entre ambos valores se consideró que era el valor de reescalado correcto. Por ejemplo, el trozo de código inferior muestra el cálculo para el Panel 3–Cd. A cada canal de la serie de experimento se le sumó la cantidad correspondiente

```
> medias.diff <- TMediansMinima.antes - TMediansMinima.despues
> medias.diff
       FL1        FL3        FL6        FL8
0.51308438 1.34915253 0.72858791 0.02654061
>
```

Los valores de intensidad reescalados de esta forma (O_{linC}) se convirtieron a la escala logarítmica mediante la ecuación (Ec. 9).

$$O_{log} = \text{int}[(\text{Log10}\,(O_{linC})\bullet\ E/Cn] \qquad \text{(Ec. 9)}$$

3.5.6 Comprobación del resultado de la compensación.

Para comprobar que el resultado de la compensación fuera correcto se representaron gráficamente las distribuciones normalizadas de los seis conjuntos de experimentos una vez compensados en la escala logarítmica original (cada conjunto fue compensado con su

correspondiente matriz). Se consideró que la compensación era correcta si visualmente las poblaciones negativas de cada marcador estaban alineadas, es decir, que en los paneles SS el nivel de fluorescencia en los canales no marcados era igual al del control un-stained. Los seis grupos de experimentos mostraron un alineamiento óptimo por lo que se consideró que las matrices de compensación calculadas eran correctas.

Como ejemplo del resultado de la compensación analizamos aquí los resultados del panel 3 Cd. La Figura 10 muestra cómo los cambios más apreciables se encuentran en aquellos canales que tienen mayor solapamiento, como es el caso del desdoblamiento de FL1 en FL3 y FL6. Las líneas verdes horizontales marcan aproximadamente la referencia en donde quedan alineadas las poblaciones básales. Las flechas rojas marcan la posición original de la población no compensada y se observa cómo están desplazadas a mayor intensidad por el solapamiento con FL1. Las flechas azules muestran las poblaciones compensadas y reescaladas. Se observa cómo los niveles de fluorescencia basal de las poblaciones se han restablecido a los valores originales y están alineados.

Tradicionalmente el resultado de la compensación se ha comprobado mediante scatter plots. El output de "*fluo.compensation.R"* incluye un conjunto gráficos de este tipo. Una forma de visualizar la compensación es observar si desaparece el "estiramiento" de la población. En el caso del panel 3, la Figura 11 muestra parte del output gráfico. Se observa el efecto de la corrección en el desdoblamiento FL1-FL3 tanto en los controles SS como en los FS.

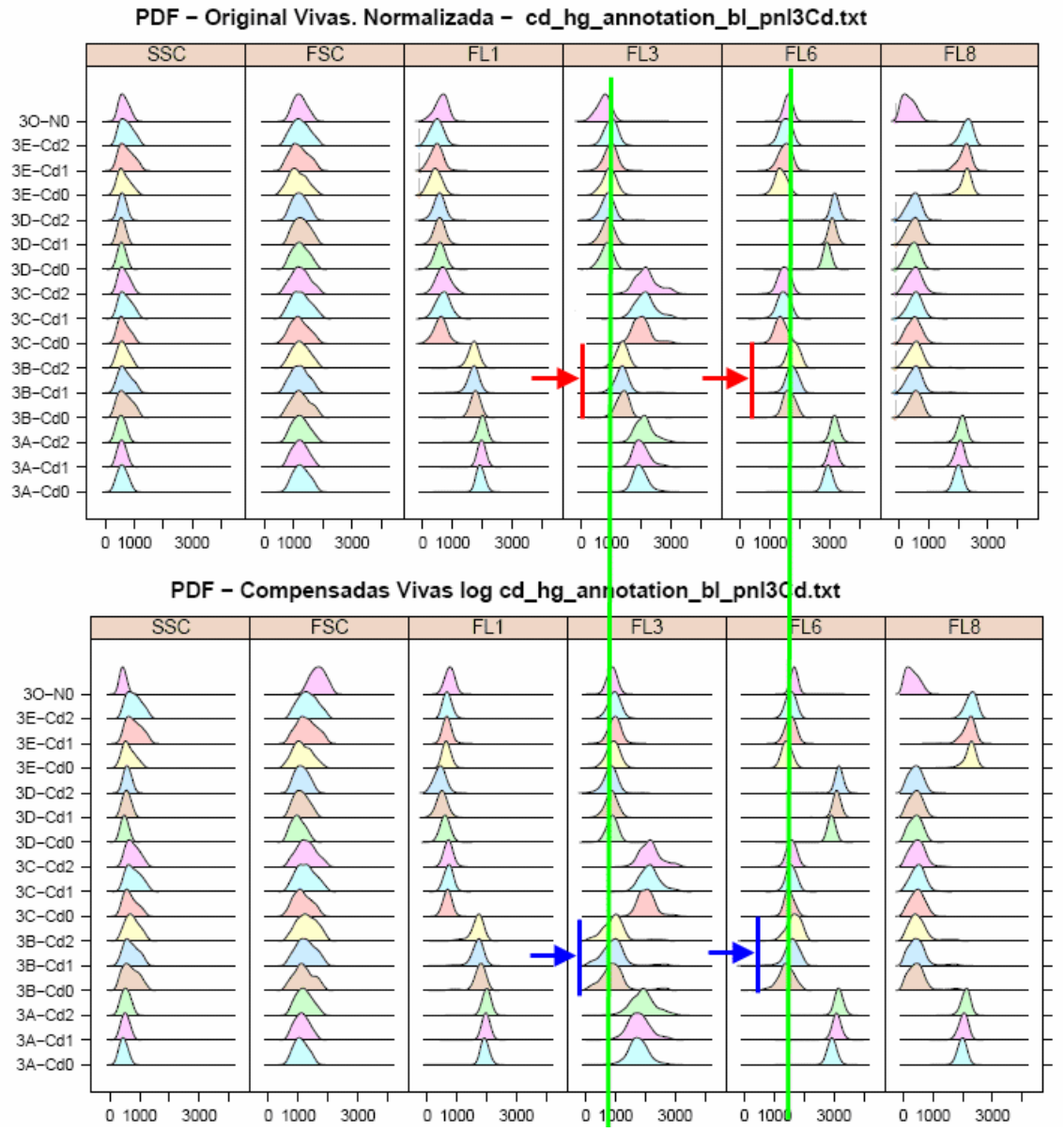

Figura 10 Gráficos de densidad normalizada de la selección de células viables del panel 3 Hg antes y después de la normalización. Las líneas verdes sirven como referencia de la mediana de la población. Las flechas rojas señalan la variación antes de la normalización. Las flechas azules señalan la corrección después de la compensación.

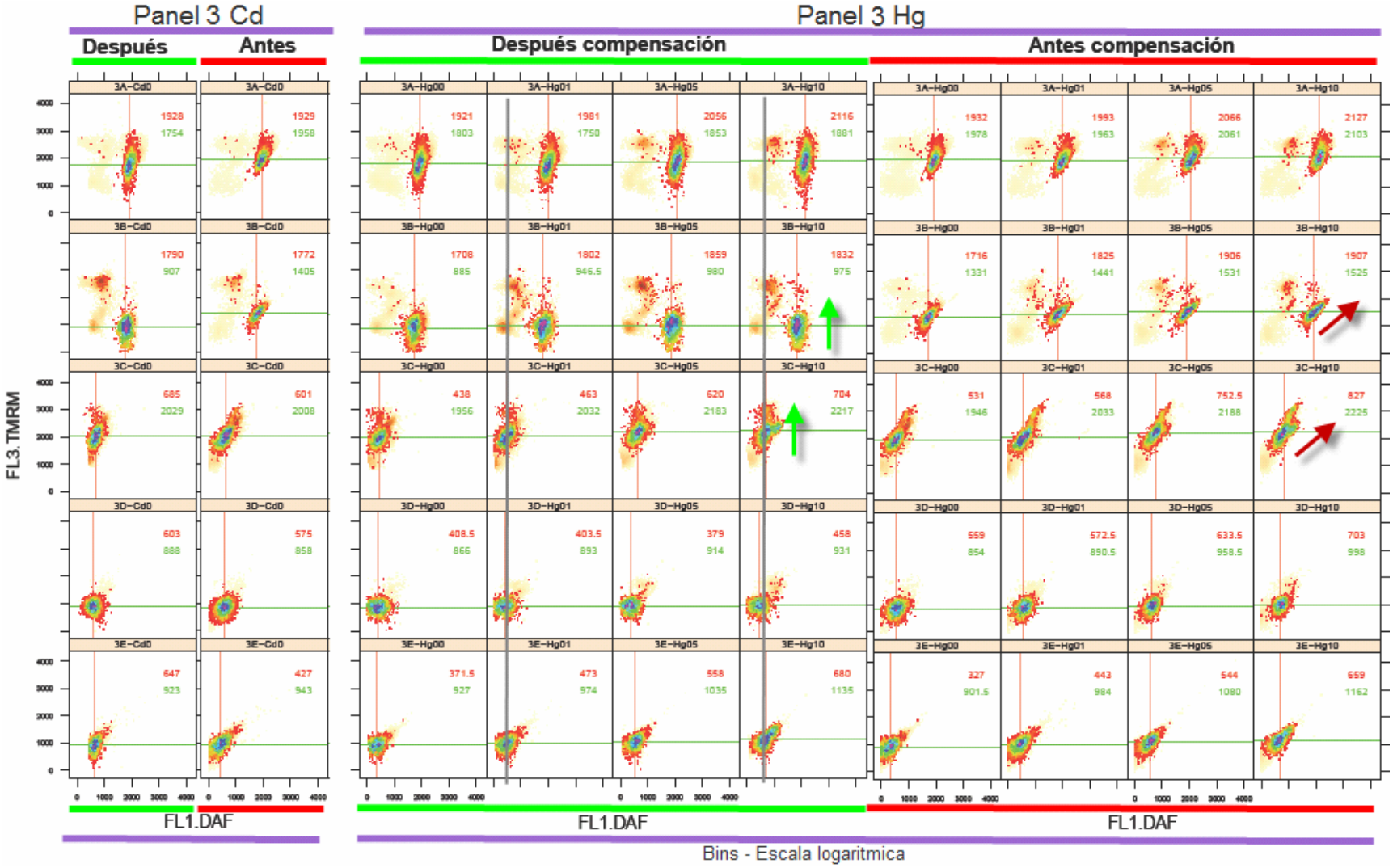

Figura 11. Resultado de la compensación en el panel 3.

3.5.7 Transformación Lin-Log y mejoras en la visualización de los flowSet.

La transformación y escalado es esencial tanto para la visualización como para el posterior tratamiento estadístico de los datos de citometría de flujo (Novo and Wood, 2008; Parks et al., 2006). Las transformaciones que se usan rutinariamente en el análisis de citometría de flujo se han implementado en *flowCore*. Independientemente, el diseño de R hace que sea fácil definir nuevas funciones arbitrarias aplicables tanto a los *flowFrame* como a los *flowSet* para incorporarlas en un protocolo habitual de *flowCore*.

Como se ha visto, la transformación logarítmica es el método comúnmente utilizado para hacer frente a la amplia gama dinámica de las medidas de fluorescencia en los citómetros. Sin embargo, la compensación de fluorescencia crea valores negativos que anteriormente se han resuelto restableciendo los niveles basales de fluorescencia. Otra alternativa es utilizar la transformación Lin-Log que es lineal en torno a cero y no lineal en otras regiones.

$$\hat{Y} = \begin{cases} M_{\text{linear}} \cdot (X_{\text{raw}} - b) & \text{if } X_{\text{raw}} < \text{transition} \\ \log_{10}(M_{\text{log}} \cdot (X_{\text{raw}} - b)) & \text{if } X_{\text{raw}} \geq \text{transition} \end{cases}$$

Aunque los valores de fluorescencia transformados con Lin-Log no tienen significado biológico directo, son útiles para generar visualizaciones no sesgadas por la transformación logarítmica habitual. En el caso que nos ocupa resulta interesante comparar visualmente la evolución conjunta de la intensidad basal y de la intensidad de fluorescencia mediante la transformación Lin-Log.

Para implementar esta funcionalidad, se han ampliado las funciones de visualización de *flowViz* con la función "*fnvisualizarScatervivas.FL*" incluida en "*funciones.propias.R"*. La función "*fnvisualizarScatervivas.FL*" se ha construido sobre la base de la funcion "*xyplot*" incluida en *flowViz* añadiendo funcionalidades importantes:

1. Se pasan como argumentos dos flowSets "*fcompTodas*" y "*fcompVivas*". El primero contiene los *flowFrames* sin filtro que se dibujan en un gradiente de color en base a la densidad de la población celular, mientas que el segundo es una selección de poblaciones del primero resultante de un filtro y se dibuja en azul para que destaque sobre el primero. De esta forma la población objeto de la selección queda destacada.
2. Como argumento se puede pasar la estructura (x vs. y) del scatter-plot de interés. Así, la función puede incluirse en un bloque que itere sobre todos los canales de fluorescencia del *flowSet* para obtener todas las combinaciones gráficas posibles.

3. La función calcula las medianas de la población de "*fcompVivas*" y las representa como líneas de diferente color. De esta forma se resalta en el gráfico la evolución de este parámetro con la concentración de tóxico.
4. Otros argumentos adicionales, heredados de la clase de origen como "*mult*" y "*my.layout*" facilitan la construcción de gráficos de múltiples paneles

```
#####################################################################
### chunk : Funcion fnvisualizarScatervivas.FL                    #
#####################################################################

fnvisualizarScatervivas.FL <- function(fcompTodas, fun= FL1 ~ FSC ,
                                       fcompVivas, main="Visualizacion",
                                       mult=1,my.layout=c(5,5)) {
x11(width=90, height=50)
print( xyplot( fun , fcompTodas,smooth=TRUE, pch = 21, cex = 0.3,
       main=main , layout=my.layout,
       panel = function(x, frames, channel.x, channel.y, ...) {
       nm <- as.character(x)
       x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
       y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
       my.panel(x, y, frame = frames[[nm]], ...)
       xyVivas <- exprs(fcompVivas[[nm ]])
       panel.abline(v= median(xyVivas[,x.intname]),col="red", alpha=0.5)
       panel.abline(h= median(xyVivas[,y.intname]),col="green", alpha=0.5)
       panel.points(xyVivas [,x.intname],  xyVivas [,y.intname] , col="blue",
                              alpha=0.3, pch=".", cex=1)
       intMedianV <- median(xyVivas[,x.intname])
       maxMedianV <- max(x)
       intMedianH <- median(xyVivas[,y.intname])
       maxMedianH <- max(y)
       panel.text(intMedianV ,maxMedianH*1.4 ,pos=1 ,
       format(intMedianV ,digits=3),col="red", cex=0.6)
       panel.text(maxMedianV ,intMedianH*1.1 ,pos=1 ,
       format(intMedianH ,digits=3),col="green", cex=0.6)
        }))
       }
```

Código 4 Función "*fnvisualizarScatervivas.FL*".

El bloque de código de Código 4 sirve como ejemplo de cómo modificar la funcionalidad de *flowViz* para adaptarla a los requerimientos del análisis. En la librería "*funciones.propias.R"* se incluyen otras variaciones de esta función que pueden ser útiles para resolver cualquier problema de visualización de datos de citometría de flujo.

La Figura 12, Figura 13 y Figura 14 son ejemplos de cómo la visualización de los scatter-plots en escala Lin-Log mediante la función "*fnvisualizarScatervivas.FL*" facilita interpretación de datos con alta complejidad.

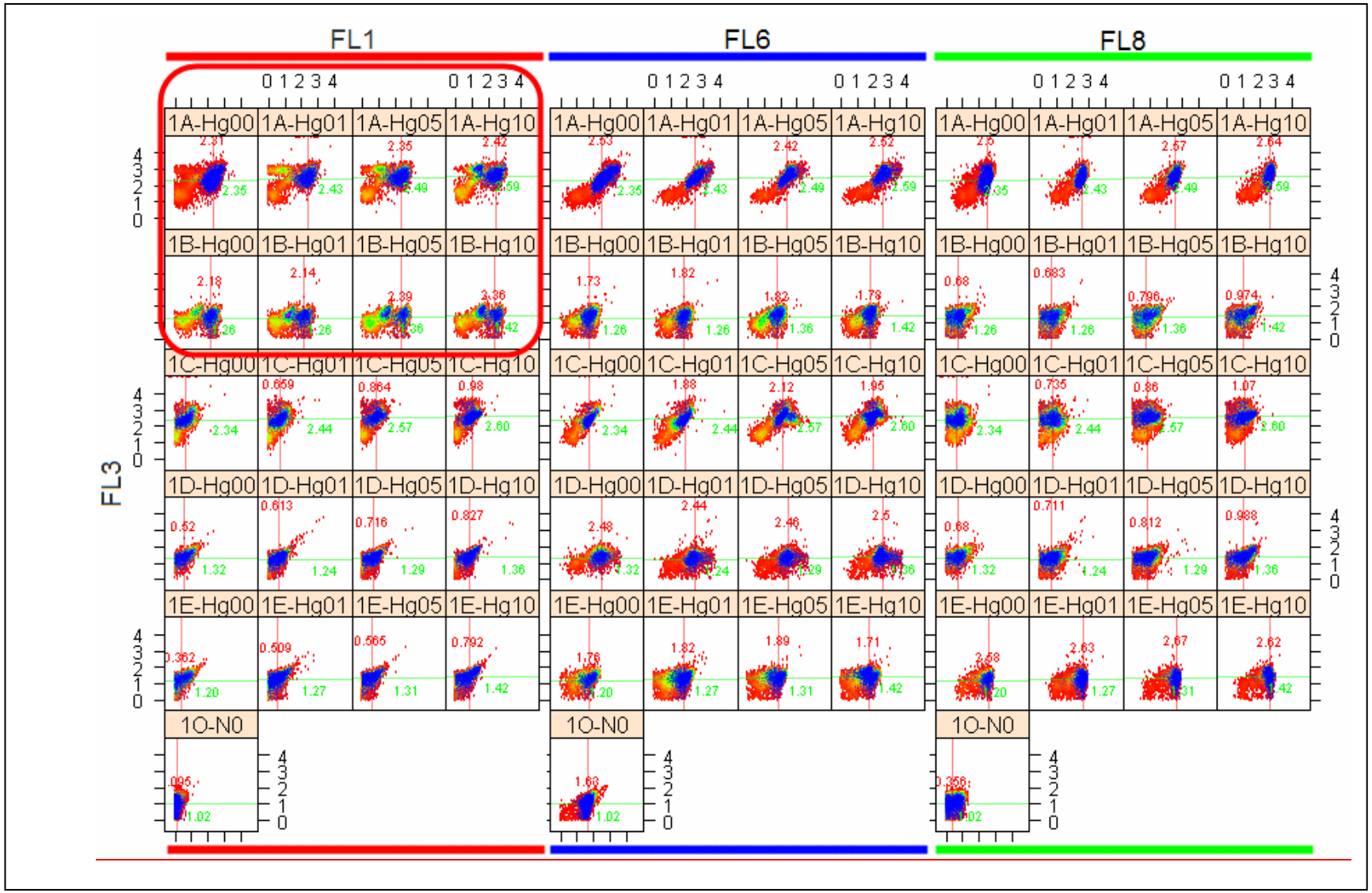

Figura 12. Visualización LinLog de la población viable (morphGateScale=2.0) resaltada en azul del panel 1 (Hg). El recuadro rojo muestra como FL1 se desdobla en dos poblaciones independientes con la adición de Hg. Se aprecia que los niveles de auto fluorescencia detectados en el canal FL3 no son iguales para las tres poblaciones.

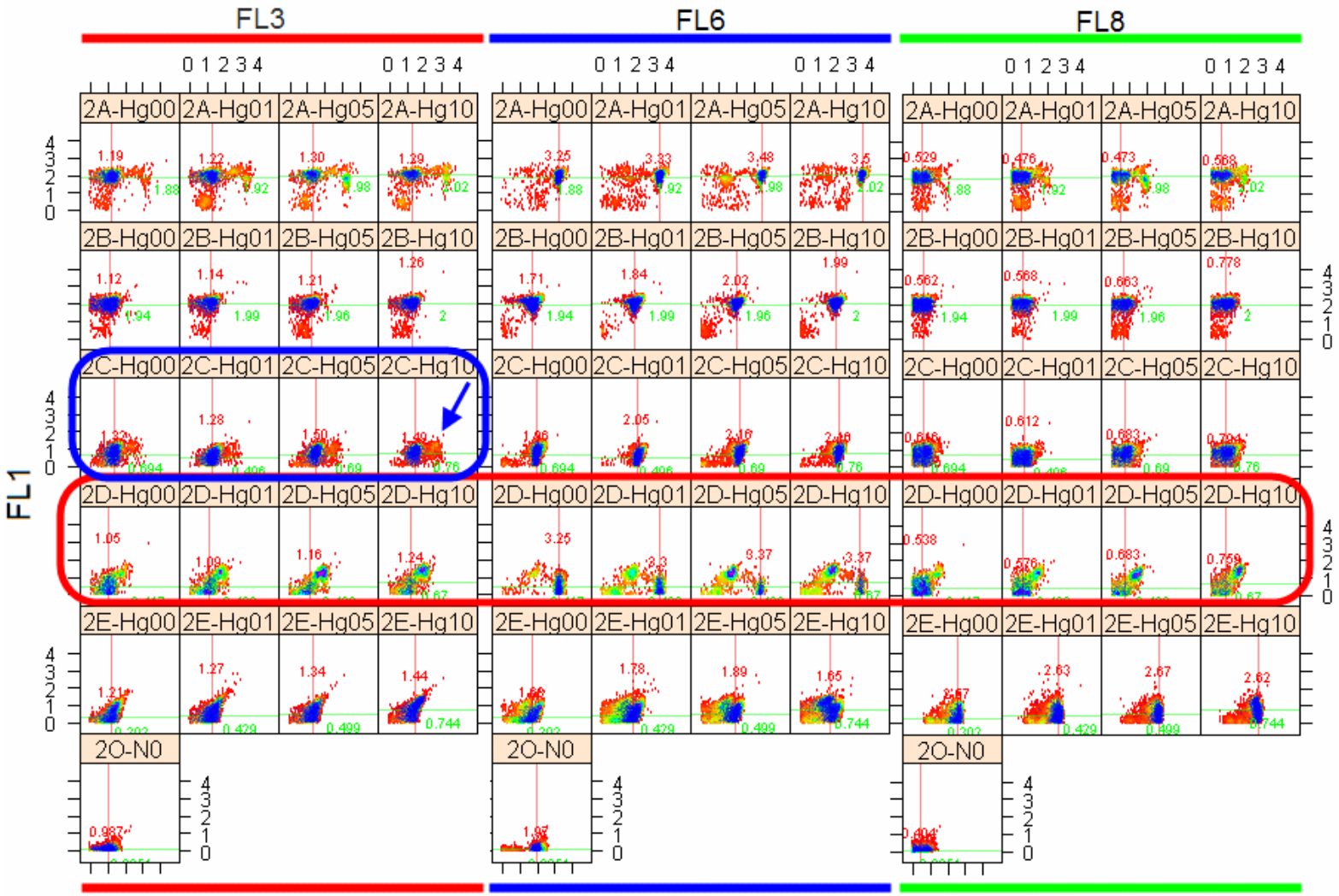

Figura 13. Visualización LinLog de la población viable (morphGateScale=2.0) resaltada en azul del panel 2 (Hg). La flecha azul muestra la posición de la población positiva de FL3.

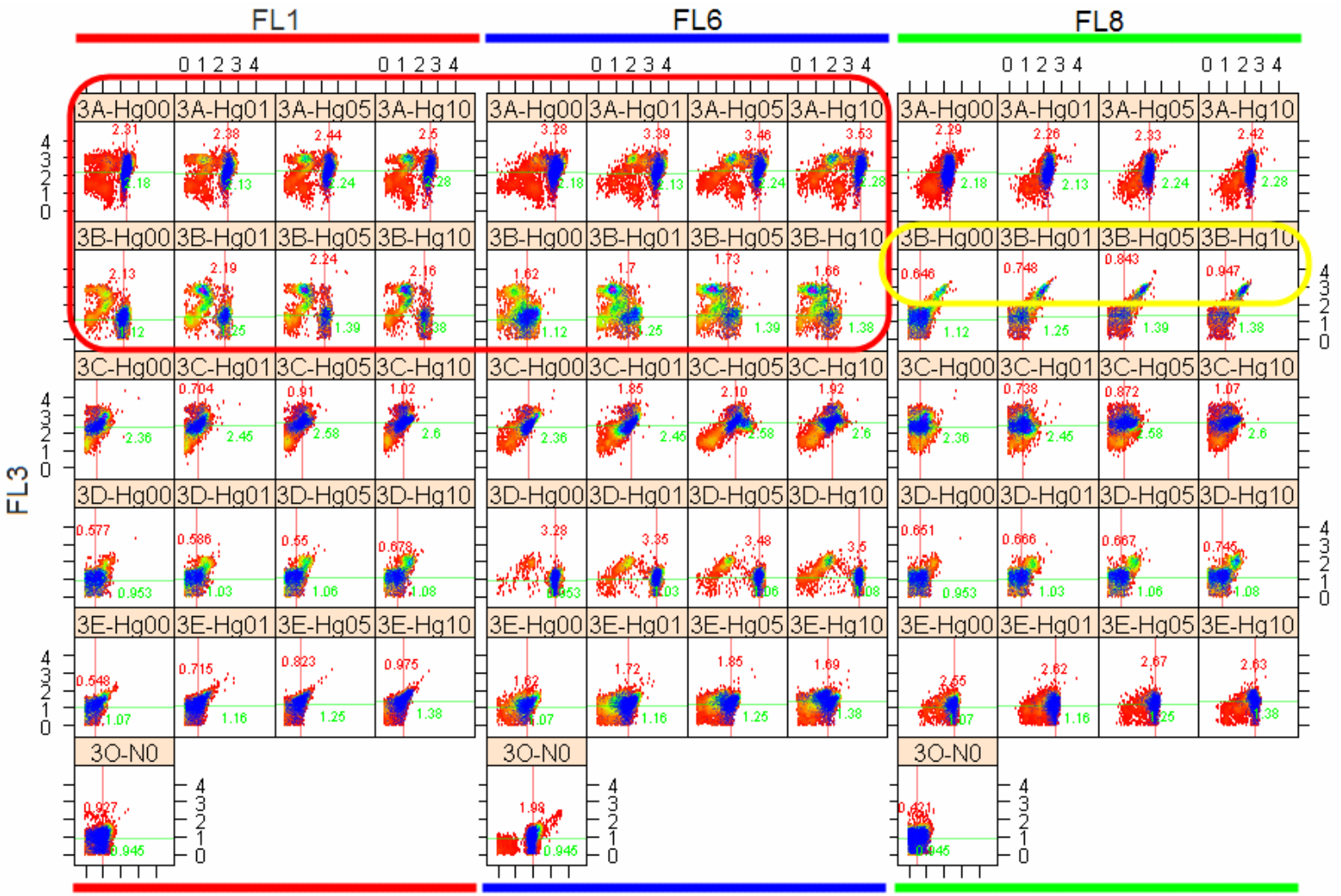

Figura 14. Visualización LinLog de la población viable (morphGateScale=2.0) resaltada en azul del panel 3

3.6 Evaluación del efecto del Cd y Hg en la intensidad de fluorescencia

3.6.8 Método de cálculo e implementación

Mediante el script "*fluo.compensation.R"*se obtienen las estadísticas habituales de citometría de flujo como son: los porcentajes de células viables seleccionadas (Figura 1S en el anexo) y las medias y medianas de las poblaciones resultantes (no se adjuntan lo resultados). Además, los *flowFrames* compensados en los apartados anteriores, fueron almacenados en disco para su posterior procesado. De esta forma se evita repetir el cálculo de la compensación al añadir al script nuevas etapas de análisis. Los *flowframes* se almacenaron en el formato FCS original y pueden ser procesados con cualquier software compatible.

El cálculo de la mediana de la población viable seleccionada con los filtros descritos anteriormente es una buena estimación de la variación de los marcadores de viabilidad con la concentración de toxico. Como se ha comprobado, algunos marcadores de viabilidad responden de forma ON/OFF con la adición de tóxico, generando dos poblaciones y la media o mediana de la población no marcará exactamente la variación real de la población original (Hahne et al., 2006).

Para realizar un análisis más detallado del efecto de la concentración de tóxico sobre la población viable se ha implementado el script "*analysis.compensated.R*". La construcción del script es una muestra de la potencia de R y *flowCore* para manejar un conjunto extenso de experimentos agrupados por paneles y factores. El script lee y linealiza el conjunto completo de ficheros FCS y los trata mediante técnicas de programación estructurada (listas, iteraciones, etc.) para obtener un fichero de resultados en forma de matrices. Los valores de las matrices resultantes se grafican como parte del output.

El objetivo del script "*analysis.compensated.R*" es obtener las medianas en escala lineal de las poblaciones negativas y positivas de la población viable en los experimentos compensados (aplicando los filtros descritos en secciones anteriores). Las poblaciones positivas y negativas de cada marcador se estiman con la función "*rangeGate()*" de *flowStats*. Esta función recibe como entrada un flowSet, la población de cada canal en cada *flowFrame* se colapsa en una misma población y en la población resultante se busca el punto de corte óptimo que separará las poblaciones positivas y negativas (ver detalles de la implementación en la documentación de *flowStats*). La Figura 9 muestra el resultado de la función sobre los controles del panel 2 Hg. La línea roja representa el punto de corte. Se observa que para FL3 sólo existe una población, en ese caso la función retorna una estimación basada en 95% de la población.

Sobre la población de células viables positivas y negativas de cada muestra se calculan las siguientes estadísticas:

1. Porcentajes de células viables positivas para cada marcador en cada panel (Figura S1)
2. Mediana de la población positiva y negativa de cada muestra.
3. Ratio de variación de la intensidad de fluorescencia intrínseca (en escala lineal) de la población positiva de cada marcador respecto de su control (en adelante lo denominaremos RVIF) (Smith et al., 2006).

El cálculo del RVIF es especialmente importante ya que es proporcional a la variación del número de moléculas de fluorocromo activadas entre el control (sin toxico) y el tratamiento.

$$O_{linC} = C \cdot f \cdot Nc + C \cdot A \; ; \; O_{linT} = C \cdot f \cdot Nt + C \cdot A \qquad \text{(Ecs. 9)}$$

Según la ecuación (Ec. 7) y suponiendo que la auto-fluorescencia (término $C \cdot A$ en la Ec. 9) se puede estimar como la mediana de la población negativa del control y no varia con el tratamiento, el ratio RVIF queda expresado por la ecuación (Ec. 11), en porcentaje en la ecuación (Ec. 12). La implementación en R del método de cálculo se muestra en el Código 5.

$$\text{RVIF} = (O_{linT} - C \cdot A) / (O_{linC} - C \cdot A) \qquad \text{(Ec. 10)}$$

$$\text{RVIF} = \text{Nt/Nc} \qquad \text{(Ec. 11)}$$

$$\%\text{RVIF} = 100 \text{ x } ((\text{Nt/Nc}) - 1) \qquad \text{(Ec. 12)}$$

```
### Calcular el ratio de variación respecto del control
controles.pos <- MediansData.pos.lin [ which (MediansData.pos.lin ["CTox"]==0), ]
controles.neg <- MediansData.neg.lin [ which (MediansData.neg.lin ["CTox"]==0), ]
Expers <- MediansData.pos.lin [ which (MediansData.pos.lin["CTox"]!=0), ]
ContrId <- Expers[,"name"]
Expers <- cbind(Expers, ContrId )
porcVar <- cbind(Expers, ContrId )
for (i in 1:nrow(Expers)){
       st <- Expers [i,"StainId"]
       tx <- Expers [i,"Tox"]
       pn <- Expers [i,"panel"]
       isControl <-  which (controles.pos["StainId"]==st  &
                            controles.pos["Tox"]==tx &
                            controles.pos["panel"]==pn )
       isBaseLine <- which (controles.neg["StainId"]==st  &
                            controles.neg["Tox"]==tx &
                            controles.neg["panel"]==pn )
       control <- controles.pos[ isControl  , ]
       baseLine <-  controles.neg[ isBaseLine  , ]

       Expers [i,"FL1"] <- round((Expers [i,"FL1"] - baseLine[1,"FL1"])/
                                 (control[1,"FL1"] - baseLine[1,"FL1"]),2)
       #### …. El código es similar para FL3, Fl6 y F8 … se omite
       porcVar [i,"FL1"] <- round(((porcVar [i,"FL1"] - baseLine[1,"FL1"])/
                            (control[1,"FL1"]-baseLine[1,"FL1"])-1)*100,0)
       #### …. El código es similar para FL3, Fl6 y F8 … se omite
                        }
```

Código 5. Implementación en R del cálculo del ratio RVFI y su porcentaje.

3.6.9 Resultados de los porcentajes de células viables positivas.

La Figura 15 muestra el porcentaje de células viables positivas para cada marcador. Se detectan tres casos que se comentan a continuación:

Como ya se ha comentado, el Mitosox, marcador del superóxido mitocondrial (FL3 - Panel 2) muestra un efecto ON/OFF. La muestra control tiene una población positiva muy débil que aumenta con el tratamiento con Cd^{2+} y Hg^{2+}. Se observa que a igual tratamiento la población de células positivas viables es siempre mayor en las muestras SS que en las muestras FS.

La población de viables positivas de Indo-1, marcador de calcio intracelular (FL6 – Panel 1) aun siendo en las muestras de control superior al 80% aumenta levemente hasta el 95% con la adición de Cd^{2+} o Hg^{2+}. Por el contrario el marcador DAF de oxido nítrico (FL1 – panel 3) tiene una tendencia contraria al Indo-1.

El resto de marcadores muestran un porcentaje elevado (superior al 85%) de viables positivas sin tendencias apreciables con la concentración de tóxico.

3.6.10 Resultados de las medianas de las poblaciones viables positivas y negativas.

La Figura 16 muestra los resultados del cálculo de las medianas de las poblaciones positivas y negativas. En la Tabla 7 se incluyen los valores calculados del porcentaje de RVFI.

Las flechas naranjas de la Figura 16 señalan las variaciones más apreciables en la intensidad de autofluorescencia. Los marcadores que muestran estas variaciones son el Mitosox (FL3 – panel 2) y MCB (FL6 panel 2 y 3). El hecho de que las tendencias en MCB se presenten en las muestras SS (muestras C) como en las FS sugiere que el efecto no es debido a artefactos del método de cálculo.

Las flechas verdes de la Figura 16 indican las tendencias reproducibles entre muestras SS y FS. En concreto, los marcadores: TMRM (FL3 panel 1 y 3), Indo-1 (FL6 panel 1), Mitosox (FL3 panel 2) MCB (FL6 panel 2 y 3), tienen tendencias reproducibles tanto en los experimentos con Cd como con Hg. El marcador Bodipy 675 (FL8 panel 1 y 3) muestra tendencias similares en el Hg^{2+}, pero no para el Cd (flechas azules). No obstante las variaciones parecen reproducibles entre los paneles 1 y 3, salvando las diferencias entre los niveles de intensidad en los experimentos full-stain con Cd^{2+}. Las flechas rojas indican tendencias contrapuestas o aparentemente no correlacionadas entre experimentos FS y SS.

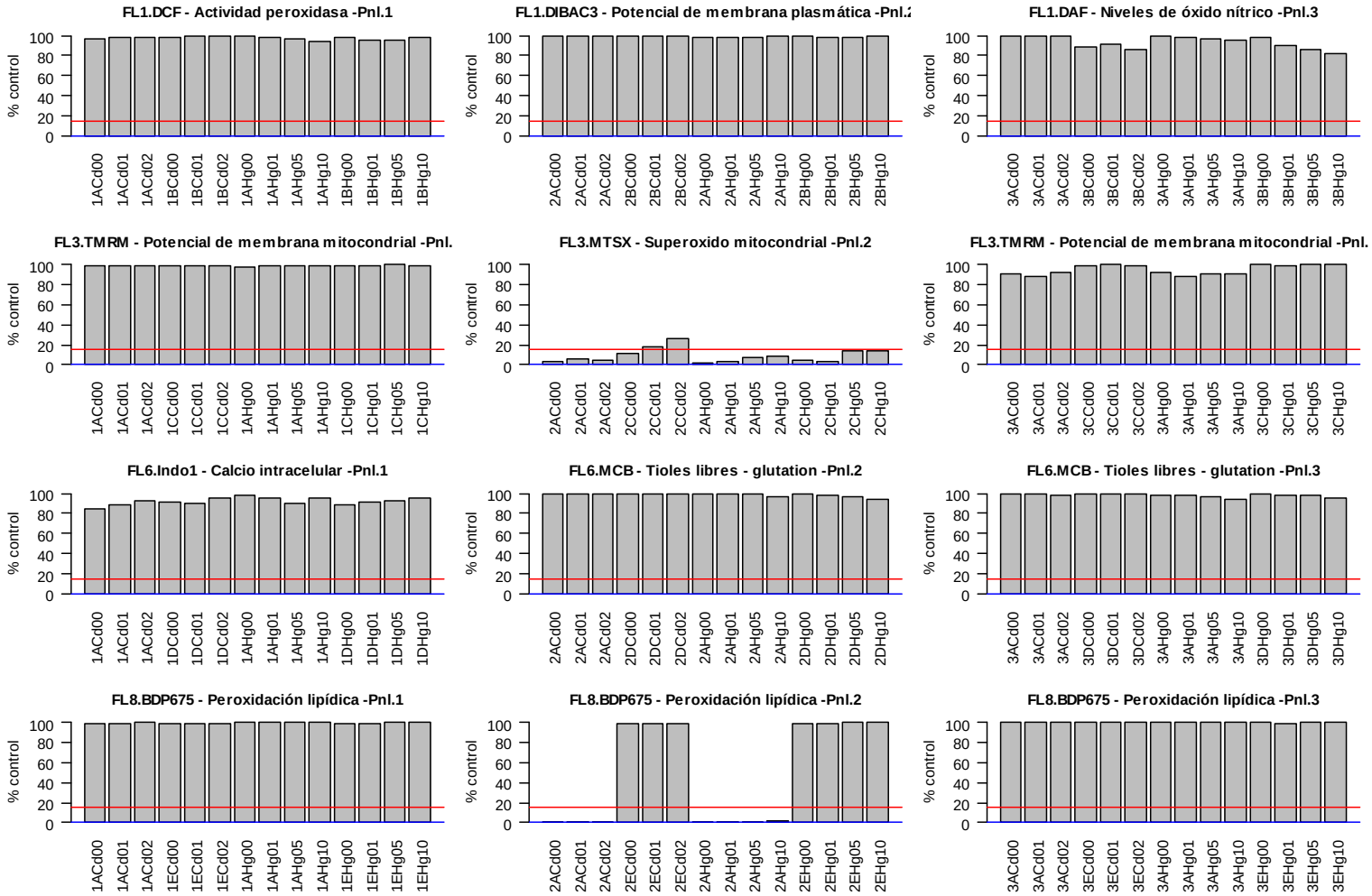

Figura 15. Porcentaje de células positivas para el marcador especificado en la selección de células viables. (Por error las muestras A de panel 2 no fueron marcadas con Bodipy 675)

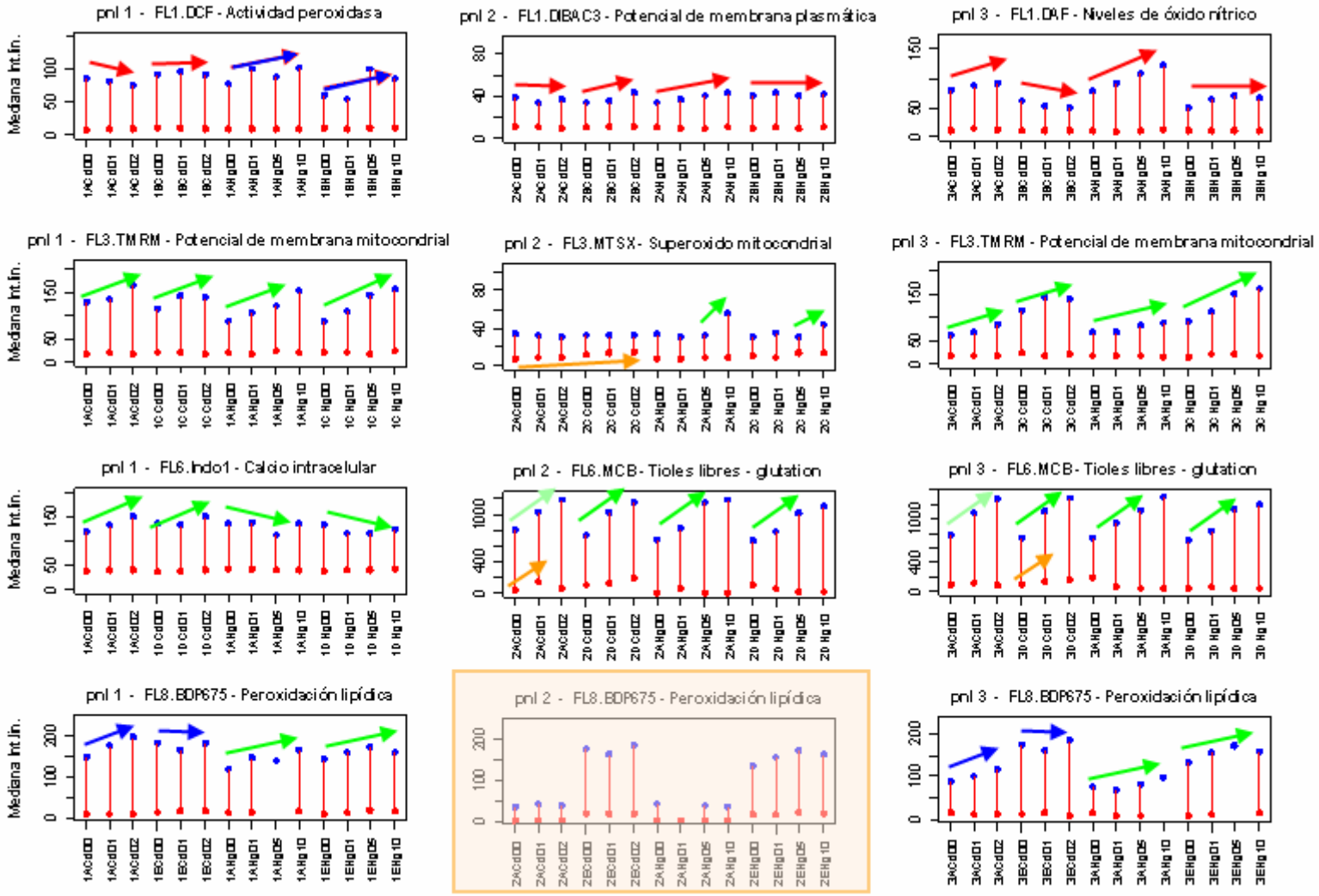

Figura 16. Medianas de las poblaciones positivas (círculos azules) y negativas (círculos rojos) de la selección de la población viable. La ausencia de círculo rojo indica que no se ha detectado población negativa.

3.6.11 Resultados de los valores de RVFL

La Figura 17 muestra los valores calculados del porcentaje de RVFI. La línea roja en los gráficos indica el 15% (ó -15% de variación).

En el panel 1, se observa que no hay correlación entre los valores obtenidos con los experimentos SS y FS. En el panel 3 las tendencias de los FS y SS para el Hg^{2+}son reproducibles, pero las del Cd^{2+} no lo son.

En el panel 2, si que es posible una cuantificación conjunta de Mitosox y MCB (por error no se añadió Bodypy 675 a los experimentos FS). La elevada intensidad de fluorescencia de MCB (FL6, 800-1200), sobre Dibac3 (FL1, 35-40) resulta en una aparente falta de compensación en el canal FL1.

3.6.12 Comparación del efecto de los controles positivos con los marcadores.

Se dispone de un conjunto de experimentos con controles positivos para algunos de los marcadores funcionales y se desea comparar el efecto de estos sobre los marcadores. El script "*Analisis.positive.ctrl.R*" toma como entrada un único fichero de anotación (Tabla S1 en el Anexo) y las matrices de compensación para cada panel calculadas anteriormente.

El script compensa los experimentos FS y los experimentos SS se mantienen sin compensar. Posteriormente realiza una selección de la población viable con los filtros empleados anteriormente y se calculan las medianas de las poblaciones resultantes. Para comparar el efecto de cada toxico, se obtiene el ratio entre las medianas de la intensidad del tratamiento e intensidad del control.

Las Tablas S2 del anexo contienen los resultados de los ratios. Un ratio mayor de 1 significa aumento de la fluorescencia y menor que 1 disminución. Los valores significativos se marcan en rojo para la disminución y en azul para el aumento. Los valores resaltados en amarillo corresponden a los controles en los que se conoce la actividad sobre ese parámetro. En los controles positivos no es posible calcular el porcentaje RVFI ya que las intensidades de autofluorescencia son visiblemente diferentes (no se muestran los datos). En las Tablas S2 se incluyen los ratios de los canales no marcados sin compensar para evaluar el efecto de la no compensación sobre el ratio.

Los valores obtenidos están dentro del margen de los obtenidos para el Cd^{2+} y el Hg^{2+} y por tanto se pueden usar como referencia para posteriores estudios. La diferencia entre los ratios de las muestras SS y FS en los controles positivos es un indicativo de la precisión de la compensación. Los valores más significativos se resumen en la Tabla 7. Se aprecia que los

controles CHP, FCCPP, IONOM, BSO tienen un ratio reproducible y por tanto la compensación es correcta. Los controles GRAM (FL1 – panel 2) y NOR-1 (FL1 – panel 3) parecen indicar que existen problemas de compensación o interferencia.

Esta apreciación está de acuerdo con lo observado en los valores de las matrices de compensación. En el panel 2, el desdoblamiento de FL6 (MCB) en FL1 es de 0.36 (Cd) y 0.42(Hg), mientras que el calculado para el panel 3 es de 0.53 (Cd) y 0.51 (Hg). Teóricamente, el coeficiente debería ser similar en los cuatro casos. El parámetro empleado en la compensación FL6 – FL1 en los controles positivos fue de 0.53. Para comprobar si este era el valor correcto, se realizaron compensaciones con un barrido de valores entre 0.3-0.6 (no se muestran los datos) y en ningún caso se obtuvieron ratios reproducibles ni en los controles positivos ni en los tratamientos con tóxicos.

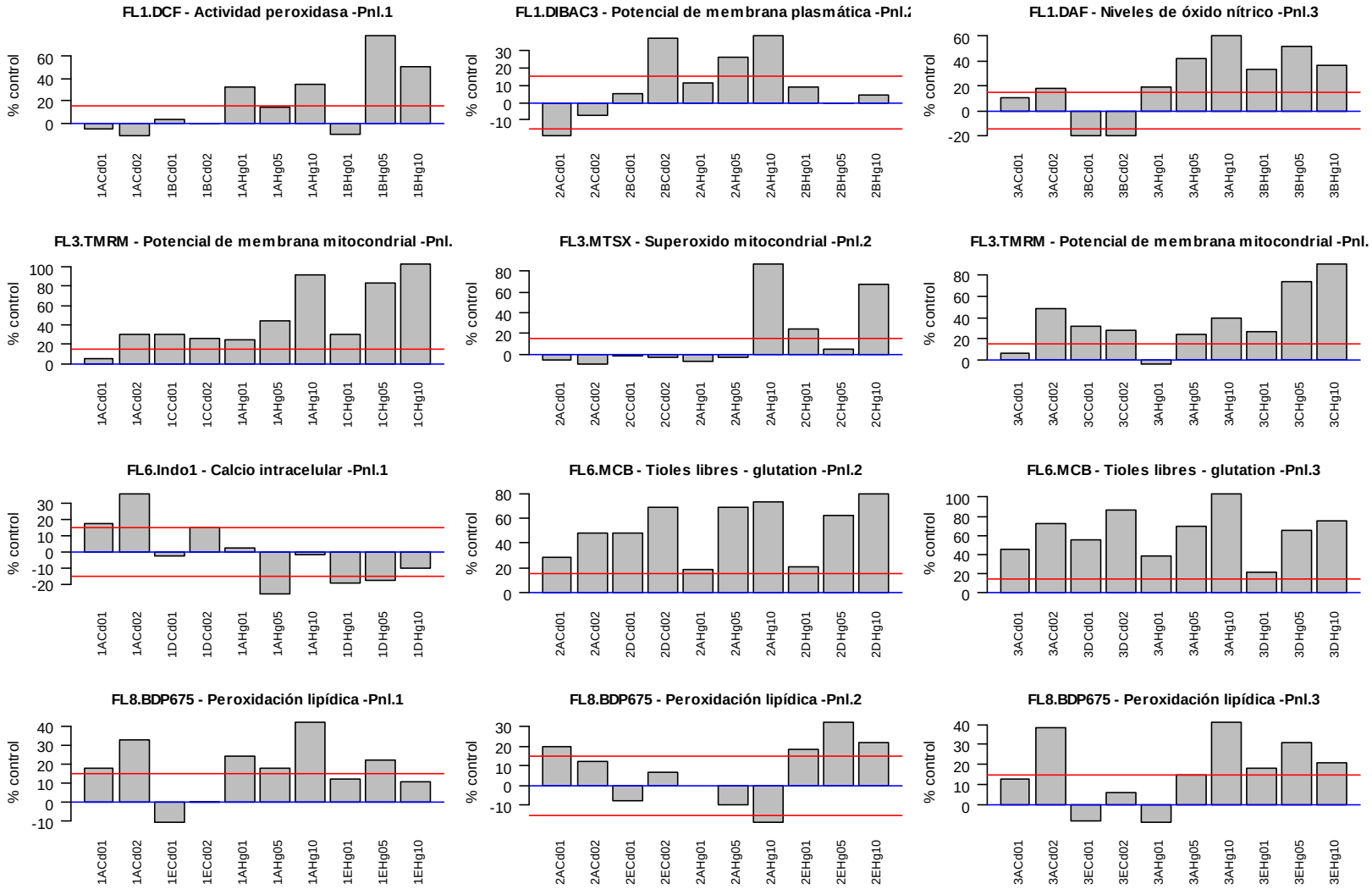

Figura 17. Porcentaje de variación de la intensidad de fluorescencia intrínseca del marcador respecto del control (RVFI). Los valores fueron calculados mediante la ecuación (Ec. 12). La línea roja señala el 15% y el -15% si procede.

			%RVFL		It/Ic			
Marcador	**Actividad**	**Panel**	**Cd (2µM)**	**Hg (37 µM)**	**Cd (2 µM)**	**Hg (37 µM)**	**Control positivo**	
DCF	Actividad peroxidasa	1	0	51	1.21	1.94	2.71 (2.25)	CHP
TMRM	Potencial de membrana mitocondrial	1	28	104	1.36	1.85	0.53 (0.52)	FCCPP
		3	28	90	1.34	1.86		
Indo-1	Calcio intracelular.	1	15	-10	1.16	1.10	1.88 (1.93)	IONOM
BODIPY 675	Peroxidación lipídica	1	0	11	1.05	1.22		
		2	7	22				
		3	6	21				
DIBAC3(4)	Potencial de membrana plasmática.	2	37	4	1.39	1.06	1.77 (3.07)	GRAM (10 µg/ml)
MCB	Niveles de tioles libres (glutation)	2	69	80	1.79	1.76	0.47 (0.54)	BSO
		3	86	75				
DAF	Niveles de óxido nítrico	3	-20	37	0.87	1.56	4.88(2.71)	NOR-1
Mitosox	Superoxido mitocondrial	2	-3	66	1.40	1.45	(1.34)	PARAQ (200 µM)

Tabla 7. Valores de %RVFL y del ratio entre las medianas de la intensidad del tratamiento e intensidad del control (columna It/Ic). Entre paréntesis se muestra el valor de los experimentos FS.

3.7 Valor biológico de los resultados

Los efectos de la exposición aguda a metales pesados ha sido recientemente revisada (Bashashati and Brinkman, 2009). El Cd^{2+} y Hg^{2+} son conocidos por su elevada toxicidad y han sido utilizados como venenos. A dosis subletales los cationes metálicos son carcinógenos, causantes de estrés oxidativo y apoptosis (Crespo-Lopez et al., 2009; Cuypers et al., 2010; Valko et al., 2005). La bibliografía acerca del efecto sobre parámetros funcionales en dosis bajas es amplia y controvertida debido a la diferencias metabólicas entre las líneas celulares en las que se realizan los estudios (Cuypers et al., 2010). La Tabla 8 muestra una relación de artículos sobre la actividad del Cd^{2+} y Hg^{2+} sobre los parámetros funcionales estudiados en este trabajo. Se han seleccionado aquellos artículos con dosis similares a las empleadas en los ficheros analizados.

La citotoxicidad de los cationes metálicos es debida a su afinidad por los grupos tioles. Una de las diferencias importantes de la toxicidad del Cd^{2+} y Hg^{2+} con otros cationes metálicos es que alteran el metabolismo del glutation de forma que se aumenta su síntesis (Hultberg et al., 1998; Hultberg et al., 1997). Los resultados obtenidos en las tendencias de la fluorescencia del MCB en los experimentos SS y FS confirman esta observación.

Un estudio de toxicidad en *Medicago Sativa* (Alfalfa) confirma la disminución del oxido nítrico con Cd^{2+}. Sin embargo, no se han encontrado referencias bibliográficas que expliquen el aumento y posterior disminución del oxido nitrico con la concentración de Hg^{2+}. (de Marco et al., 2010; Majumder et al., 2009; Majumder et al., 2008). Majumder et al. sugieren que el MCB puede interferir la medida de los indicadores de estrés oxidativo, como el óxido nítrico y el superóxido. El MCB al interaccionar con el glutation lo inactiva con el consiguiente aumento de los niveles de oxido nítrico y alteración de su medida. De esta forma se puede explicar la tendencia contrapuesta obtenida con DAF (FL1 panel 3), ya que cuando la sonda DAF combinada con MCB muestra un incremento de los niveles de óxido nítrico, mientras que en ausencia de MCB la respuesta celular detectada es una disminución.

Los estudios de actividad peroxidasa en Cd^{2+} muestran que no hay efecto o una disminución de la actividad peroxidasa, de acuerdo con lo observado en este trabajo. En el caso del Hg^{2+}, a diferencia del Cd^{2+}, los estudios recientes (Lopez et al., 2006; Majumder et al., 2009; Reddy et al., 1981) sugieren que se inhibe la actividad peroxidasa y estudios anteriores que se incrementa (Black et al., 1979; Jamall and Smith, 1985) .Los valores obtenidos de la actividad peroxidasa para el mercurio oscilan entre diferentes tratamientos, lo que sugiere que las medidas de este parámetro pueden estar afectadas por la cinética de la reacción entre el marcador y su

molecula diana. La peroxidación lipídica en ambos casos aumenta de acuerdo con lo observado en la bibliografía.

Las variaciones observadas en el potencial de membrana mitocondrial no están de acuerdo con lo observado generalmente en la bibliografía (InSug et al., 1997; Lopez et al., 2006). Sin embargo, un estudio reciente de (Nesovic-Ostojic et al., 2008) con concentraciones micromolares de Cd^{2+}y Hg^{2+} muestran hiperpolarización mitocondrial y plasmática, además de un aumento del Ca^{2+} intracelular de acuerdo con lo observado en los cálculos de este trabajo.

Un estudio de (Belyaeva et al., 2008; Belyaeva et al., 2006) muestra el impacto de diferentes concentraciones de Cd^{2+} y Hg^{2+} sobre los potenciales de membrana plasmática y mitocondrial en una línea celular de hepatoma de rata. Los resultados indican que con concentraciones del orden de 10–500 μM el efecto sobre el potencial de membrana mitocondrial es dependiente del tiempo de incubación y de la concentración. Con tiempos de incubación cortos (<24 h) y bajas concentraciones se observa un aumento en los radicales libres, mientras que con tiempos de incubación mayores de 48 horas se observa una disminución. El tiempo de incubación de las muestras analizadas en este estudio fue de aproximadamente un día. El limite de toxicidad en humanos es de 0.2-0.4 μM (Hultberg et al., 1998; Ortega-Villasante et al., 2007), inferior al empleado en este estudio (5-50 μM). Por tanto, es posible que las tendencias observadas sean debidas a la cinética de la toxicidad y a la cercanía de la concentración empleada al límite de toxicidad.

Función celular	Cd		Hg		Referencias
	Respuesta obtenida	Respuesta conocida	Respuesta obtenida	Respuesta conocida	
Actividad peroxidasa	⇔	⇧⇩*	⇧	⇧⇩*	(Black et al., 1979; Jamall and Smith, 1985; Lopez et al., 2006; Reddy et al., 1981)
Potencial de membrana mitocondrial	⇧	⇩⇧*	⇧	⇩⇧*	(InSug et al., 1997; Lopez et al., 2006; Nesovic-Ostojic et al., 2008)
Calcio intracelular.	⇧	⇧	⇩	⇩	(Belyaeva et al., 2008; Belyaeva et al., 2006; Nesovic-Ostojic et al., 2008)
Peroxidación lipídica	⇔	⇧	⇧	⇧	(Black et al., 1979; Jamall and Smith, 1985; Lopez et al., 2006; Reddy et al., 1981)
Potencial de membrana plasmática.	⇧	⇩⇧*	⇔	⇩⇧*	(Belyaeva et al., 2008; Belyaeva et al., 2006; Nesovic-Ostojic et al., 2008)
Niveles de tioles libres (glutation)	⇧	⇧	⇧	⇧	(Hultberg et al., 1998; Hultberg et al., 1997; Ortega-Villasante et al., 2005; Woods and Ellis, 1995)
Niveles de óxido nítrico	⇩	⇩	⇧	⇧	(de Marco et al., 2010; Majumder et al., 2009; Majumder et al., 2008)
Superoxido mitocondrial	⇔	⇧	⇧	⇧	(Cuypers et al., 2010; Ortega-Villasante et al., 2007; Ortega-Villasante et al., 2005)

* Alta dependencia de la línea celular, del tiempo de incubación, del medio y de la concentración de tóxico.

Tabla 8. Comparación de la variación de los marcadores funcionales con los datos existentes en la bibliografía.

4 Conclusiones

4.1 Sobre el uso de R-bioconductor para el tratamiento de datos.

1. Emplear R-bioconductor y las librerías tiene sentido cuando es necesario tratar un conjunto elevado de experimentos (50 o más) de forma iterativa.
2. Se han desarrollado librerías y funcionalidades adicionales a *flowCore* que permiten analizar conjuntos de experimentos con un número indefinido de paneles y combinaciones de tóxicos.
3. Se ha resuelto satisfactoriamente el problema estadístico que supone la amplificación logarítmica que realizan los citómetros.
4. Las funcionas gráficas desarrolladas permiten una visualización de los parámetros de evolución con el tratamiento hasta ahora no disponibles en la librería estándar *flowViz.*
5. Tanto las librerías como las funciones desarrolladas se han aplicado con éxito en un ejemplo real, permitiendo obtener resultados con valor biológico.

4.2 Futuras implementaciones o mejoras a los scripts

1. Las librerías podrían encapsularse en una interface gráfica. De esta forma se facilitaría su uso por investigadores no acostumbrados a trabajar con lenguajes de programación.
2. Se tiene que resolver el excesivo tamaño de los ficheros gráficos en formato pdf. De momento una salida gráfica completa de "*fluo.compensation.R*" supone alrededor de 300 Mb de memoria en disco.
3. Mediante el uso de microesferas calibradas (Gratama et al., 1998; Shackney et al., 2006) sería posible mejorar la compensación de fluorescencia en los parámetros que tiene baja intensidad (Por ejemplo FL1 en los paneles 1, 2 y 3).

4.3 Consideraciones sobre el problema biológico abordado en este trabajo

1. El estudio del mecanismo de toxicidad a dosis subletales por metales pesados es extremadamente complejo y requiere de mayor experimentación. En concreto habría que:
    - ampliar el número de dosis, el rango de las mismas, y asegurar la repetibilidad y reproducibilidad.
    - realizar estudios de la cinética de la toxicidad.

- estudiar el efecto de los tóxicos sobre la autofluorescencia celular.

2. La problemática de la medida multiparamétrica de los niveles de tioles, óxido nítrico y superoxido se podría abordar desarrollando un modelo matemático de respuesta a ambos factores.
3. Se ha comprobado que las intensidades de fluorescencia resultantes del tratamiento con Cd^{2+} y Hg^{2+} están correlacionadas con la morfología celular (medida por SSC y FSC). Aunque se han implementado scripts en R (no se muestran los resultados) son necesarios experimentos adicionales para evaluar la validez de los modelos.

5 Bibliografía

Reference List

1. Bashashati,A. and Brinkman,R.R. (2009) A survey of flow cytometry data analysis methods. *Adv. Bioinformatics*, 584603.

2. Belyaeva,E.A. et al. (2006) Reactive oxygen species produced by the mitochondrial respiratory chain are involved in Cd2+-induced injury of rat ascites hepatoma AS-30D cells. *Biochim. Biophys. Acta*, 1757, 1568-1574.

3. Belyaeva,E.A. et al. (2008) Mitochondria as an important target in heavy metal toxicity in rat hepatoma AS-30D cells. *Toxicol. Appl. Pharmacol.*, 231, 34-42.

4. Black,R., Whanger,P. and Tripp,M. (1979) Influence of silver, mercury, lead, cadmium, and selenium on glutathione peroxidase and transferase activities in rats. *Biological Trace Element Research*, 1, 313-324.

5. Crespo-Lopez,M.E. et al. (2009) Mercury and human genotoxicity: critical considerations and possible molecular mechanisms. *Pharmacol. Res.*, 60, 212-220.

6. Cuypers,A. et al. (2010) Cadmium stress: an oxidative challenge. *Biometals*.

7. de Marco,K.C. et al. (2010) Environmental exposure to methylmercury is associated with a decrease in nitric oxide production. *Basic Clin. Pharmacol. Toxicol.*, 106, 411-415.

8. Givan,A.L. (2004) Flow cytometry: an introduction. *Methods Mol. Biol.*, 263, 1-32.

9. Gratama,J.W. et al. (1998) Flow cytometric quantitation of immunofluorescence intensity: problems and perspectives. European Working Group on Clinical Cell Analysis. *Cytometry*, 33, 166-178.

10. Hahne,F. et al. (2006) Statistical methods and software for the analysis of highthroughput reverse genetic assays using flow cytometry readouts. *Genome Biol.*, 7, R77.

11. Hahne,F. et al. (2010) Per-channel basis normalization methods for flow cytometry data. *Cytometry A*, 77, 121-131.

12. Hahne,F. et al. (2009) flowCore: a Bioconductor package for high throughput flow cytometry. *BMC. Bioinformatics*, 10, 106.

13. Herrera,G. et al. (2007) Cytomics: A multiparametric, dynamic approach to cell research. *Toxicol. In Vitro*, 21, 176-182.

14. Hultberg,B., Andersson,A. and Isaksson,A. (1997) Copper ions differ from other thiol reactive metal ions in their effects on the concentration and redox status of thiols in HeLa cell cultures. *Toxicology*, 117, 89-97.

15. Hultberg,B., Andersson,A. and Isaksson,A. (1998) Alterations of thiol metabolism in human cell lines induced by low amounts of copper, mercury or cadmium ions. *Toxicology*, 126, 203-212.

16. InSug,O. et al. (1997) Mercuric compounds inhibit human monocyte function by inducing apoptosis: evidence for formation of reactive oxygen species, development of mitochondrial membrane permeability transition and loss of reductive reserve. *Toxicology*, 124, 211-224.

17. Jamall,I.S. and Smith,J.C. (1985) Effects of cadmium on glutathione peroxidase, superoxide dismutase, and lipid peroxidation in the rat heart: a possible mechanism of cadmium cardiotoxicity. *Toxicol. Appl. Pharmacol.*, 80, 33-42.

18. Johnson,I. (1998) Fluorescent probes for living cells. *Histochem. J.*, 30, 123-140.

19. Johnson,I.D. (2001) Cellular function probes. *Curr. Protoc. Cytom.*, Chapter 4, Unit.

20. Leary,J.F. (2001) Listmode data processing. *Curr. Protoc. Cytom.*, Chapter 10, Unit.

21. Lee,J.A. et al. (2008) MIFlowCyt: the minimum information about a Flow Cytometry Experiment. *Cytometry A*, 73, 926-930.

22. Lee,K. et al. (2009) iFlow: A Graphical User Interface for Flow Cytometry Tools in Bioconductor. *Adv. Bioinformatics.*, 103839.

23. Lo,K. et al. (2009) flowClust: a Bioconductor package for automated gating of flow cytometry data. *BMC. Bioinformatics.*, 10, 145.

24. Lopez,E. et al. (2006) Cadmium induces reactive oxygen species generation and lipid peroxidation in cortical neurons in culture. *Free Radic. Biol. Med.*, 40, 940-951.

25. Majumder,S. et al. (2009) Cadmium attenuates bradykinin-driven nitric oxide production by interplaying with the localization pattern of endothelial nitric oxide synthase. *Biochem. Cell Biol.*, 87, 605-620.

26. Majumder,S. et al. (2008) Cadmium reduces nitric oxide production by impairing phosphorylation of endothelial nitric oxide synthase. *Biochem. Cell Biol.*, 86, 1-10.

27. Nesovic-Ostojic,J. et al. (2008) Low micromolar concentrations of cadmium and mercury ions activate peritubular membrane K+ conductance in proximal tubular cells of frog kidney. *Comp Biochem. Physiol A Mol. Integr. Physiol*, 149, 267-274.

28. Novo,D. and Wood,J. (2008) Flow cytometry histograms: transformations, resolution, and display. *Cytometry A*, 73, 685-692.

29. Ortega-Villasante,C. et al. (2007) Rapid alteration of cellular redox homeostasis upon exposure to cadmium and mercury in alfalfa seedlings. *New Phytol.*, 176, 96-107.

30. Ortega-Villasante,C. et al. (2005) Cellular damage induced by cadmium and mercury in Medicago sativa. *J. Exp. Bot.*, 56, 2239-2251.

31. Parks,D.R., Roederer,M. and Moore,W.A. (2006) A new "Logicle" display method avoids deceptive effects of logarithmic scaling for low signals and compensated data. *Cytometry A*, 69, 541-551.

32. Reddy,C.C., Scholz,R.W. and Massaro,E.J. (1981) Cadmium, methylmercury, mercury, and lead inhibition of calf liver glutathione S-transferase exhibiting selenium-independent glutathione peroxidase activity. *Toxicol. Appl. Pharmacol.*, 61, 460-468.

33. Roederer,M. (2001) Spectral compensation for flow cytometry: visualization artifacts, limitations, and caveats. *Cytometry*, 45, 194-205.

34. Sarkar,D., Le,M.N. and Gentleman,R. (2008) Using flowViz to visualize flow cytometry data. *Bioinformatics.*, 24, 878-879.

35. Shackney,S. et al. (2006) Guidelines for improving the reproducibility of quantitative multiparameter immunofluorescence measurements by laser scanning cytometry on fixed cell suspensions from human solid tumors. *Cytometry B Clin. Cytom.*, 70, 10-19.

36. Smith,C.A. et al. (2006) A simple correction for cell autofluorescence for multiparameter cell-based analysis of human solid tumors. *Cytometry B Clin. Cytom.*, 70, 91-103.

37. Smith,P.J., Khan,I.A. and Errington,R.J. (2007) Cytomics and drug development. *Cytometry A*, 71, 349-351.

38. Spidlen,J. et al. (2006) Data standards for flow cytometry. *OMICS.*, 10, 209-214.

39. Spidlen,J. et al. (2008) Gating-ML: XML-based gating descriptions in flow cytometry. *Cytometry A*, 73A, 1151-1157.

40. Spidlen,J. et al. (2010) Data File Standard for Flow Cytometry, version FCS 3.1. *Cytometry A*, 77, 97-100.

41. Strain,E. et al. (2009) Analysis of High-Throughput Flow Cytometry Data Using plateCore. *Adv. Bioinformatics.*, 356141.

42. Valet,G. (2006) Cytomics as a new potential for drug discovery. *Drug Discov. Today*, 11, 785-791.

43. Valko,M., Morris,H. and Cronin,M.T. (2005) Metals, toxicity and oxidative stress. *Curr. Med. Chem.*, 12, 1161-1208.

44. Vermes,I., Haanen,C. and Reutelingsperger,C. (2000) Flow cytometry of apoptotic cell death. *J. Immunol. Methods*, 243, 167-190.

45. Wood,J.C. (1998) Fundamental flow cytometer properties governing sensitivity and resolution. *Cytometry*, 33, 260-266.

46. Woods,J.S. and Ellis,M.E. (1995) Up-regulation of glutathione synthesis in rat kidney by methyl mercury. Relationship to mercury-induced oxidative stress. *Biochem. Pharmacol.*, 50, 1719-1724.

Análisis de datos de Citometría de flujo con R-Bioconductor. Aplicación al estudio de marcadores de funcionalidad celular en toxicología.

Máster en Aproximaciones Moleculares en Ciencias de la Salud

Trabajo Fin de Master

RAMÓN TAMARIT AGUSTÍ

ANEXO

1 Tabla S1. Fichero de anotación de experimentos.

file	Id	Date	panel	StainId	CTox	Tox
1A-Cd0.FCS	1A-Cd000	20100305	1	A	0	Cd
1A-Cd1.FCS	1A-Cd001	20100305	1	A	5	Cd
1A-Cd2.FCS	1A-Cd002	20100305	1	A	10	Cd
1A-Hg00.FCS	1A-Hg000	20100305	1	A	0	Hg
1A-Hg01.FCS	1A-Hg001	20100305	1	A	1	Hg
1A-Hg05.FCS	1A-Hg005	20100305	1	A	5	Hg
1A-Hg10.FCS	1A-Hg010	20100305	1	A	10	Hg
1B-Cd0.FCS	1B-Cd000	20100305	1	B	0	Cd
1B-Cd1.FCS	1B-Cd001	20100305	1	B	5	Cd
1B-Cd2.FCS	1B-Cd002	20100305	1	B	10	Cd
1B-Hg00.FCS	1B-Hg000	20100305	1	B	0	Hg
1B-Hg01.FCS	1B-Hg001	20100305	1	B	1	Hg
1B-Hg05.FCS	1B-Hg005	20100305	1	B	5	Hg
1B-Hg10.FCS	1B-Hg010	20100305	1	B	10	Hg
1C-Cd0.FCS	1C-Cd000	20100305	1	C	0	Cd
1C-Cd1.FCS	1C-Cd001	20100305	1	C	5	Cd
1C-Cd2.FCS	1C-Cd002	20100305	1	C	10	Cd
1C-Hg00.FCS	1C-Hg000	20100305	1	C	0	Hg
1C-Hg01.FCS	1C-Hg001	20100305	1	C	1	Hg
1C-Hg05.FCS	1C-Hg005	20100305	1	C	5	Hg
1C-Hg10.FCS	1C-Hg010	20100305	1	C	10	Hg
1D-Cd0.FCS	1D-Cd000	20100305	1	D	0	Cd
1D-Cd1.FCS	1D-Cd001	20100305	1	D	5	Cd
1D-Cd2.FCS	1D-Cd002	20100305	1	D	10	Cd
1D-Hg00.FCS	1D-Hg000	20100305	1	D	0	Hg
1D-Hg01.FCS	1D-Hg001	20100305	1	D	1	Hg
1D-Hg05.FCS	1D-Hg005	20100305	1	D	5	Hg
1D-Hg10.FCS	1D-Hg010	20100305	1	D	10	Hg
1E-Cd0.FCS	1E-Cd000	20100305	1	E	0	Cd
1E-Cd1.FCS	1E-Cd001	20100305	1	E	5	Cd
1E-Cd2.FCS	1E-Cd002	20100305	1	E	10	Cd
1E-Hg00.FCS	1E-Hg000	20100305	1	E	0	Hg
1E-Hg01.FCS	1E-Hg001	20100305	1	E	1	Hg
1E-Hg05.FCS	1E-Hg005	20100305	1	E	5	Hg
1E-Hg10.FCS	1E-Hg010	20100305	1	E	10	Hg
1O-N0.FCS	1O-N000	20100417	1	O	0	N
2A-Cd0.FCS	2A-Cd000	20100305	2	A	0	Cd
2A-Cd1.FCS	2A-Cd001	20100305	2	A	5	Cd
2A-Cd2.FCS	2A-Cd002	20100305	2	A	10	Cd
2A-Hg00.FCS	2A-Hg000	20100305	2	A	0	Hg
2A-Hg01.FCS	2A-Hg001	20100305	2	A	1	Hg
2A-Hg05.FCS	2A-Hg005	20100305	2	A	5	Hg
2A-Hg10.FCS	2A-Hg010	20100305	2	A	10	Hg
2B-Cd0.FCS	2B-Cd000	20100305	2	B	0	Cd
2B-Cd1.FCS	2B-Cd001	20100305	2	B	5	Cd
2B-Cd2.FCS	2B-Cd002	20100305	2	B	10	Cd

file	Id	Date	panel	StainId	CTox	Tox
2B-Hg00.FCS	2B-Hg000	20100305	2	B	0	Hg
2B-Hg01.FCS	2B-Hg001	20100305	2	B	1	Hg
2B-Hg05.FCS	2B-Hg005	20100305	2	B	5	Hg
2B-Hg10.FCS	2B-Hg010	20100305	2	B	10	Hg
2C-Cd0.FCS	2C-Cd000	20100305	2	C	0	Cd
2C-Cd1.FCS	2C-Cd001	20100305	2	C	5	Cd
2C-Cd2.FCS	2C-Cd002	20100305	2	C	10	Cd
2C-Hg00.FCS	2C-Hg000	20100305	2	C	0	Hg
2C-Hg01.FCS	2C-Hg001	20100305	2	C	1	Hg
2C-Hg05.FCS	2C-Hg005	20100305	2	C	5	Hg
2C-Hg10.FCS	2C-Hg010	20100305	2	C	10	Hg
2D-Cd0.FCS	2D-Cd000	20100305	2	D	0	Cd
2D-Cd1.FCS	2D-Cd001	20100305	2	D	5	Cd
2D-Cd2.FCS	2D-Cd002	20100305	2	D	10	Cd
2D-Hg00.FCS	2D-Hg000	20100305	2	D	0	Hg
2D-Hg01.FCS	2D-Hg001	20100305	2	D	1	Hg
2D-Hg05.FCS	2D-Hg005	20100305	2	D	5	Hg
2D-Hg10.FCS	2D-Hg010	20100305	2	D	10	Hg
2E-Cd0.FCS	2E-Cd000	20100305	2	E	0	Cd
2E-Cd1.FCS	2E-Cd001	20100305	2	E	5	Cd
2E-Cd2.FCS	2E-Cd002	20100305	2	E	10	Cd
2E-Hg00.FCS	2E-Hg000	20100305	2	E	0	Hg
2E-Hg01.FCS	2E-Hg001	20100305	2	E	1	Hg
2E-Hg05.FCS	2E-Hg005	20100305	2	E	5	Hg
2E-Hg10.FCS	2E-Hg010	20100305	2	E	10	Hg
2O-N0.FCS	2O-N000	20100417	2	O	0	N
3A-Cd0.FCS	3A-Cd000	20100305	3	A	0	Cd
3A-Cd1.FCS	3A-Cd001	20100305	3	A	5	Cd
3A-Cd2.FCS	3A-Cd002	20100305	3	A	10	Cd
3A-Hg00.FCS	3A-Hg000	20100305	3	A	0	Hg
3A-Hg01.FCS	3A-Hg001	20100305	3	A	1	Hg
3A-Hg05.FCS	3A-Hg005	20100305	3	A	5	Hg
3A-Hg10.FCS	3A-Hg010	20100305	3	A	10	Hg
3B-Cd0.FCS	3B-Cd000	20100305	3	B	0	Cd
3B-Cd1.FCS	3B-Cd001	20100305	3	B	5	Cd
3B-Cd2.FCS	3B-Cd002	20100305	3	B	10	Cd
3B-Hg00.FCS	3B-Hg000	20100305	3	B	0	Hg
3B-Hg01.FCS	3B-Hg001	20100305	3	B	1	Hg
3B-Hg05.FCS	3B-Hg005	20100305	3	B	5	Hg
3B-Hg10.FCS	3B-Hg010	20100305	3	B	10	Hg
3C-Cd0.FCS	3C-Cd000	20100305	3	C	0	Cd
3C-Cd1.FCS	3C-Cd001	20100305	3	C	5	Cd
3C-Cd2.FCS	3C-Cd002	20100305	3	C	10	Cd
3C-Hg00.FCS	3C-Hg000	20100305	3	C	0	Hg
3C-Hg01.FCS	3C-Hg001	20100305	3	C	1	Hg
3C-Hg05.FCS	3C-Hg005	20100305	3	C	5	Hg
3C-Hg10.FCS	3C-Hg010	20100305	3	C	10	Hg
3D-Cd0.FCS	3D-Cd000	20100305	3	D	0	Cd
3D-Cd1.FCS	3D-Cd001	20100305	3	D	5	Cd

file	Id	Date	panel	StainId	CTox	Tox
3D-Cd2.FCS	3D-Cd002	20100305	3	D	10	Cd
3D-Hg00.FCS	3D-Hg000	20100305	3	D	0	Hg
3D-Hg01.FCS	3D-Hg001	20100305	3	D	1	Hg
3D-Hg05.FCS	3D-Hg005	20100305	3	D	5	Hg
3D-Hg10.FCS	3D-Hg010	20100305	3	D	10	Hg
3E-Cd0.FCS	3E-Cd000	20100305	3	E	0	Cd
3E-Cd1.FCS	3E-Cd001	20100305	3	E	5	Cd
3E-Cd2.FCS	3E-Cd002	20100305	3	E	10	Cd
3E-Hg00.FCS	3E-Hg000	20100305	3	E	0	Hg
3E-Hg01.FCS	3E-Hg001	20100305	3	E	1	Hg
3E-Hg05.FCS	3E-Hg005	20100305	3	E	5	Hg
3E-Hg10.FCS	3E-Hg010	20100305	3	E	10	Hg
3O-N0.FCS	3O-N000	20100417	3	O	0	N
PARAQ0 2A.fcs	2A-PARAQ000	20090620	2	A	0	PARAQ
PARAQ100 2A.fcs	2A-PARAQ100	20090620	2	A	100	PARAQ
PARAQ200 2A.fcs	2A-PARAQ200	20090620	2	A	200	PARAQ
PARAQ400 2A.fcs	2A-PARAQ400	20090620	2	A	400	PARAQ
PARAQ0 2B.fcs	2B-PARAQ000	20090620	2	B	0	PARAQ
PARAQ100 2B.fcs	2B-PARAQ100	20090620	2	B	100	PARAQ
PARAQ200 2B.fcs	2B-PARAQ200	20090620	2	B	200	PARAQ
PARAQ400 2B.fcs	2B-PARAQ400	20090620	2	B	400	PARAQ
NOR0 3A.fcs	3A-NOR000	20090620	3	A	0	NOR
NOR1 3A.fcs	3A-NOR001	20090620	3	A	1	NOR
NOR0 3B.fcs	3B-NOR000	20090620	3	B	0	NOR
NOR1 3B.fcs	3B-NOR001	20090620	3	B	1	NOR
IONOM0 1A.fcs	1A-IONOM000	20090620	1	A	0	IONOM
IONOM1 1A.fcs	1A-IONOM001	20090620	1	A	70	IONOM
IONOM0 1D.fcs	1D-IONOM000	20090620	1	D	0	IONOM
IONOM1 1D.fcs	1D-IONOM001	20090620	1	D	70	IONOM
GRAM0 2A.fcs	2A-GRAM000	20090620	2	A	0	GRAM
GRAM10 2A.fcs	2A-GRAM010	20090620	2	A	10	GRAM
GRAM20 2A.fcs	2A-GRAM020	20090620	2	A	20	GRAM
GRAM0 2B.fcs	2B-GRAM000	20090620	2	B	0	GRAM
GRAM10 2B.fcs	2B-GRAM010	20090620	2	B	10	GRAM
GRAM20 2B.fcs	2B-GRAM020	20090620	2	B	20	GRAM
FCCP0 1A.fcs	1A-FCCPP000	20090620	1	A	0	FCCPP
FCCP1 1A.fcs	1A-FCCPP001	20090620	1	A	20	FCCPP
FCCP0 1C.fcs	1C-FCCPP000	20090620	1	C	0	FCCPP
FCCP1 1C.fcs	1C-FCCPP001	20090620	1	C	20	FCCPP
CHP0 1A.fcs	1A-CHP000	20090620	1	A	0	CHP
CHP100 1A.fcs	1A-CHP100	20090620	1	A	100	CHP
CHP0 1B.fcs	1B-CHP000	20090620	1	B	0	CHP
CHP100 1B.fcs	1B-CHP100	20090620	1	B	100	CHP
BSO0 2A.fcs	2A-BSO000	20090620	2	A	0	BSO
BSO0.5 2A.fcs	2A-BSO005	20090620	2	A	5	BSO
BSO0.5 2D.fcs	2D-BSO005	20090620	2	D	5	BSO
BSO0 2D.fcs	2D-BSO000	20090620	2	D	0	BSO

Unidades de concentración de los tóxicos (columna CTox):

- GRAM, Hg= µg/ml (1 µg/ml Hg =aprox. 3.7 µM)
- BSO = mM
- Cd, PARAQ, CHP, IONOM, FCCPP µM
- NOR mg/ml

Tabla S1. Fichero de anotación de experimentos.

2 Tablas S2. Resultados del script "*Analisis.positive.ctrl.R.*

Panel 1

				FL1	FL3	FL6	FL8
Files	StainId	CTox	Tox	DCF (Actividad peroxidasa)	TMRM (Potencial de membrana mitocondrial)	Indo-1 (Calcio intracelular.)	BODIPY 675 (Peroxidación lipídica)
1A-Cd001	A	1	Cd	1,01	1,11	1,25	1,24
1A-Cd002	A	2	Cd	1,02	1,36	1,48	1,40
1A-CHP100	A	100	CHP	2,25	3,12	3,39	1,12
1A-FCCPP001	A	1	FCCPP	1,25	0,52	0,74	1,00
1A-Hg001	A	1	Hg	1,30	1,20	0,99	1,23
1A-Hg005	A	5	Hg	1,11	1,35	0,76	1,15
1A-Hg010	A	10	Hg	1,33	1,69	0,96	1,39
1A-IONOM001	A	1	IONOM	2,74	0,98	1,93	0,97
1B-Cd001	B	1	Cd	1,13	1,12	1,19	0,89
1B-Cd002	B	2	Cd	1,21	1,15	1,27	0,97
1B-CHP100	B	100	CHP	2,71	1,85	1,31	1,09
1B-Hg001	B	1	Hg	1,05	1,00	1,18	0,98
1B-Hg005	B	5	Hg	2,13	1,84	1,46	1,20
1B-Hg010	B	10	Hg	1,94	1,71	1,40	1,46
1C-Cd001	C	1	Cd	1,20	1,29	1,25	1,11
1C-Cd002	C	2	Cd	1,20	1,36	1,30	1,04
1C-FCCPP001	C	1	FCCPP	1,52	0,53	0,61	1,03
1C-Hg001	C	1	Hg	1,07	1,20	1,31	1,07
1C-Hg005	C	5	Hg	1,60	1,69	2,26	1,36
1C-Hg010	C	10	Hg	1,93	1,85	1,75	1,94
1D-Cd001	D	1	Cd	1,19	1,12	0,92	1,04
1D-Cd002	D	2	Cd	1,26	1,18	1,16	1,10
1D-Hg001	D	1	Hg	1,16	0,90	0,91	1,06
1D-Hg005	D	5	Hg	1,36	1,02	0,95	1,26
1D-Hg010	D	10	Hg	1,67	1,18	1,10	1,65
1D-IONOM001	D	1	IONOM	2,41	1,51	1,88	0,97
1E-Cd001	E	1	Cd	1,07	1,10	1,44	0,93
1E-Cd002	E	2	Cd	1,11	1,14	1,46	1,05
1E-Hg001	E	1	Hg	1,29	1,20	1,28	1,18
1E-Hg005	E	5	Hg	1,60	1,46	1,74	1,35
1E-Hg010	E	10	Hg	2,13	1,82	1,34	1,22

Tabla S2.1. Resultados del script "*Analisis.positive.ctrl.R*" para el Panel 1. Los valores significativos se marcan en rojo para la disminución y en azul para el aumento. Los valores resaltados en amarillo corresponden a los controles en los que se conoce la actividad sobre ese parámetro

Panel 2

				FL1	FL3	FL6	FL8
Files	Stain	CTox	Tox	DiBAC3(4) (Potencial de membrana plasmática.)	Mitosox (Superoxido mitocondrial)	MCB (Niveles de tioles libres)	BODIPY 675 (Peroxidación lipídica)
2A-BSO005	A	5	BSO	1,20	1,45	0,54	1,28
2A-Cd001	A	1	Cd	0,86	1,09	1,32	-0,24
2A-Cd002	A	2	Cd	0,97	1,10	1,54	-1,21
2A-GRAM010	A	10	GRAM	3,07	0,98	1,06	1,14
2A-GRAM020	A	20	GRAM	3,88	1,15	1,04	1,09
2A-Hg001	A	1	Hg	1,07	1,13	1,20	0,58
2A-Hg005	A	5	Hg	1,22	1,33	1,75	-0,32
2A-Hg010	A	10	Hg	1,31	1,32	1,80	0,03
2A-PARAQ100	A	100	PARAQ	0,97	1,08	1,01	1,12
2A-PARAQ200	A	200	PARAQ	1,33	1,34	1,13	1,13
2A-PARAQ400	A	400	PARAQ	2,12	20,32	0,45	1,02
2B-Cd001	B	1	Cd	1,09	1,08	1,32	0,97
2B-Cd002	B	2	Cd	1,39	1,20	1,48	1,04
2B-GRAM010	B	10	GRAM	4,77	1,53	1,11	0,98
2B-GRAM020	B	20	GRAM	5,92	1,81	0,88	0,94
2B-Hg001	B	1	Hg	1,06	1,05	1,22	1,09
2B-Hg005	B	5	Hg	0,95	1,13	1,59	1,26
2B-Hg010	B	10	Hg	1,06	1,22	1,54	1,53
2B-PARAQ100	B	100	PARAQ	1,13	1,24	0,92	1,18
2B-PARAQ200	B	200	PARAQ	1,46	1,59	0,83	1,35
2B-PARAQ400	B	400	PARAQ	2,49	3,88	0,53	1,78
2C-Cd001	C	1	Cd	1,13	1,14	1,40	1,06
2C-Cd002	C	2	Cd	1,25	1,40	1,67	1,06
2C-Hg001	C	1	Hg	0,78	0,92	1,17	0,97
2C-Hg005	C	5	Hg	1,10	1,44	1,53	1,13
2C-Hg010	C	10	Hg	1,22	1,45	1,53	1,38
2D-BSO005	D	5	BSO	0,79	0,87	0,47	0,81
2D-Cd001	D	1	Cd	0,98	1,06	1,50	1,08
2D-Cd002	D	2	Cd	0,96	1,07	1,79	1,10
2D-Hg001	D	1	Hg	1,04	1,09	1,18	1,05
2D-Hg005	D	5	Hg	1,18	1,27	1,64	1,15
2D-Hg010	D	10	Hg	1,38	1,39	1,76	1,24
2E-Cd001	E	1	Cd	1,07	1,10	1,44	0,93
2E-Cd002	E	2	Cd	1,11	1,14	1,46	1,05
2E-Hg001	E	1	Hg	1,29	1,20	1,28	1,18
2E-Hg005	E	5	Hg	1,60	1,47	1,74	1,35
2E-Hg010	E	10	Hg	2,13	1,82	1,34	1,22

Tabla S2.2. Resultados del script "*Analisis.positive.ctrl.R*" para el Panel 2. Los valores significativos se marcan en rojo para la disminución y en azul para el aumento. Los valores resaltados en amarillo corresponden a los controles en los que se conoce la actividad sobre ese parámetro

Panel 3

Files	Stain	CTox	Tox	FL1 DAF (Niveles de óxido nítrico)	FL3 TMRM (Potencial de membrana mitocondrial)	FL6 MCB (Niveles de tioles libres)	FL8 BODIPY 675 (Peroxidación lipídica)
3A-Cd001	A	1	Cd	1,10	1,03	1,40	1,12
3A-Cd002	A	2	Cd	1,18	1,47	1,64	1,34
3A-Hg001	A	1	Hg	1,14	0,87	1,28	0,94
3A-Hg005	A	5	Hg	1,36	1,11	1,51	1,11
3A-Hg010	A	10	Hg	1,55	1,18	1,79	1,35
3A-NOR001	A	1	NOR	2,71	0,55	0,92	0,91
3B-Cd001	B	1	Cd	0,87	0,90	1,27	0,97
3B-Cd002	B	2	Cd	0,87	0,93	1,39	1,00
3B-Hg001	B	1	Hg	1,29	1,29	1,26	1,17
3B-Hg005	B	5	Hg	1,54	1,58	1,46	1,22
3B-Hg010	B	10	Hg	1,56	1,57	1,45	1,33
3B-NOR001	B	1	NOR	4,88	3,10	0,93	0,84
3C-Cd001	C	1	Cd	1,18	1,27	1,23	1,10
3C-Cd002	C	2	Cd	1,18	1,34	1,28	1,03
3C-Hg001	C	1	Hg	1,07	1,20	1,31	1,07
3C-Hg005	C	5	Hg	1,61	1,69	2,26	1,37
3C-Hg010	C	10	Hg	1,93	1,86	1,75	1,95
3D-Cd001	D	1	Cd	0,98	1,06	1,50	1,08
3D-Cd002	D	2	Cd	0,96	1,07	1,79	1,10
3D-Hg001	D	1	Hg	1,04	1,09	1,18	1,05
3D-Hg005	D	5	Hg	1,18	1,27	1,64	1,15
3D-Hg010	D	10	Hg	1,38	1,39	1,76	1,24
3E-Cd001	E	1	Cd	1,07	1,10	1,44	0,93
3E-Cd002	E	2	Cd	1,11	1,14	1,46	1,05
3E-Hg001	E	1	Hg	1,28	1,20	1,27	1,18
3E-Hg005	E	5	Hg	1,60	1,46	1,74	1,35
3E-Hg010	E	10	Hg	2,13	1,82	1,34	1,22

Tabla S2.3. Resultados del script "*Analisis.positive.ctrl.R*" para el Panel 3. Los valores significativos se marcan en rojo para la disminución y en azul para el aumento. Los valores resaltados en amarillo corresponden a los controles en los que se conoce la actividad sobre ese parámetro

3 Figuras suplementarias

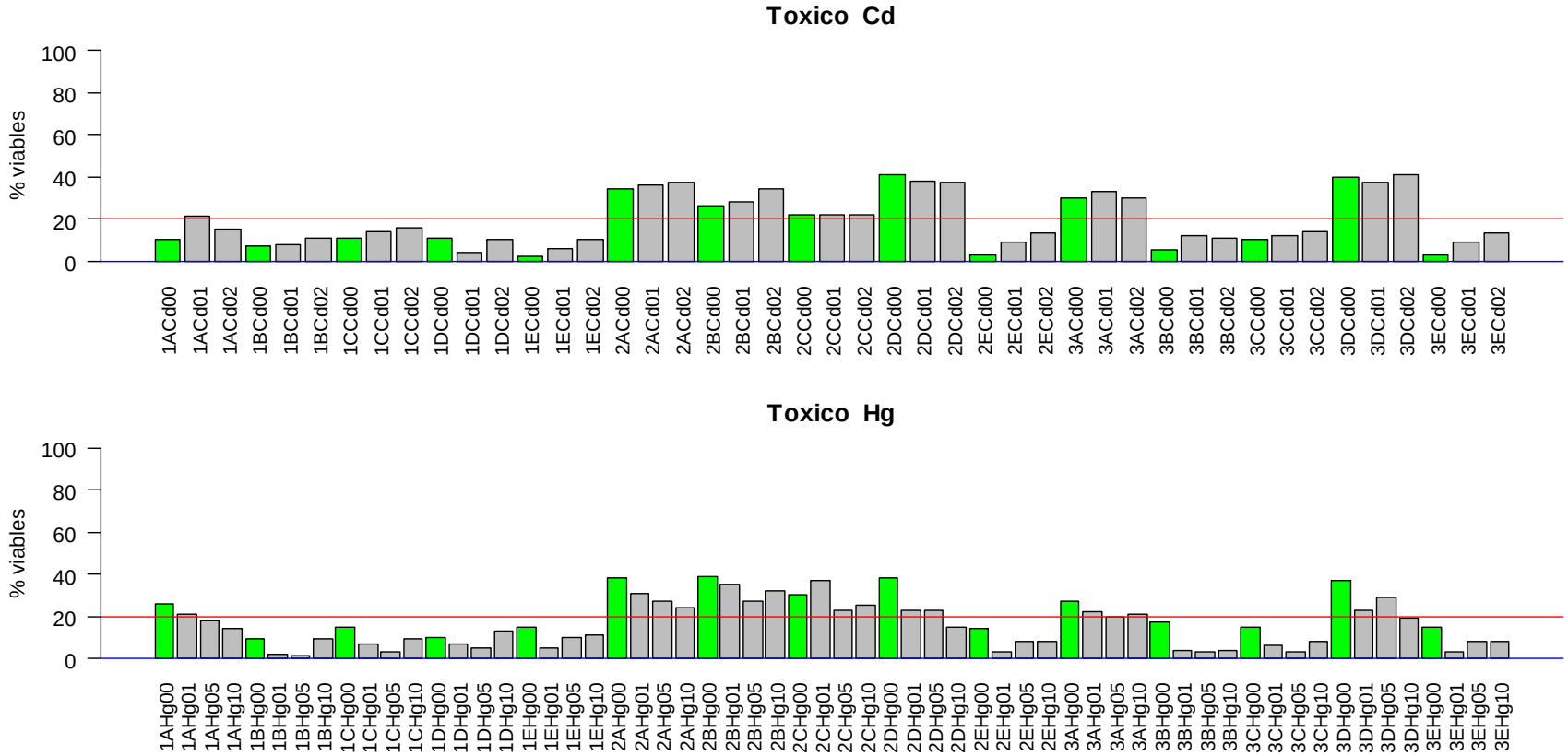

Figura S1. Porcentaje de células viables, tras la selección con un filtro rectangular (FSC minimo:500 , FSC maximo:2400 , SSC minimo:200, SSC maximo: 2000) y posteriormente un filtro "*norm2filter*" que encierra una región que incluye el 95% de la población).

4 Scripts.

Por motivos de espacio los scripts se incluyen en la versión electrónica de este documento.

```
  ### PROGRAMA: head.repair.R
  ### VERSION: 3.01.06.2010
  ### Repara la cabecera de los ficheros FCS

#########################################################
### chunk number 1: Load Libraries
#########################################################
 library(flowCore)
 library(flowQ)
 library(flowViz)
 library(flowStats)
 library(flowUtils)
 library(geneplotter)
 library(colorspace)
 library(grid)
 library(MASS)
 library(flowFP)



#########################################################
### chunk number 0: Set working dir
#########################################################
###workingDir <- getwd()
dataDir <- "C:/Users/DataPrj"
dataDirrep <- paste(dataDir , "/out/",sep="")
setwd(dataDir)
pat<-".\\.fcs"
ffiles <- list.files(, pattern = pat)
ffiles
for (i in 1:length(ffiles))
  {
  fichero<-ffiles[i]
  print(fichero)
  fichero.out<-paste(dataDirrep ,fichero, sep="")
  print(fichero.out)

  frameRead1<-read.FCS(fichero, transformation = FALSE)

  frameRead.out<-frameRead1
  names(frameRead.out)
  featureNames(frameRead.out)
  frameRead.out

  expresion<-exprs(frameRead.out)
  expresion
  framenew<-new("flowFrame",exprs = expresion)

  keyword(framenew) <- list ( `$P1E`="0.0,0.0" )
  keyword(framenew) <- list ( `$P2E`="0.0,0.0" )
  keyword(framenew) <- list ( `$P3E`="4.0,1.0" )
  keyword(framenew) <- list ( `$P4E`="4.0,1.0" )
  keyword(framenew) <- list ( `$P5E`="4.0,1.0" )
  keyword(framenew) <- list ( `$P6E`="4.0,1.0" )


  keyword(framenew) <- list ( `$P1R`="4096" )
  keyword(framenew) <- list ( `$P2R`="4096" )
  keyword(framenew) <- list ( `$P3R`="4096" )
  keyword(framenew) <- list ( `$P4R`="4096" )
  keyword(framenew) <- list ( `$P5R`="4096" )
  keyword(framenew) <- list ( `$P6R`="4096" )


  keyword(framenew) <- list ( `$P1B`="16" )
  keyword(framenew) <- list ( `$P2B`="16" )
  keyword(framenew) <- list ( `$P3B`="16" )
  keyword(framenew) <- list ( `$P4B`="16" )
```

```
  keyword(framenew) <- list ( `$P5B`="16" )
  keyword(framenew) <- list ( `$P6B`="16" )


  keyword(framenew) <- list ( `$P1N`="FSC" )
  keyword(framenew) <- list ( `$P2N`="SSC" )
  keyword(framenew) <- list ( `$P3N`="FL1" )
  keyword(framenew) <- list ( `$P4N`="FL3" )
  keyword(framenew) <- list ( `$P5N`="FL6" )
  keyword(framenew) <- list ( `$P6N`="FL8" )


  keyword(framenew) <- list ( `$P1S`="FSC" )
  keyword(framenew) <- list ( `$P2S`="SSC" )
  keyword(framenew) <- list ( `$P3S`="flFL1" )
  keyword(framenew) <- list ( `$P4S`="flFL3" )
  keyword(framenew) <- list ( `$P5S`="flFL6" )
  keyword(framenew) <- list ( `$P6S`="flFL8" )


  keyword(framenew) <- list ( `$P1L`="    0.000000" )
  keyword(framenew) <- list ( `$P2L`= "   0.000000" )
  keyword(framenew) <- list ( `$P3L`="    0.000000" )
  keyword(framenew) <- list ( `$P4L`="    0.000000" )
  keyword(framenew) <- list ( `$P5L`="    0.000000" )
  keyword(framenew) <- list ( `$P6L`="    0.000000" )


  keyword(framenew) <- list ( `$P1P`="0" )
  keyword(framenew) <- list ( `$P2P`="0" )
  keyword(framenew) <- list ( `$P3P`="0" )
  keyword(framenew) <- list ( `$P4P`="0" )
  keyword(framenew) <- list ( `$P5P`="0" )
  keyword(framenew) <- list ( `$P6P`="0" )


  #x11()
  #splom(framenew)
  write.FCS(framenew, fichero.out, what="integer")
  framenew2<-read.FCS(fichero.out, transformation = FALSE )
  #x11()
  #splom(framenew2)


  }
```

```
 1:
 2:  ### PROGRAMA: fluo.compensation.R
 3:  ### VERSION: 5.11.07.2010
 4:  ### CALCULO DE LA MATRIZ DE COMPENSACION CON DATOS LINEALES
 5:  ### Y POSTERIOR TRANSFORMACIÓN LINLOG.
 6:  ### CALCULO DE MEDIAS Y MEDIANAS DE UNA SERIE DE EXPERIMENTOS
 7:
 8:  ########################################################
 9:  ### chunk 1: Load Libraries
10:  ########################################################
11:
12:
13:   library(flowCore)
14:   library(flowQ)
15:   library(flowViz)
16:   library(flowStats)
17:   library(flowUtils)
18:   library(geneplotter)
19:   library(colorspace)
20:   library(grid)
21:   library(MASS)
22:   library(flowFP)
23:   library(geneplotter)
24:   require(RColorBrewer)
25:   library(MASS)
26:   ## Descomentar para limpiar memoria ##########
27:   # print(Objlist <- ls())
28:   # rm(Objlist )
29:
30:   ########################
31:   ## Puede ser necesario aumentar la memoria asignada a R
32:   ## dependienendo del numero de flowFrames.
33:   memory.size()
34:       memory.size(TRUE)
35:       memory.limit()
36:       memory.size(max = 2000)
37:       memory.size()
38:       memory.size(TRUE)
39:       memory.limit()
40:
41:   ### Se requiere la carga de la libreia de funciones de apoyo.
42:       source("C:/Users/ramon/DataPrj/CdHg/funciones.propias.R",echo = TRUE)
43:
44:  ########################################################
45:  ### chunk 2: Definicion de las variables globales
46:  ###      para la ejecucion del script.
47:  ########################################################
48:  ###workingDir <- getwd()
49:  ## Definir el directorio en donde están almacenados los ficheros
50:  ## FCS de los experimentos.
51:  dataDir <- "C:/Users/DataPrj/"
52:  setwd(dataDir)
53:
54:
55:  ### Definir el directorio de salida de los ficheros FCS compensados
56:  dataDirOut <-"C:/Users/DataPrj/CompensacionOutPut"
57:
58:
59:
60:  ## Si VisualizarResultado="S" se genera un output visual
61:  VisualizarResultado <- "N"
62:  print(paste("### VisualizarResultado <- " , VisualizarResultado) )
63:
64:  ## Si ifDebug="S" se ejecutan diferentes tests necesarios para
65:  ## depurar el código y visualizar los resultados intermedios.
66:  ifDebug <- FALSE
67:  print(paste("### ifDebug <- " , ifDebug ) )
68:
```

```
69:   ## Matrices de anonatación que contienen los experimentos
70:   ## con Cd y Hg separados. Es adecuado cargar una u otra debido
71:   ## a que las funciones de visualización necesitan parametros
72:   ## diferentes.
73:
74:   PanelToxico <- "Cd"
75:   if (PanelToxico=="Cd"){
76:     matPanels <- c(
77:                   "cd_hg_annotation_bl_pnl1Cd.txt",
78:                   "cd_hg_annotation_bl_pnl2Cd.txt",
79:                   "cd_hg_annotation_bl_pnl3Cd.txt")
80:
81:     ##para cd
82:     laycol <-c(3,5)
83:     my.layout=c(3,6)
84:     rango <- c(2:16)
85:   }
86:
87:   if (PanelToxico=="Hg"){
88:     matPanels <- c(
89:                   "cd_hg_annotation_bl_pnl1Hg.txt",
90:                   "cd_hg_annotation_bl_pnl2Hg.txt",
91:                   "cd_hg_annotation_bl_pnl3Hg.txt")
92:     ### Parametros para la visualización
93:     ##para Hg
94:     laycol <-c(4,5)
95:     my.layout=c(4,6)
96:     rango <- c(2:21)
97:   }
98:
99:   ## Descripcion de los marcadores asociados a cada canal en los diferentes
100:  ## paneles de experimentos. Se almacena en una lista. cada elemento de la
101:  ## lista contiene una matriz con la asociacion de cada canal de fluorescencia
102:  ## a su marcador.
103:
104:  panel.desc <- list()
105:  panel.desc[[1]] <- c("FSC", "SSC", "FL1.DCF"   ,
106:                       "FL3.TMRM", "FL6.Indo1", "FL8.BDP675")
107:
108:  panel.desc[[2]] <- c("FSC", "SSC", "FL1.DIBAC3",
109:                       "FL3.MTSX", "FL6.MCB"  , "FL8.BDP675")
110:
111:  panel.desc[[3]] <- c("FSC", "SSC", "FL1.DAF"   ,  "FL3.TMRM",
112:                       "FL6.MCB"  , "FL8.BDP675")
113:  for (r in 1:length(panel.desc)){
114:      names(panel.desc[[r]]) <- c("FSC", "SSC", "FL1" , "FL3", "FL6", "FL8")
115:      }
116:
117:
118:  ########################################################
119:  ### chunk 3 : Bucle principal de ejecuciçon.
120:  ###     Iteraciçon sobre los flowSet de la
121:  ###     matriz matPanels
122:  ########################################################
123:
124:  ## Inicio de la iteración a través de los paneles descritos en la matriz
125:  for (panelFile in 1:length(matPanels)) {
126:  ##for debug panelFile <- 3
127:  print("#################### Carga del Panel  ################################")
128:  ExpList <- matPanels[panelFile]
129:  flowData <- read.flowSet(path = ".", phenoData = ExpList, transformation=FALSE)
130:  sampleNames(flowData) <- as.character((pData(flowData)[, "Id"]))
131:  print("###############Experimentos################")
132:  Titulos<-sampleNames(flowData)
133:  Titulos
134:  pData(phenoData(flowData))
135:  head(pData(flowData))
136:  sampleNames(flowData)
```

```
137:   Tclist<- sampleNames(flowData)
138:   factores <-pData(flowData)
139:   columnas<-colnames(flowData[[1]])
140:
141:   viewPars(flowData[1])
142:   summary(flowData[1:5])
143:   flowData.orig <- flowData
144:
145:
146:   ######################################################
147:   ### chunk 4: CARGA DEL PANEL CON DATOS LINEALES
148:   ######################################################
149:   flowData.lin <- read.flowSet(path = ".", phenoData = ExpList,
150:      transformation="linearize")
151:   sampleNames(flowData.lin ) <- as.character((pData(flowData)[, "Id"]))
152:
153:
154:
155:
156:   ######################################################
157:   ### chunk 5: ## NORMALIZATION
158:   ######################################################
159:
160:   #Normalizamos los canales FSC y SSC ya que los parametros de captura no son
161:   # Iguales en todos los experimentos
162:
163:   ##### normalizar datos en escala lineal
164:   pars<-c("FSC")
165:   norm <- normalization(normFun = function(x, parameters, ...)
166:     warpSet(x, parameters, ...), parameters = pars, normalizationId = "Warping")
167:   flowData.lin.N1<-normalize(flowData.lin, norm)
168:
169:   pars<-c("SSC")
170:   norm <- normalization(normFun = function(x, parameters, ...)
171:     warpSet(x, parameters, ...), parameters = pars, normalizationId = "Warping")
172:   flowData.lin.N2<-normalize(flowData.lin.N1, norm)
173:
174:   flowData.lin.N <- flowData.lin.N2
175:
176:   viewPars(flowData.lin.N[1])
177:   summary(flowData.lin.N[1:5])
178:
179:   ##### normalizar datos en escala Logaritmica
180:
181:   pars<-c("FSC")
182:   norm <- normalization(normFun = function(x, parameters, ...)
183:     warpSet(x, parameters, ...), parameters = pars, normalizationId = "Warping")
184:   flowData.orig.N1<-normalize(flowData.orig, norm)
185:
186:   pars<-c("SSC")
187:   norm <- normalization(normFun = function(x, parameters, ...)
188:     warpSet(x, parameters, ...), parameters = pars, normalizationId = "Warping")
189:   flowData.orig.N2<-normalize(flowData.orig.N1, norm)
190:
191:   flowData.orig.N <- flowData.orig.N2
192:
193:   viewPars(flowData.lin.N[1])
194:   summary(flowData.lin.N[1:5])
195:
196:   print("
197:   ######################################################################
198:   ### chunk 6: Definición de Filtros para selección de las celulas viables
199:   ######################################################################
200:
201:   ######################################################################
202:   ##### Configuracion de filtro rectangular
203:   print(minFSC <- 500)
204:   print(maxFSC <- 2200)
```

```
205:   print(minSSC <- 200)
206:   print(maxSSC <- 2000)
207:   rectGate <- rectangleGate(filterId="rectangleGate",
208:             "FSC"=c(minFSC , maxFSC),
209:             "SSC"=c(minSSC , maxSSC))
210:
211:   ######################################################################
212:   ##### Configuracion de filtro morphGateScale
213:   #####  norm2filter
214:   print(morphGateScale <- 1.5)
215:     morphGate <-norm2Filter(filterId = "MorphologyGate",
216:                "FSC",  "SSC",
217:                scale = morphGateScale )
218:   ######################################################################
219:
220:
221:   ##########################################################################
222:   print("
223:   ##########################################################################
224:   ### Seleccion de positivos en el flowSet original en escala lin
225:   ##########################################################################
226:   ##########################################################################")
227:   ###filtramos primero con rectGate
228:   ##########################################################################
229:   frcfs <- filter(flowData.lin.N, rectGate)
230:   rcPos.flowData.lin.N <-Subset(flowData.lin.N, rectGate)
231:   if (ifDebug){
232:     prepareOutput(mult=0.8)
233:     print(xyplot(SSC ~ FSC , flowData.lin.N,smooth=TRUE, filter=rectGate ,
234:       layout=c(6,1), subset = (CTox == 0), xlab="",
235:       scales = list(x = list(axs = "i"), y = list(draw = FALSE)),
236:       main=paste("Seleccion- minFSC:", minFSC,
237:                " maxFSC:", maxFSC,
238:                " minSSC:", minSSC,
239:                " maxSSC:", maxSSC)))
240:       }
241:   ### Seleccion de la población con mayor densidad.
242:   ### con un filtro norm2Filter
243:  ##########################################################################
244:   fPosfs <- filter(rcPos.flowData.lin.N , morphGate )
245:   Pos.flowData.lin.N  <-Subset(rcPos.flowData.lin.N , morphGate )
246:   ## El flowSet original con la seleccion
247:   ## >>> rectangular rcPos.flowData.lin.N
248:   ## >>> norm2Filter Pos.flowData.lin.N
249:   if (ifDebug){
250:     prepareOutput(mult=0.8)
251:     print(xyplot(SSC ~ FSC , rcPos.flowData.lin.N,smooth=TRUE,
252:       filter=morphGate , layout=c(6,1), subset = (CTox == 0), xlab="",
253:       scales = list(x = list(axs = "i"), y = list(draw = FALSE)),
254:       main=paste("Seleccion- morphGateScale <- ", morphGateScale )))
255:       }
256:
257:
258:   ##########################################################################
259:   print("
260:   ##########################################################################
261:   ### Seleccion de positivos en el flowSet original en escala logaritmica
262:   ##########################################################################
263:   ##########################################################################")
264:   ###filtramos primero con rectGate
265:   ##########################################################################
266:   frcfs <- filter(flowData.orig.N, rectGate)
267:   rcPos.flowData.orig.N <-Subset(flowData.orig.N, rectGate)
268:   if (ifDebug){
269:     prepareOutput(mult=0.8)
270:     print(xyplot(SSC ~ FSC , flowData.orig.N,smooth=TRUE, filter=rectGate ,
271:       layout=c(6,1), subset = (CTox == 0), xlab="",
272:       scales = list(x = list(axs = "i"), y = list(draw = FALSE)),
```

```
273:       main=paste("Seleccion- minFSC:", minFSC,
274:                  " maxFSC:", maxFSC,
275:                  " minSSC:", minSSC,
276:                  " maxSSC:", maxSSC)))
277:       }
278:   ### Seleccion de la población con mayor densidad.
279:   ### con un filtro norm2Filter
280:        ##############################################################################
281:   fPosfs <- filter(rcPos.flowData.lin.N , morphGate )
282:   Pos.flowData.orig.N  <-Subset(rcPos.flowData.orig.N , morphGate )
283:   ## El flowSet original con la seleccion
284:   ## >>> rectangular rcPos.flowData.orig.N
285:   ## >>> norm2Filter Pos.flowData.orig.N
286:   if (ifDebug){
287:     prepareOutput(mult=0.8)
288:     print(xyplot(SSC ~ FSC , rcPos.flowData.orig.N, smooth=TRUE,
289:       filter=morphGate , layout=c(6,1), subset = (CTox == 0), xlab="",
290:       scales = list(x = list(axs = "i"), y = list(draw = FALSE)),
291:       main=paste("Seleccion- morphGateScale <- ", morphGateScale )))
292:       }
293:
294:   par.pos.orig <- viewPars(Pos.flowData.orig.N)
295:   par.pos.lin <- viewPars(Pos.flowData.lin.N)
296:
297:
298:   print("
299:   ###################################################################
300:   ### chunk 7: ### Cálculo de las estadisticas de selección
301:   ###################################################################")
302:
303:   ##############################################################################
304:
305:   Total <-as.numeric(fsApply(flowData.orig.N, nrow, use.exprs = TRUE))
306:   Pos <-as.numeric(fsApply(Pos.flowData.orig.N, nrow, use.exprs = TRUE))
307:   data1 <-data.frame(Files = Tclist, "Total Cells" = Total, "Live Cells" = Pos )
308:   data1 <-transform(data1, Percent = data1[, 3] * 100/data1[,2])
309:   EstatSel <-as.matrix(data1)
310:   EstatSel
311:
312:    ##############################################################################
313:    print("##Guardamos a disco la matriz de estadisticas")
314:    txtFilename <- paste(dataDirOut,"/" , ExpList ,
315:                         ".EstadisticasSeleccion", ".txt", sep="")
316:
317:    write.table(EstatSel ,file=txtFilename ,
318:                row.names=TRUE,sep="\t",eol="\n",dec = ",")
319:    print(txtFilename )
320:    #ForDebug panelFile <- 1
321:    if (panelFile ==1){FinEstadisticasSeleccion <- EstatSel } else {
322:        FinEstadisticasSeleccion  <- rbind(FinEstadisticasSeleccion ,EstatSel)}
323:    ##############################################################################
324:
325:   ################## Comprobaciones ################################
326:   if (ifDebug){
327:
328:   ############################################################
329:   ## Comprobar los maximos y minimos
330:   ## A maxima resolucion
331:     fDensityTest(Pos.flowData.lin.N[1:6], texto="Pos.flowData.lin.N[1:6]",
332:       iminVs=-200, imaxVs=1000,factorAg="CTox",
333:       intMax=0.025)
334:   fDensityTest(flowData.lin.N[1:6], texto="rcPosTFS", iminVs=0,
335:                imaxVs=1000,factorAg="CTox", intMax=0.04,isXlog=TRUE)
336:
337:   fDensityTest(Pos.flowData.lin.N[1:6], texto="rcPosTFS", iminVs=0,
338:                imaxVs=1000,factorAg="CTox", intMax=0.04,isXlog=TRUE)
339:
340:   fDensityTest(Pos.flowData.lin.N[1:6], texto="rcPosTFS", iminVs=0,
```

```
341:                imaxVs=100,factorAg="CTox", intMax=0.2,isXlog=TRUE )
342:     }
343:   ################## fin Comprobaciones ###################################
344:
345:
346:   ########################################################################
347:   #####Medias y medianas de los frames  en escala lineal ##############
348:   ##############################################################
349:   #Calculamos las medianas de los experimentos
350:
351:   TMedians.lin.N <- NULL
352:   data.TMedians.lin.N<- NULL
353:   MediansData.TMedians.lin.N<- NULL
354:
355:   TMeans.lin.N<- NULL
356:   Data.TMedians.lin.N <- NULL
357:   MediansData.lin.N<- NULL
358:   ########## Cálculo de medianas ###################################
359:   TMedians.lin.N<-as.matrix(fsApply(Pos.flowData.lin.N, each_col, median))
360:   TMedians.lin.N
361:   data.TMedians.lin.N<-data.frame(Files = Tclist,
362:                       "FSC" = TMedians.lin.N[, 1],
363:              "SSC" = TMedians.lin.N[, 2],
364:              "FL1" = TMedians.lin.N[, 3],
365:              "FL3" = TMedians.lin.N[, 4],
366:              "FL6" = TMedians.lin.N[, 5],
367:              "FL8" = TMedians.lin.N[, 6])
368:
369:   MediansData.lin.N<- cbind(data.TMedians.lin.N,factores)
370:
371:   if (panelFile ==1){ finMediansData.lin.N<- MediansData.lin.N} else {
372:   finMediansData.lin.N<- rbind(finMediansData.lin.N,MediansData.lin.N)}
373:
374:    print("##Guardamos a disco la matriz de medianas")
375:    txtFilename <- paste(dataDirOut,"/" , ExpList ,
376:                         ".MatrizMedianasOriginaleslin.", ".txt", sep="")
377:
378:    write.table(MediansData.lin.N ,file=txtFilename ,row.names=TRUE,
379:                sep="\t",eol="\n",dec = ",")
380:
381:
382:   ########## Cálculo de medias ###################################
383:   TMeans.lin.N<-as.matrix(fsApply(Pos.flowData.lin.N, each_col, mean))
384:   TMeans.lin.N
385:   data.TMeans.lin.N<-data.frame(Files = Tclist,
386:                       "FSC" = TMeans.lin.N[, 1],
387:              "SSC" = TMeans.lin.N[, 2],
388:              "FL1" = TMeans.lin.N[, 3],
389:              "FL3" = TMeans.lin.N[, 4],
390:              "FL6" = TMeans.lin.N[, 5],
391:              "FL8" = TMeans.lin.N[, 6])
392:
393:   MeansData.lin.N<- cbind(data.TMeans.lin.N,factores)
394:
395:   if (panelFile ==1){ finMeansData.lin.N<- MeansData.lin.N} else {
396:   finMeansData.lin.N<- rbind(finMeansData.lin.N,MeansData.lin.N)}
397:
398:
399:    print("##Guardamos a disco la matriz de medias")
400:    txtFilename <- paste(dataDirOut,"/" , ExpList ,
401:                         ".MatrizMediasOriginaleslin.", ".txt", sep="")
402:    write.table(MeansData.lin.N ,file=txtFilename ,row.names=TRUE,sep="\t",
403:                eol="\r\n",dec = ",")
404:   ##############################################################################
405:
406:
407:
408:
```

```
409:  ##############################################################################
410:  #Medias y medianas de los frames  en escala original (log)")
411:  ############################################################################
412:  #Calculamos las medianas de los experimentos")
413:
414:  TMedians.orig.N <- NULL
415:  data.TMedians.orig.N<- NULL
416:  MediansData.TMedians.orig.N<- NULL
417:
418:  TMeans.orig.N<- NULL
419:  Data.TMedians.orig.N <- NULL
420:  MediansData.orig.N<- NULL
421:  ########## Cálculo de medianas ###################################
422:  TMedians.orig.N<-as.matrix(fsApply(Pos.flowData.orig.N, each_col, median))
423:  TMedians.orig.N
424:  data.TMedians.orig.N<-data.frame(Files = Tclist,
425:                      "FSC" = TMedians.orig.N[, 1],
426:             "SSC" = TMedians.orig.N[, 2],
427:             "FL1" = TMedians.orig.N[, 3],
428:             "FL3" = TMedians.orig.N[, 4],
429:             "FL6" = TMedians.orig.N[, 5],
430:             "FL8" = TMedians.orig.N[, 6])
431:
432:  MediansData.orig.N<- cbind(data.TMedians.orig.N,factores)
433:
434:  if (panelFile ==1){ finMediansData.orig.N<- MediansData.orig.N} else {
435:  finMediansData.orig.N<- rbind(finMediansData.orig.N,MediansData.orig.N)}
436:
437:
438:
439:   print("##Guardamos a disco la matriz de medianas")
440:   txtFilename <- paste(dataDirOut,"/" , ExpList ,
441:                  ".MatrizMedianasOriginalesLog.", ".txt", sep="")
442:
443:   write.table(MediansData.orig.N ,file=txtFilename ,row.names=TRUE,
444:               sep="\t",eol="\n",dec = ",")
445:
446:
447:  ########## Cálculo de medias ###################################
448:  TMeans.orig.N<-as.matrix(fsApply(Pos.flowData.orig.N, each_col, mean))
449:  TMeans.orig.N
450:  data.TMeans.orig.N<-data.frame(Files = Tclist,
451:                      "FSC" = TMeans.orig.N[, 1],
452:             "SSC" = TMeans.orig.N[, 2],
453:             "FL1" = TMeans.orig.N[, 3],
454:             "FL3" = TMeans.orig.N[, 4],
455:             "FL6" = TMeans.orig.N[, 5],
456:             "FL8" = TMeans.orig.N[, 6])
457:
458:  MeansData.orig.N<- cbind(data.TMeans.orig.N,factores)
459:
460:  if (panelFile ==1){ finMeansData.orig.N<- MeansData.orig.N} else {
461:  finMeansData.orig.N<- rbind(finMeansData.orig.N,MeansData.orig.N)}
462:
463:
464:   print("##Guardamos a disco la matriz de medias")
465:   txtFilename <- paste(dataDirOut,"/" , ExpList ,
466:                        ".MatrizMediasOriginalesLog.", ".txt", sep="")
467:
468:   write.table(MeansData.orig.N ,file=txtFilename ,row.names=TRUE,
469:               sep="\t",eol="\r\n",dec = ",")
470:
471:  ##############################################################################
472:
473:
474:  print("
475:  ##############################################################################
476:  ### chunk 8: CALCULO DE LA MATRIZ DE COMPENSACION
```

```
477:   ##############################################################################
478:   # El calculo se raliza con las celulas seleccionadas en eslala lineal##")
479:
480:
481:   ###############################################################################
482:   ## Separamos en un flowSet los controles de compensacion. FiltFS
483:   ###############################################################################
484:    PreFiltFS = flowSet(Pos.flowData.lin.N[[1]],
485:         Pos.flowData.lin.N[[2]],
486:         Pos.flowData.lin.N[[3]],
487:         Pos.flowData.lin.N[[4]],
488:         Pos.flowData.lin.N[[5]])
489:   ControlsNames <- sampleNames(flowData[1:5])
490:       sampleNames(PreFiltFS )<-ControlsNames
491:
492:   summary(PreFiltFS)
493:
    ##################################################################################################
494:
495:
496:
497:   # Los flowframes tienen que estar en el siguiente orden
498:   # 1. unstained control
499:   # 2. single stain for first column after SSC
500:   # 3. single stain for second column after SSC
501:   # etc
502:   ## Esto es unicamente necesario si hay dos poblaciones
503:   ## Separamos la poblacion alta con kmeansFilter
504:
505:   kmfilt1 <-kmeansFilter("FL1" = c("Low", "High"))
506:   filtrados1<-filter(PreFiltFS[[2]],kmfilt1)
507:   FL1.high<-split(PreFiltFS[[2]], filtrados1)$`High`
508:
509:
510:   kmfilt2 <-kmeansFilter("FL3" = c("Low", "High"))
511:   filtrados2<-filter(PreFiltFS[[3]],kmfilt2)
512:   FL3.high<-split(PreFiltFS[[3]], filtrados2)$`High`
513:
514:
515:   kmfilt3 <-kmeansFilter("FL6" = c("Low", "High"))
516:   filtrados3<-filter(PreFiltFS[[4]],kmfilt3)
517:   FL6.high<-split(PreFiltFS[[4]], filtrados3)$`High`
518:
519:
520:   kmfilt4 <-kmeansFilter("FL8" = c("Low", "High"))
521:   filtrados4<-filter(PreFiltFS[[5]],kmfilt4)
522:   FL8.high<-split(PreFiltFS[[5]], filtrados4)$`High`
523:
524:   FL1.high
525:   FL3.high
526:   FL6.high
527:   FL8.high
528:
529:   # Combinamos los resultados en un nuevo flowSet
530:   FiltFS = flowSet(PreFiltFS[[1]],
531:       FL1.high,
532:       FL3.high,
533:       FL6.high,
534:       FL8.high)
535:   FiltFS
536:
537:   sampleNames(FiltFS) <- sampleNames(PreFiltFS[1:5])
538:
539:   summary(FiltFS)
540:
541:   ## Si no se quiere usar la poblacion alta descomentar
542:   #FiltFS <- PreFiltFS[1:5]
543:
```

```
544:   ##############################################################################
545:   #Calculamos la intensidad de fondo para cada parametro
546:   #para la sensibilidad que tenemos los valores de intensidad de fondo
547:   # del control blanco son muy bajos¡¡¡¡. ")
548:   CMed = as.matrix(fsApply(FiltFS, each_col, median)[, -c(1:2)])
549:   CMed
550:
551:
552:   print("##Guardamos a disco la matriz de intensidad de fondo")
553:   txtFilename <- paste(dataDirOut,"/" , ExpList ,
554:                        ".MatrizIntensidadFondo", ".txt", sep="")
555:
556:   write.table(CMed  ,file=txtFilename ,row.names=TRUE,sep="\t",
557:               eol="\r\n",dec = ",")
558:   print(txtFilename )
559:
560:   ##############################################################################
561:   #Desplazar las medianas de cada experimento respecto del control
562:   # con menor intensidad de fondo
563:   bFiltFS <-transform(FiltFS,
564:             "FL1" = `FL1` -min(CMed [,1]),
565:         "FL3" = `FL3` -min(CMed [,2]),
566:         "FL6" = `FL6` -min(CMed [,3]),
567:         "FL8" = `FL8` -min(CMed [,4]))
568:   sampleNames(bFiltFS)
569:   summary(bFiltFS)
570:
571:   if (ifDebug){
572:
573:   ###### Comprobar los histogramas en escala lineal
574:   VisualizarHistogramas(bFiltFS [[1]],main=paste("bPos - Lineal",Titulos[6]),
575:                                                  xmin=-100, xmax=3500)
576:   VisualizarHistogramas(bFiltFS [[2]],main=paste("bPos - Lineal",Titulos[2]),
577:                                                  xmin=-100, xmax=3500)
578:   VisualizarHistogramas(bFiltFS [[3]],main=paste("bPos - Lineal",Titulos[3]),
579:                                                  xmin=-100, xmax=3500)
580:   VisualizarHistogramas(bFiltFS [[4]],main=paste("bPos - Lineal",Titulos[4]),
581:                                                  xmin=-100, xmax=3500)
582:   VisualizarHistogramas(bFiltFS [[5]],main=paste("bPos - Lineal",Titulos[5]),
583:                                                  xmin=-100, xmax=3500)
584:
585:   }
586:
587:
588:
589:
590:   ##############################################################################
591:   #Obtener las medianas de los experimentos single-stained
592:   ##############################################################################
593:   FObs = as.matrix(fsApply(bFiltFS[2:5], each_col, median)[,-c(1:2)])
594:   FObs
595:
    ###############################################################################################
596:   # Estimar la matriz de compensacion
597:     fij = solve(FObs) %*% diag(diag(FObs))
598:     colnames(fij)<- rownames(fij)
599:     print("##### Matriz fij ######")
600:     round(fij,3)
601:     tfij <- t(fij)
602:     print("##### Matriz tfij ######")
603:     round(tfij,3)
604:   ##############################################################################
605:   spillmat <- round(solve(fij),3)
606:   print("##### Matriz spillmat de fij######")
607:   spillmat
608:   ### En realidad la que se utiliza en los citometros es
609:   ### la matriz traspuesta, pero en flowCore se utiliza
610:   ### spillmat
```

```
611:          tspillmat <- t(spillmat)
612:    print("##### Matriz spillmat de fij######")
613:    round(tspillmat,3)
614:
615:
616:    ##############################################################################
617:    ### chunk 9: APLICAR LA MATRIZ DE COMPENSACION
618:    ##############################################################################
619:    ##############################################################################
620:    # compensamos los datos en lineal con spillmat
621:    ##############################################################################
622:
623:    ####cFlowData.lin  contiene todos los flowframe
624:    ####compensados sin filtros en lineal
625:    cFlowData.lin <- compensate(flowData.lin, spillmat)
626:    cFlowData.lin
627:    summary(cFlowData.lin)
628:    viewPars(cFlowData.lin)
629:
630:    ##############################################################################
631:    ####cPos.flowData.lin.N  contiene todos los filtrados compensados en lineal
632:    cPos.flowData.lin.N <- compensate( Pos.flowData.lin.N  , spillmat)
633:    cPos.flowData.lin.N
634:    print(summary(cPos.flowData.lin.N ))
635:
636:
637:
    ##############################################################################################
638:    print("##Guardar a disco la matriz de compensacion")
639:    txtFilename <- paste(dataDirOut,"/" , ExpList
640:                                        ,"MatrizCompensacionfij"
641:                                        , ".txt", sep="")
642:    print(txtFilename)
643:    write.table(as.matrix(round(fij,4)),file=txtFilename
644:                              ,row.names=TRUE,sep="\t",eol="\r\n",dec = ",")
645:    txtFilename <- paste(dataDirOut,"/" , ExpList
646:                                        ,"MatrizCompensacionTras"
647:                                        , ".txt", sep="")
648:      print(txtFilename)
649:    write.table(as.matrix(round(spillmat,4)),file=txtFilename
650:                              ,row.names=TRUE,sep="\t",eol="\r\n",dec = ",")
651:
652:
653:
654:
    ##############################################################################################
655:    ### Uno de los problemas es que se generan muchos valores negativos
656:    ###    ** Los canales positivos y negativos están perfectamente alineados
657:    if (ifDebug){
658:    VisualizarHistogramas( flowData.lin.N[[6]],main=paste("All - Lineal",
659:          Titulos[6]), xmin=-100, xmax=1000)
660:    VisualizarHistogramas( flowData.lin.N[[2]],main=paste("All - Lineal",
661:          Titulos[2]), xmin=-100, xmax=1000)
662:    VisualizarHistogramas( flowData.lin.N[[3]],main=paste("All - Lineal",
663:          Titulos[3]), xmin=-100, xmax=1000)
664:    VisualizarHistogramas( flowData.lin.N[[4]],main=paste("All - Lineal",
665:          Titulos[4]), xmin=-100, xmax=1000)
666:    VisualizarHistogramas( flowData.lin.N[[5]],main=paste("All - Lineal",
667:          Titulos[5]), xmin=-100, xmax=1000)
668:
669:    viewPars(cFlowData.lin)
670:    summary(cFlowData.lin)
671:       ### Las funciónes visualizarDensity y visualizarDensityRec
672:       ### permiten visualizar el grafico de densidad reescalado.
673:
674:      visualizarDensity(flowData.lin.N)
675:      viewPars(flowData.lin.N)
676:      visualizarDensityRec(cFlowData.lin,xlim=c(-500, 2000))
```

```
677:     visualizarDensityRec(cFlowData.lin,xlim=c(-500, 1000))
678:
679:   }
680:
681:
682:
683:
     print("#############################################################################################
684:   print("#Medias y medianas de los frames compensados en escala lineal")
685:
     print("#############################################################################################
686:   print("#Calculamos las medianas de los experimentos")
687:
688:   cTMedians.lin.N <- NULL
689:   cdata.TMedians.lin.N<- NULL
690:   cMediansData.TMedians.lin.N<- NULL
691:
692:   cTMeans.lin.N<- NULL
693:   cData.TMedians.lin.N <- NULL
694:   cMediansData.lin.N<- NULL
695:   ########## Cálculo de medianas ######################################
696:   cTMedians.lin.N<-as.matrix(fsApply(cPos.flowData.lin.N, each_col, median))
697:   cTMedians.lin.N
698:   cdata.TMedians.lin.N<-data.frame(Files = Tclist,
699:                       "FSC" = cTMedians.lin.N[, 1],
700:              "SSC" = cTMedians.lin.N[, 2],
701:              "FL1" = cTMedians.lin.N[, 3],
702:              "FL3" = cTMedians.lin.N[, 4],
703:              "FL6" = cTMedians.lin.N[, 5],
704:              "FL8" = cTMedians.lin.N[, 6])
705:
706:   cMediansData.lin.N<- cbind(cdata.TMedians.lin.N,factores)
707:
708:   if (ifDebug){
709:     ### Datos ordenados por factores
710:     mat <- cMediansData.lin.N
711:     matord <- mat[order(mat[,"StainId"]),]
712:
713:     }
714:
715:   if (panelFile ==1){ cfinMediansData.lin.N<- cMediansData.lin.N} else {
716:   cfinMediansData.lin.N<- rbind(cfinMediansData.lin.N,cMediansData.lin.N)}
717:
718:
719:
720:    print("##Guardamos a disco la matriz de medianas")
721:    txtFilename <- paste(dataDirOut,"/" , ExpList ,".MatrizMedianasCompensadaslin.",
      ".txt", sep="")
722:    write.table(cMediansData.lin.N ,file=txtFilename ,row.names=TRUE,sep="\t",
     eol="\n",dec = ",")
723:    print(txtFilename )
724:
725:
726:   ########## Cálculo de medias ######################################
727:   cTMeans.lin.N<-as.matrix(fsApply(cPos.flowData.lin.N, each_col, mean))
728:   cTMeans.lin.N
729:   cdata.TMeans.lin.N<-data.frame(Files = Tclist,
730:                       "FSC" = cTMeans.lin.N[, 1],
731:              "SSC" = cTMeans.lin.N[, 2],
732:              "FL1" = cTMeans.lin.N[, 3],
733:              "FL3" = cTMeans.lin.N[, 4],
734:              "FL6" = cTMeans.lin.N[, 5],
735:              "FL8" = cTMeans.lin.N[, 6])
736:
737:   cMeansData.lin.N<- cbind(cdata.TMeans.lin.N,factores)
738:
739:   if (panelFile ==1){ cfinMeansData.lin.N<- cMeansData.lin.N} else {
740:   cfinMeansData.lin.N<- rbind(cfinMeansData.lin.N,cMeansData.lin.N)}
```

```
741:
742:
743:    print("##Guardamos a disco la matriz de medias")
744:    txtFilename <- paste(dataDirOut,"/" , ExpList ,".MatrizMediasCompensadaslin.",
    ".txt", sep="")
745:    write.table(cMeansData.lin.N ,file=txtFilename ,row.names=TRUE,sep="\t",
    eol="\r\n",dec = ",")
746:    print(txtFilename )
747:
    ##############################################################################################
748:
749:
750:
751:
752:   ######################################################################
753:   ### chunk 10: ### Transformación de los valores de fluorescencia
754:   ###              linealizados a la escala logaritmica original
755:   ######################################################################
756:
757:   ######################################################################
758:   ### Para devolver los datos a la escala logaritmica ")
759:   ### Hay que tener en cuenta que valor minimo permisible era 1 >>log10(0)...
760:
761:   print("##Medianas en los datos originales en escala lin")
762:     TMediansMinima.antes <- apply(TMedians.lin.N[,-(1:2)],2,min)
763:   print("##Medianas en los datos compensados en escala lin")
764:     TMediansMinima.despues<- apply(cTMedians.lin.N[,-(1:2)],2,min)
765:   ## Hay que sumar el minimo a todos los canales
766:   ## para ponerlos en un margen aceptable
767:
768:   print("## Comprobamos la diferencia de las medianas")
769:     #### ¿Como es la distribucion en los extremos?
770:     print(medias.diff <- TMediansMinima.antes - TMediansMinima.despues)
771:   print("## Rango logaritmico antes compensacion")
772:     viewPars(flowData.orig)
773:   print("## Rango lineal antes compensacion")
774:     viewPars(flowData.lin )
775:   print("## Rango lineal despues compensacion")
776:     viewPars(cFlowData.lin )
777:
778:   ## Reescalamos antes de revertir
779:   summary(cFlowData.lin)
780:   cFlowData.lin.0<- cFlowData.lin
781:   summary(cFlowData.lin.0)
782:   ## Copia de trabajo en cFlowData.lin.0
783:   cFlowData.log.0<- NULL
784:   cFlowData.lin.0<- cFlowData.lin
785:   ## Tomamos como origen el flowframe no normalizado
786:   cFlowData.log.0 <- flowData.orig
787:   viewPars(cFlowData.log.0)
788:
789:   ## Con minScale 0 el reesacalado no es correcto, por lo que como se
790:   ## cita en la bibliografia los datos están en minScale 1
791:
792:   #### generamos una estructura de flowFrame con los parametros anteriores
793:   rescaling(cFlowData.log.0, minScale=1, maxScale=4097)
794:   for (i in 1:length(cFlowData.log.0)){
795:     keyword(cFlowData.log.0[[i]]) <- list ( `$P3E`="4.0,1.0" )
796:     keyword(cFlowData.log.0[[i]]) <- list ( `$P4E`="4.0,1.0" )
797:     keyword(cFlowData.log.0[[i]]) <- list ( `$P5E`="4.0,1.0" )
798:     keyword(cFlowData.log.0[[i]]) <- list ( `$P6E`="4.0,1.0" )   }
799:
800:   viewPars(cFlowData.log.0)
801:   ### Modificamos los valores de expresion para cada frame
802:   rm(listaMatsInit)
803:   rm(listaMatsFinal)
804:   rm(listaMatsLog)
805:   rm(expresionInit)
```

```
806:   listaMatsInit <- as.list(Titulos)
807:
808:   listaMatsFinal<- as.list(Titulos)
809:
810:   listaMatsLog<- as.list(Titulos)
811:
812:
813:   for (i in 1:length(cFlowData.lin.0)){
814:       rm(expresionInit); rm(expresionFinal);rm(expresionLog);
815:       expresionInit <- exprs(cFlowData.lin.0[[i]])[,-c(1:2)]
816:       expresionFinal <- expresionInit
817:       expresionLog <- exprs(flowData.orig[[i]])[,-c(1:2)]
818:       columns<-colnames(expresionInit )
819:       for (j in 1:length(columns)){
820:           int <- round((log10(expresionInit[,j] + medias.diff[j]))*4096/4)
821:           int2<-ifelse(is.nan(int),0,int)
822:           expresionFinal[,j] <- int2
823:           rm(int);rm(int2)
824:         }
825:
826:       listaMatsInit[[sampleNames(cFlowData.lin.0[i])]] <- expresionInit
827:       summary(listaMatsInit[[i]])
828:       expresionFinal <- replace(expresionFinal , expresionFinal<1, 1)
829:       listaMatsFinal[[sampleNames(cFlowData.lin.0[i])]] <- expresionFinal
830:       listaMatsLog[[sampleNames(cFlowData.lin.0[i])]] <-  expresionLog
831:       exprs(cFlowData.log.0[[i]])[,-c(1:2)] <- expresionFinal
832:     }
833:   viewPars(cFlowData.log.0)
834:   if (ifDebug){
835:   visualizarScater(cFlowData.log.0, main="Compensados - Escala log",mult=0.9,
836:                    layout=c(6,3))
837:   scatersComparados(flowData,cFlowData.log.0,mult=1,Texto="ScatersComparados")
838:   }
839:
840:   summary(listaMatsFinal)
841:   ### Eliminar negativos ¡¡¡¡¡¡¡
842:   ######################################################################
843:   ##### Configuracion de filtro rectangular
844:   print(minchan <- 2)
845:   print(maxchan <- 4097)
846:   fullrectGate <- rectangleGate(filterId="fullrectangleGate",
847:            "FSC"=c(minchan , maxchan ),
848:            "SSC"=c(minchan , maxchan ),
849:            "FL1"=c(minchan , maxchan ),
850:            "FL3"=c(minchan , maxchan ),
851:            "FL6"=c(minchan , maxchan ),
852:            "FL8"=c(minchan , maxchan )  )
853:
854:       ## Filtering using fullrectGate
855:       bfres.log.0<- filter(cFlowData.log.0, fullrectGate )
856:       ##  FlowSet Original sin outliners
857:   cFlowData.log<-Subset(cFlowData.log.0, bfres.log.0)
858:   summary(cFlowData.log)
859:   viewPars(cFlowData.log)
860:
861:   viewPars(cFlowData.log)
862:   summary(cFlowData.log)
863:   #visualizarDensity(cFlowData.log)
864:
865:   ##############################################################################
866:   # seleccion de densidad para eliminar el debris ####################
867:   ###filtramos primero con rectangleGate
868:
869:   frcPos.cFlowData.log <- filter(cFlowData.log, rectGate )
870:   rcPos.cFlowData.log  <- Subset(cFlowData.log, rectGate )
871:   NegC.cFlowData.log   <- Subset(cFlowData.log, !rectGate )
872:   ###################################
873:   fPos.cFlowData.log <- filter(rcPos.cFlowData.log , morphGate )
```

```
874:   cPos.cFlowData.log <-Subset(rcPos.cFlowData.log , morphGate )
875:   NegMTFS <- Subset(rcPos.cFlowData.log, !morphGate )
876:   summary(cPos.cFlowData.log )
877:   cPosTFS.log <- cPos.cFlowData.log
878:   ### por compatibilidad con los graficos
879:
880:
881:   ##############################################################################
882:   #Medias y medianas de los frames compensados en escala logaritmica")
883:   ##############################################################################
884:   #Calculamos las medianas de los experimentos")
885:
886:   cTMedians.log <- NULL
887:   cdata.TMedians.log<- NULL
888:   cMediansData.TMedians.log<- NULL
889:
890:   cTMeans.log<- NULL
891:   cData.TMedians.log <- NULL
892:   cMediansData.log<- NULL
893:   ########## Cálculo de medianas ######################################
894:   cTMedians.log<-as.matrix(fsApply(cPos.cFlowData.log, each_col, median))
895:   cTMedians.log
896:   cdata.TMedians.log<-data.frame(Files = Tclist,
897:                       "FSC" = cTMedians.log[, 1],
898:              "SSC" = cTMedians.log[, 2],
899:              "FL1" = cTMedians.log[, 3],
900:              "FL3" = cTMedians.log[, 4],
901:              "FL6" = cTMedians.log[, 5],
902:              "FL8" = cTMedians.log[, 6])
903:
904:   cMediansData.log<- cbind(cdata.TMedians.log,factores)
905:
906:   if (panelFile ==1){ cfinMediansData.log<- cMediansData.log} else {
907:   cfinMediansData.log<- rbind(cfinMediansData.log,cMediansData.log)}
908:
909:
910:
911:    print("##Guardamos a disco la matriz de medianas")
912:    txtFilename <- paste(dataDirOut,"/" , ExpList ,
913:                         ".MatrizMedianasCompensadasLog.", ".txt", sep="")
914:    write.table(cMediansData.log,file=txtFilename ,row.names=TRUE,
915:                sep="\t",eol="\n",dec = ",")
916:
917:
918:
919:   ########## Cálculo de medias ######################################
920:   cTMeans.log<-as.matrix(fsApply(cPos.cFlowData.log, each_col, mean))
921:   cTMeans.log
922:   cdata.TMeans.log<-data.frame(Files = Tclist,
923:                       "FSC" = cTMeans.log[, 1],
924:              "SSC" = cTMeans.log[, 2],
925:              "FL1" = cTMeans.log[, 3],
926:              "FL3" = cTMeans.log[, 4],
927:              "FL6" = cTMeans.log[, 5],
928:              "FL8" = cTMeans.log[, 6])
929:
930:   cMeansData.log<- cbind(cdata.TMeans.log,factores)
931:
932:   if (panelFile ==1){ cfinMeansData.log<- cMeansData.log} else {
933:   cfinMeansData.log<- rbind(cfinMeansData.log,cMeansData.log)}
934:
935:
936:    print("##Guardamos a disco la matriz de medias")
937:    txtFilename <- paste(dataDirOut,"/" , ExpList ,
938:                         ".MatrizMediasCompensadasLog.", ".txt", sep="")
939:    write.table(cMeansData.log ,file=txtFilename ,row.names=TRUE,sep="\t",
940:                    eol="\r\n",dec = ",")
941:   ##############################################################################
```

```
942:
943:
944:
945:   ## Test para comprobar los margenes de compensación
946:   ## Basicamente interesa conocer como se desdobla la compensación sobre FL3
947:   ## el panel 1..... ( es del 37%)
948:   if (ifDebug){
949:
950:     ##Los datos en el rango lineal son:
951:     ## listaMatsInit es una lista con los 16 exp.
952:     ## cada elemento de la lista es una matriz
953:     ## con los valores de fluorescencia en lineal
954:     names(listaMatsInit) <- Titulos
955:     ## el elemento 1 es el blanco
956:     UnstainedControl<- listaMatsInit[[1]]
957:     head(UnstainedControl)
958:     UnsData <- UnstainedControl[,c("FL1", "FL3")]
959:     head(UnsData )
960:     UnsData <- cbind(UnstainedControl[,"FL1"],UnstainedControl[,"FL3"])
961:     head(UnsData )
962:     head(UnstainedControl)
963:     FL1Stain<-listaMatsInit[[2]]
964:     FL1Data <- cbind(FL1Stain[,"FL1"],FL1Stain[,"FL3"])
965:     head(FL1Data )
966:     FL3Stain<-listaMatsInit[[3]]
967:     FL3Data <- cbind(FL3Stain[,"FL1"],FL3Stain[,"FL3"])
968:
969:     ## Suponemos que FL3 = Base + FL1*(%Comp)
970:     spillmat["FL1","FL3"]
971:     ## Rango de datos lineal en FL1
972:     maxrange <- max(FL1Data[,1] )
973:     minrange <- min(FL1Data[,1] )
974:     funcComp <- function(x) {x*spillmat["FL1","FL3"]}
975:     funcCompLog <- function(x) {round(log10(x)*4096/4)}
976:
977:   x11()
978:   plot(funcCompLog , xlim=c(1,4500), ylim=c(1,4500))
979:
980:
981:   plimx<- c(0,10)
982:   plimy<- c(-10,200)
983:   X11(width = 15, height = 10, pointsize = 20)
984:
985:     scaterX<- function(xframe){
986:         smoothScatter(xframe, xlim=plimx, ylim=plimy)
987:           abline(v=median(xframe[,1]), col="red")
988:         abline(h=median(xframe[,2]), col="green")
989:       }
990:
991:
992:   scaterX(UnsData )
993:   points(FL3Data , col="black", cex=1.4, pch=".")
994:   points(FL1Data , col="green", cex=1.4, pch=".")
995:   curve(funcComp , col="red", cex=3,xlim=plimx, ylim=plimy, add=TRUE)
996:   curve(funcCompLog , col="red", cex=3,xlim=plimx, ylim=plimy, add=TRUE)
997:
998:
999:   }
1000:
1001:
1002:  ######################################################################
1003:  ### chunk 11: ### Transformación de los valores de fluorescencia
1004:  ###             a escala LinLog
1005:  ## aplicamos la transformacion linlog sobre los datos lineales
1006:  ## factor deamplificacion para mejorar la visualizacion
1007:
1008:  ### testeamos la mejor trasformacion
1009:
```

```
1010:   if (ifDebug){
1011:     ### medianas en lineal
1012:   # TMediansMinima.despues
1013:   #> TMediansMinima.despues["FL1"]
1014:   #     FL1
1015:   #1.168559
1016:   ### esta trasnsformacion es más o menos equivalente a la del citometro
1017:   lnlgT <- linlogTransform(transformationId = "splitscale",
1018:            median =TMediansMinima.despues["FL1"], dist = 3000)
1019:
1020:
1021:   cFlowData.linlogt<- transform(cFlowData.lin,
1022:         "FL1" = (lnlgT(FL1)*preAmpFactor) + scalefactor ,
1023:         "FL3" = (lnlgT(FL3)*preAmpFactor) + scalefactor ,
1024:         "FL6" = (lnlgT(FL6)*preAmpFactor) + scalefactor ,
1025:         "FL8" = (lnlgT(FL8)*preAmpFactor) + scalefactor )
1026:   visualizarDensity(cFlowData.linlogt)
1027:   visualizarScater(cFlowData.linlogt,mult=0.9, layout=c(6,3))
1028:         }
1029:
1030:   preAmpFactor <- 1
1031:   scalefactor <- 0
1032:
1033:   lnlgT <- linlogTransform(transformationId = "splitscale",
1034:            median =1, dist = 1)
1035:
1036:
1037:   cFlowData.linlog.0<- transform(cFlowData.lin,
1038:         "FL1" = (lnlgT(FL1)*preAmpFactor) + scalefactor ,
1039:         "FL3" = (lnlgT(FL3)*preAmpFactor) + scalefactor ,
1040:         "FL6" = (lnlgT(FL6)*preAmpFactor) + scalefactor ,
1041:         "FL8" = (lnlgT(FL8)*preAmpFactor) + scalefactor )
1042:
1043:   # cFlowData.linlog es el flowset compensado y en escala
1044:   # linlog ###################################################
1045:   summary(cFlowData.linlog.0)
1046:   viewPars(cFlowData.linlog.0)
1047:
1048:     if (ifDebug){
1049:       visualizarDensity(cFlowData.linlog.0)
1050:       VisualizarHistogramas(cFlowData.linlog.0[[6]],
1051:           main=paste("All - linlog- comp",Titulos[6]), xmin=-1, xmax=1)
1052:       VisualizarHistogramas(cFlowData.linlog.0[[2]],
1053:           main=paste("All - linlog- comp",Titulos[2]), xmin=-1, xmax=1)
1054:       VisualizarHistogramas(cFlowData.linlog.0[[3]],
1055:           main=paste("All - linlog- comp",Titulos[3]), xmin=-1, xmax=1)
1056:       VisualizarHistogramas(cFlowData.linlog.0[[4]],
1057:           main=paste("All - linlog- comp",Titulos[4]), xmin=-1, xmax=1)
1058:       VisualizarHistogramas(cFlowData.linlog.0[[5]],
1059:           main=paste("All - linlog- comp",Titulos[5]), xmin=-1, xmax=1)
1060:     }
1061:   ##########################################################################
1062:   # seleccion de densidad para eliminar el debris ####################
1063:
1064:   ###Primero eliminar los negativos
1065:
1066:   ##### Configuracion de filtro rectangular
1067:   print(minchan <- 0)
1068:   print(maxchan <- 4097)
1069:   fullrectGate <- rectangleGate(filterId="fullrectangleGate",
1070:            "FSC"=c(minchan , maxchan ),
1071:            "SSC"=c(minchan , maxchan ),
1072:            "FL1"=c(minchan , maxchan ),
1073:            "FL3"=c(minchan , maxchan ),
1074:            "FL6"=c(minchan , maxchan ),
1075:            "FL8"=c(minchan , maxchan )  )
1076:
1077:       ## Filtering using fullrectGate
```

```
1078:       bfres.linlog.0<- filter(cFlowData.linlog.0, fullrectGate )
1079:       ##  FlowSet Original sin outliners
1080:   cFlowData.linlog<-Subset(cFlowData.linlog.0,  bfres.linlog.0)
1081:   summary(cFlowData.linlog)
1082:   viewPars(cFlowData.linlog)
1083:   if (ifDebug){
1084:       visualizarDensity(cFlowData.linlog)
1085:     }
1086:
1087:   ###filtrar con rectangleGate
1088:   frcPos.cFlowData.linlog <- filter(cFlowData.linlog, rectGate )
1089:   rcPos.cFlowData.linlog  <- Subset(cFlowData.linlog, rectGate )
1090:   cNegCTFS.cFlowData.linlog   <- Subset(cFlowData.linlog, !rectGate )
1091:
1092:   ###Filtrar con morphGate
1093:   fPos.rcPosTFS.cFlowData.linlog <- filter(rcPos.cFlowData.linlog, morphGate )
1094:   cPos.cFlowData.linlog <-Subset(rcPos.cFlowData.linlog, morphGate )
1095:   cNeg.cFlowData.linlog <- Subset(rcPos.cFlowData.linlog, !morphGate )
1096:   summary(cPos.cFlowData.linlog)
1097:
1098:   ##############################################################################
1099:
1100:   if (ifDebug){
1101:   #### Sisualizar el resultado de diferente forma
1102:   visualizarDensity(cPos.cFlowData.linlog)
1103:
1104:   visualizarScater(cFlowData.linlog,
1105:             main="Visualizacion vivas. LinLog",mult=1,
1106:             layout=c(6,4))
1107:
1108:   visualizarScatervivas.FS.SC(cFlowData.linlog,
1109:             cPos.cFlowData.linlog,
1110:             main="Visualizacion vivas. LinLog",mult=1,
1111:             my.layout=c(6,4))
1112:
1113:   visualizarScatervivas.FL(cFlowData.linlog,
1114:             cPos.cFlowData.linlog,
1115:             main="Visualizacion vivas. LinLog",mult=1,
1116:             my.layout=c(6,4)) }
1117:
1118:
1119:
1120:
1121:
1122:   ############################################################
1123:   ### chunk 12: SALVAMOS EL FLOWSET COMPENSADO           ##
1124:   ############################################################
1125:
1126:
1127:   ##Id tiene el nombre del fichero fcs original
1128:   ficheros <- pData(flowData)$Id
1129:   ficherosC.lin <- paste("CompMedians.lin",".fcs",ficheros,".fcs", sep="" )
1130:   ficherosC.log <- paste("CompMedians.log",".fcs",ficheros,".fcs", sep="" )
1131:   ficherosC.linlog <- paste("CompMedians.linlog",ficheros,".fcs", sep="" )
1132:   ##FicherosC son los nombres de los ficheros Compensados
1133:
1134:   dataDirOut.lin<-paste(dataDirOut, "/outComp.lin", sep="")
1135:   dataDirOut.log<-paste(dataDirOut, "/outComp.log", sep="")
1136:   dataDirOut.linlog<-paste(dataDirOut, "/outComp.linlog", sep="")
1137:   dir.create(dataDirOut.lin)
1138:   dir.create(dataDirOut.log)
1139:   dir.create(dataDirOut.linlog)
1140:
1141:   write.flowSet(cFlowData.lin, outdir=dataDirOut.lin,filename=ficherosC.lin)
1142:
1143:   write.flowSet(cFlowData.linlog, outdir=dataDirOut.linlog,
1144:                 filename=ficherosC.linlog)
1145:   #### Se pueden guaradar uno a uno .... sino flowCore los linealiza ¡¡¡¡¡
```

```
1146:   for (fileF in 1:length(ficherosC.log)) {
1147:     write.FCS(cFlowData.log[[fileF]],
1148:         paste(dataDirOut.log,"/",ficherosC.log[fileF], sep=""), what="integer")
1149:
1150:       }
1151:
1152:
1153:   ##########################################################
1154:   ### chunk : Guardamos a disco la matriz de anotacion ##
1155:   ##########################################################
1156:
1157:
1158:   modExpList.lin <- factores
1159:   modExpList.log <- factores
1160:   modExpList.linlog <- factores
1161:   rownames(modExpList.lin)<- ficherosC.lin
1162:   rownames(modExpList.log)<- ficherosC.log
1163:   rownames(modExpList.linlog)<- ficherosC.linlog
1164:
1165:   txtFilename.lin <- paste(dataDirOut.lin , "/annot.",ExpList , sep="")
1166:   txtFilename.log <- paste(dataDirOut.log , "/annot.",ExpList , sep="")
1167:   txtFilename.linlog <- paste(dataDirOut.linlog , "/annot.",ExpList , sep="")
1168:
1169:   write.table(modExpList.lin,file=txtFilename.lin ,row.names=TRUE,
1170:                          sep="\t",eol="\n",dec = ",",quote=FALSE)
1171:
1172:   write.table(modExpList.log,file=txtFilename.log ,row.names=TRUE,
1173:                          sep="\t",eol="\n",dec = ",",quote=FALSE)
1174:
1175:   write.table(modExpList.linlog,file=txtFilename.linlog ,row.names=TRUE,
1176:                          sep="\t",eol="\n",dec = ",",quote=FALSE)
1177:
1178:
1179:   ##############################################################################
1180:
1181:   if (ifDebug){
1182:
1183:   ########################################################################
1184:   ### chunk : Funciones mejoradas para visualizacion de positivos ##
1185:   ########################################################################
1186:
1187:   x11()
1188:   fnvisualizarScatervivasScreen (cFlowData.log, fun= FL3 ~ FL1 ,
1189:                  cPos.cFlowData.log, main="", mult=0.7,
1190:                  my.layout=c(6,4),
1191:                  filename="Rplot%d.png",
1192:                  imageW=21, imageH=29.7,
1193:                  my.scales= list(x = list(axs = "i", cex=0.5),
1194:                  y = list(axs = "i", cex=0.5)),
1195:                  my.colorsSup= colorRampPalette(heat.colors(10)),
1196:                  my.colorsInf= colorRampPalette(Oranges)
1197:                  )
1198:
1199:   x11()
1200:   fnvisualizarScatervivasScreen (flowData.orig, fun= FL3 ~ FL1 ,
1201:                  Pos.flowData.orig.N, main="", mult=0.7,
1202:                  my.layout=c(6,4),
1203:                  filename="Rplot%d.png",
1204:                  imageW=21, imageH=29.7, xlab="",
1205:                  my.scales= list(x = list(axs = "i", cex=0.5),
1206:                  y = list(axs = "i", cex=0.5)),
1207:                  my.colorsSup= colorRampPalette(heat.colors(10)),
1208:                  my.colorsInf= colorRampPalette(Oranges)
1209:                  )
1210:
1211:
1212:
1213:
```

```
1214:
1215:   x11()
1216:   fnvisualizarScatervivasScreen (cFlowData.linlog, fun= FL3 ~ FL1 ,
1217:                   cPos.cFlowData.linlog, main="", mult=0.7,
1218:                  my.layout=c(4,6),
1219:                  filename="Rplot%d.png",
1220:                  imageW=21, imageH=29.7,
1221:                  my.scales= list(x = list(axs = "i", cex=0.5),
1222:                  y = list(axs = "i", cex=0.5)),
1223:                  my.colorsSup= colorRampPalette(heat.colors(9)),
1224:                  my.colorsInf= colorRampPalette(Oranges)
1225:
1226:                  )
1227:
1228:
1229: ############################################################
1230: ### chunk : VISUALIZACION DE RESULTADOS ##
1231: ############################################################
1232:
1233:
1234:
1235:
1236: # Genera los ficheros con los scatters
1237: if (VisualizarResultado =="S") {
1238:
1239:
1240:
1241:   ######################GRAFICOS DE SELECCION#############################
1242:   pdfName<- paste(dataDirOut,"/" , ExpList , "- CompSeleccion%03d.pdf", sep="")
1243:
1244:   pdf(file=pdfName, onefile=TRUE, width = 10, height=12 )
1245:   trellis.par <- NULL
1246:   trellis.par.set(theme = col.whitebg())
1247:   lw <- list(ylab.axis.padding = list(x = 0.4), left.padding = list(x = 0.4,
1248:     units = "inches"), right.padding = list(x = 0, units = "inches"),
1249:     panel = list(x = 1.1, units = "inches"))
1250:   lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
1251:     list(x = 0, units = "inches"), panel = list(x = 1.1, units = "inches"))
1252:   lattice.options(layout.widths = lw, layout.heights = lh,las=2 )
1253:
1254:
1255:   print( xyplot(SSC ~ FSC, flowData.orig,smooth=TRUE, filter=rectGate ,
1256:           layout=my.layout,npoints=1000,cex=0.3,
1257:
1258:       main=paste("Seleccion Vivas Compensacion - minFSC:", minFSC,
1259:                  " maxFSC:", maxFSC,
1260:                  " minSSC:", minSSC,
1261:                  " maxSSC:", maxSSC)))
1262:
1263:   print( xyplot(SSC ~ FSC, rcPos.flowData.orig.N ,smooth=TRUE,
1264:       filter=morphGate,layout=my.layout,npoints=1000,
1265:       main=paste( "Seleccion Vivas Compensacion - morphGateScale:",
1266:       morphGateScale) ))
1267:
1268:   MediansData <- cMediansData.log
1269:   print( xyplot( FL1 ~ FSC , cFlowData.log, smooth=TRUE,  npoints=1000, cex=0.3,
1270:     main=paste( "Compensados en Log - Vivas:", morphGateScale) ,
1271:     ylab= panel.desc[[panelFile ]]["FL1"],
1272:     layout=my.layout,
1273:         panel = function(x, frames, channel.x, channel.y, ...) {
1274:             nm <- as.character(x)
1275:             print(nm)
1276:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
1277:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
1278:     my.panel(x, y, frame = frames[[nm]], ...)
1279:     xyVivas <- exprs(cPos.cFlowData.log[[nm ]])
1280:     panel.abline(h=MediansData[nm ,"FL1"], col="blue", alpha=0.8)
1281:     panel.points(  xyVivas [,"FSC"],  xyVivas [,"FL1"] , col="blue"  ,
```

```
1282:     alpha=0.3, pch=".", cex=1)
1283:         }))
1284: 
1285:   print( xyplot(FL3 ~ FSC , cFlowData.log,smooth=TRUE,npoints=1000,cex=0.3,
1286:     main=paste( "Compensados en Log - Vivas:", morphGateScale) ,
1287:     ylab= panel.desc[[panelFile ]]["FL3"],
1288:     layout=my.layout,
1289:         panel = function(x, frames, channel.x, channel.y, ...) {
1290:             nm <- as.character(x)
1291:             print(nm)
1292:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
1293:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
1294:     my.panel(x, y, frame = frames[[nm]], ...)
1295:     xyVivas <- exprs(cPos.cFlowData.log[[nm ]])
1296:     panel.abline(h=MediansData[nm ,"FL3"], col="blue", alpha=0.8)
1297:     panel.points(  xyVivas [,"FSC"],  xyVivas [,"FL3"] , col="blue"  ,
1298:     alpha=0.3, pch=".", cex=1)
1299:          }))
1300: 
1301: 
1302:   print( xyplot(FL6 ~ FSC  , cFlowData.log,smooth=TRUE,npoints=1000,cex=0.3,
1303:     main=paste( "Compensados en Log - Vivas:", morphGateScale) ,
1304:     ylab= panel.desc[[panelFile ]]["FL6"],
1305:     layout=my.layout,
1306:         panel = function(x, frames, channel.x, channel.y, ...) {
1307:             nm <- as.character(x)
1308:             print(nm)
1309:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
1310:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
1311:     my.panel(x, y, frame = frames[[nm]], ...)
1312:     xyVivas <- exprs(cPos.cFlowData.log[[nm ]])
1313:     panel.abline(h=MediansData[nm ,"FL6"], col="blue", alpha=0.8)
1314:     panel.points(  xyVivas [,"FSC"],  xyVivas [,"FL6"] , col="blue" ,
1315:     alpha=0.3, pch=".", cex=1)
1316:          }))
1317: 
1318: 
1319:   print( xyplot(FL8 ~ FSC, cFlowData.log,smooth=TRUE, npoints=1000,cex=0.3,
1320:     main=paste( "Compensados en Log - Vivas:", morphGateScale) ,
1321:     ylab= panel.desc[[panelFile ]]["FL8"],
1322:     layout=my.layout,
1323: 
1324:         panel = function(x, frames, channel.x, channel.y, ...) {
1325:             nm <- as.character(x)
1326:             print(nm)
1327:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
1328:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
1329:     my.panel(x, y, frame = frames[[nm]], ...)
1330:     xyVivas <- exprs(cPos.cFlowData.log[[nm ]])
1331:     panel.abline(h=MediansData[nm ,"FL8"], col="blue", alpha=0.8)
1332:     panel.points(  xyVivas [,"FSC"],  xyVivas [,"FL8"] , col="blue"  ,
1333:     alpha=0.3, pch=".", cex=1)
1334:          }))
1335: 
1336: 
1337:   print( xyplot( FL1 ~ SSC  , cFlowData.log,smooth=TRUE, npoints=1000,cex=0.3,
1338:     main=paste( "Compensados en Log - Vivas:", morphGateScale) ,
1339:     ylab= panel.desc[[panelFile ]]["FL1"],
1340:     layout=my.layout,
1341: 
1342:         panel = function(x, frames, channel.x, channel.y, ...) {
1343:             nm <- as.character(x)
1344:             print(nm)
1345:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
1346:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
1347:     my.panel(x, y, frame = frames[[nm]], ...)
1348:     xyVivas <- exprs(cPos.cFlowData.log[[nm ]])
1349:     panel.abline(h=MediansData[nm ,"FL1"], col="blue", alpha=0.8)
```

```
1350:     panel.points(  xyVivas [,"SSC"],  xyVivas [,"FL1"] , col="blue"  ,
1351:     alpha=0.3, pch=".", cex=1)
1352:          }))
1353:
1354:   print( xyplot(FL3 ~ SSC , cFlowData.log,smooth=TRUE,npoints=1000,cex=0.3,
1355:     main=paste( "Compensados en Log - Vivas:", morphGateScale) ,
1356:     ylab= panel.desc[[panelFile ]]["FL3"],
1357:     layout=my.layout,
1358:
1359:         panel = function(x, frames, channel.x, channel.y, ...) {
1360:             nm <- as.character(x)
1361:             print(nm)
1362:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
1363:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
1364:     my.panel(x, y, frame = frames[[nm]], ...)
1365:     xyVivas <- exprs(cPos.cFlowData.log[[nm ]])
1366:     panel.abline(h=MediansData[nm ,"FL3"], col="blue", alpha=0.8)
1367:     panel.points(  xyVivas [,"SSC"],  xyVivas [,"FL3"] , col="blue"  ,
1368:     alpha=0.3, pch=".", cex=1)
1369:          }))
1370:
1371:
1372:   print( xyplot(FL6 ~ SSC  , cFlowData.log,smooth=TRUE,  npoints=1000,cex=0.3,
1373:     main=paste( "Compensados en Log - Vivas:", morphGateScale) ,
1374:     ylab= panel.desc[[panelFile ]]["FL6"],
1375:     layout=my.layout,
1376:
1377:         panel = function(x, frames, channel.x, channel.y, ...) {
1378:             nm <- as.character(x)
1379:             print(nm)
1380:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
1381:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
1382:     my.panel(x, y, frame = frames[[nm]], ...)
1383:     xyVivas <- exprs(cPos.cFlowData.log[[nm ]])
1384:     panel.abline(h=MediansData[nm ,"FL6"], col="blue", alpha=0.8)
1385:     panel.points(  xyVivas [,"SSC"],  xyVivas [,"FL6"] , col="blue"  ,
1386:     alpha=0.3, pch=".", cex=1)
1387:          }))
1388:
1389:
1390:   print( xyplot(FL8 ~ SSC, cFlowData.log,smooth=TRUE, npoints=1000,cex=0.3,
1391:     main=paste( "Compensados en Log - Vivas:", morphGateScale) ,
1392:     ylab= panel.desc[[panelFile ]]["FL8"],
1393:     layout=my.layout,
1394:         panel = function(x, frames, channel.x, channel.y, ...) {
1395:             nm <- as.character(x)
1396:             print(nm)
1397:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
1398:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
1399:     my.panel(x, y, frame = frames[[nm]], ...)
1400:     xyVivas <- exprs(cPos.cFlowData.log[[nm ]])
1401:     panel.abline(h=MediansData[nm ,"FL8"], col="blue", alpha=0.8)
1402:     panel.points(  xyVivas [,"SSC"],  xyVivas [,"FL8"] , col="blue"  ,
1403:     alpha=0.3, pch=".", cex=1)
1404:          }))
1405:   dev.off()
1406:
1407:       ######################GRAFICOS DE DENSIDAD#############################
1408:   pdfName<- paste(dataDirOut,"/" , ExpList , "- CompDensityMed%03d.pdf", sep="")
1409:   pdf(file=pdfName, onefile=TRUE, width = 15, height=10 )
1410:
1411:   trellis.par.set(theme = col.whitebg())
1412:    lw <- list(ylab.axis.padding = list(x = 0.5), left.padding = list(x = 0.5,
1413:      units = "inches"), right.padding = list(x = 0, units = "inches"),
1414:      panel = list(x = 1.4, units = "inches"))
1415:   lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
1416:      list(x = 0, units = "inches"), panel = list(x = 4, units = "inches"))
1417:
```

```
1418:   lattice.options(layout.widths = lw, layout.heights = lh,las=2)
1419:
1420:
1421:   #X11(width = 14, height = 7, pointsize = 12)
1422:   print(densityplot(~ `SSC` + `FSC` + `FL1` + `FL3` +  `FL6` + `FL8` ,
1423:         flowData.orig , overlap=0.1,
1424:         main=paste("PDF - Original sin filtros - ",
1425:         as.character(ExpList))))
1426:
1427:   #X11(width = 14, height = 7, pointsize = 12)
1428:   print(densityplot(~ `SSC` + `FSC` + `FL1` + `FL3` +  `FL6` + `FL8` ,
1429:         flowData.orig.N , overlap=0.1,
1430:         main=paste("PDF - Original sin filtros. Normalizada - ",
1431:         as.character(ExpList))))
1432:
1433:   #X11(width = 14, height = 7, pointsize = 12)
1434:   print(densityplot(~ `SSC` + `FSC` + `FL1` + `FL3` +  `FL6` + `FL8` ,
1435:         Pos.flowData.orig.N , overlap=0.1,
1436:         main=paste("PDF - Original Vivas. Normalizada - ",
1437:         as.character(ExpList))))
1438:
1439:   #X11(width = 14, height = 7, pointsize = 12)
1440:   print(densityplot(~ `SSC` + `FSC` + `FL1` + `FL3` +  `FL6` + `FL8` ,
1441:         cFlowData.log, overlap=0.1,
1442:         main=paste("PDF - Compensados sin filtros log",
1443:         as.character(ExpList))))
1444:
1445:   #X11(width = 14, height = 7, pointsize = 12)
1446:   print(densityplot(~ `SSC` + `FSC` + `FL1` + `FL3` +  `FL6` + `FL8` ,
1447:         cPos.cFlowData.log, overlap=0.1,
1448:         main=paste("PDF - Compensadas Vivas log",
1449:         as.character(ExpList))))
1450:
1451:   #X11(width = 14, height = 7, pointsize = 12)
1452:   print(densityplot(~ `SSC` + `FSC` + `FL1` + `FL3` +  `FL6` + `FL8` ,
1453:         cFlowData.linlog, overlap=0.1,
1454:         main=paste("PDF - Compensados sin filtros linlog",
1455:         as.character(ExpList))))
1456:
1457:   #X11(width = 14, height = 7, pointsize = 12)
1458:   print(densityplot(~ `SSC` + `FSC` + `FL1` + `FL3` +  `FL6` + `FL8` ,
1459:         cFlowData.lin, overlap=0.1,
1460:         main=paste("PDF - Compensados sin filtros lin",
1461:         as.character(ExpList))))
1462:
1463:   #X11(width = 14, height = 7, pointsize = 12)
1464:   print(densityplot(~ `SSC` + `FSC` + `FL1` + `FL3` +  `FL6` + `FL8` ,
1465:         cPos.cFlowData.linlog , overlap=0.1,
1466:         main=paste("PDF - Compensados Vivas linlog - ",
1467:         as.character(ExpList))))
1468:
1469:
1470:
1471:
1472:   dev.off()
1473:
1474:   ##############################################################################
1475:
1476:
1477:
1478:   ####SCATERS CON MEDIANAS#########################################################
1479:
1480:
1481:   ## para cd rango <- c(2:16)
1482:   ## para hg rango <- c(2:21)
1483:
1484:
1485:
```

```
1486:
1487:   pdfName<- paste(dataDirOut,"/" , ExpList ,
1488:                   "- CompScatersVisFL1FL3%03d.pdf", sep="")
1489:   pdf(file=pdfName, onefile=TRUE, width = 15, height=10 )
1490:   #X11(width = 15, height = 17)
1491:   tp1<- fnvisualizarScatervivasScreen (cFlowData.log[rango],
1492:                  fun=FL3 ~ FL1 ,cPos.cFlowData.log[rango ],
1493:                  main= "Después compensación - log", mult=1.2,
1494:                  my.layout=laycol,
1495:                  filename="Rplot%d.png",
1496:                  imageW=21, imageH=29.7, paneldesc = panel.desc[[panelFile ]],
1497:                  my.scales= list(x = list(axs = "i", cex=0.4),
1498:                  y = list(axs = "i", cex=0.4))
1499:                  )
1500:
1501:   tp2<- fnvisualizarScatervivasScreen (flowData.orig[rango],
1502:                  fun=FL3 ~ FL1 ,Pos.flowData.orig.N[rango ],
1503:                  main="Antes compensación - log", mult=1.2,
1504:                  my.layout=laycol,
1505:                  filename="Rplot%d.png",
1506:                  imageW=21, imageH=29.7,paneldesc = panel.desc[[panelFile ]],
1507:                  my.scales= list(x = list(axs = "i", cex=0.4),
1508:                  y = list(axs = "i", cex=0.4))
1509:                  )
1510:
1511:
1512:     #x11(width = 25, height = 15, pointsize = 12)
1513:       print( plot(tp1, position = c(0.1, 0, 0.5 , 1), more = TRUE))
1514:       print( plot(tp2, position = c(0.5, 0 ,1 , 1), more = FALSE))
1515:
1516:   dev.off()
1517:
1518:
1519:   pdfName<- paste(dataDirOut,"/" , ExpList ,
1520:                   "- CompScatersVisFL1FL6%03d.pdf", sep="")
1521:   pdf(file=pdfName, onefile=TRUE, width = 15, height=10 )
1522:   #X11(width = 15, height = 17)
1523:   tp1<- fnvisualizarScatervivasScreen (cFlowData.log[rango ],
1524:                  fun=FL6 ~ FL1 ,cPos.cFlowData.log[rango ],
1525:                  main= "Después compensación - log", mult=1.2,
1526:                  my.layout=laycol,
1527:                  filename="Rplot%d.png",
1528:                  imageW=21, imageH=29.7, paneldesc = panel.desc[[panelFile ]],
1529:                  my.scales= list(x = list(axs = "i", cex=0.4),
1530:                  y = list(axs = "i", cex=0.4))
1531:                  )
1532:
1533:   tp2<- fnvisualizarScatervivasScreen (flowData.orig[rango ],
1534:                  fun=FL6 ~ FL1 ,Pos.flowData.orig.N[rango ],
1535:                  main="Antes compensación - log", mult=1.2,
1536:                  my.layout=laycol,
1537:                  filename="Rplot%d.png",
1538:                  imageW=21, imageH=29.7, paneldesc = panel.desc[[panelFile ]],
1539:                  my.scales= list(x = list(axs = "i", cex=0.4),
1540:                  y = list(axs = "i", cex=0.4))
1541:                  )
1542:
1543:
1544:     #x11(width = 25, height = 15, pointsize = 12)
1545:       print( plot(tp1, position = c(0.1, 0, 0.5 , 1), more = TRUE))
1546:       print( plot(tp2, position = c(0.5, 0 ,1 , 1), more = FALSE))
1547:
1548:   dev.off()
1549:
1550:
1551:
1552:   pdfName<- paste(dataDirOut,"/" , ExpList ,
1553:                   "- CompScatersVisFL1FL8%03d.pdf", sep="")
```

```
1554:   pdf(file=pdfName, onefile=TRUE, width = 15, height=10 )
1555:   #X11(width = 15, height = 17)
1556:   tp1<- fnvisualizarScatervivasScreen (cFlowData.log[rango ],
1557:                 fun=FL8 ~ FL1 ,cPos.cFlowData.log[rango ],
1558:                 main= "Después compensación - log", mult=1.2,
1559:                 my.layout=laycol,
1560:                 filename="Rplot%d.png",
1561:                 imageW=21, imageH=29.7, paneldesc = panel.desc[[panelFile ]],
1562:                 my.scales= list(x = list(axs = "i", cex=0.4),
1563:                 y = list(axs = "i", cex=0.4))
1564:                 )
1565:
1566:   tp2<- fnvisualizarScatervivasScreen (flowData.orig[rango ],
1567:                 fun=FL8 ~ FL1 ,Pos.flowData.orig.N[rango ],
1568:                 main="Antes compensación - log", mult=1.2,
1569:                 my.layout=laycol,
1570:                 filename="Rplot%d.png",
1571:                 imageW=21, imageH=29.7, paneldesc = panel.desc[[panelFile ]],
1572:                 my.scales= list(x = list(axs = "i", cex=0.4),
1573:                 y = list(axs = "i", cex=0.4))
1574:                 )
1575:
1576:
1577:     #x11(width = 25, height = 15, pointsize = 12)
1578:       print( plot(tp1, position = c(0.1, 0, 0.5 , 1), more = TRUE))
1579:       print( plot(tp2, position = c(0.5, 0 ,1 , 1), more = FALSE))
1580:
1581:   dev.off()
1582:
1583:
1584:
1585:   pdfName<- paste(dataDirOut,"/" , ExpList ,
1586:                   "- CompScatersVisFL3FL6%03d.pdf", sep="")
1587:   pdf(file=pdfName, onefile=TRUE, width = 15, height=10 )
1588:   #X11(width = 15, height = 17)
1589:   tp1<- fnvisualizarScatervivasScreen (cFlowData.log[rango ],
1590:                 fun=FL6 ~ FL3 ,cPos.cFlowData.log[rango ],
1591:                 main= "Después compensación - log", mult=1.2,
1592:                 my.layout=laycol,
1593:                 filename="Rplot%d.png",
1594:                 imageW=21, imageH=29.7, paneldesc = panel.desc[[panelFile ]],
1595:                 my.scales= list(x = list(axs = "i", cex=0.4),
1596:                 y = list(axs = "i", cex=0.4))
1597:                 )
1598:
1599:   tp2<- fnvisualizarScatervivasScreen (flowData.orig[rango ],
1600:                 fun=FL6 ~ FL3 ,Pos.flowData.orig.N[rango ],
1601:                 main="Antes compensación - log", mult=1.2,
1602:                 my.layout=laycol,
1603:                 filename="Rplot%d.png",
1604:                 imageW=21, imageH=29.7, paneldesc = panel.desc[[panelFile ]],
1605:                 my.scales= list(x = list(axs = "i", cex=0.4),
1606:                 y = list(axs = "i", cex=0.4))
1607:                 )
1608:
1609:
1610:     #x11(width = 25, height = 15, pointsize = 12)
1611:       print( plot(tp1, position = c(0.1, 0, 0.5 , 1), more = TRUE))
1612:       print( plot(tp2, position = c(0.5, 0 ,1 , 1), more = FALSE))
1613:
1614:   dev.off()
1615:
1616:
1617:
1618:   pdfName<- paste(dataDirOut,"/" , ExpList ,
1619:                   "- CompScatersVisFL3FL8%03d.pdf", sep="")
1620:   pdf(file=pdfName, onefile=TRUE, width = 15, height=10 )
1621:   #X11(width = 15, height = 17)
```

```
1622:   tp1<- fnvisualizarScatervivasScreen (cFlowData.log[rango ],
1623:                  fun=FL8 ~ FL3 ,cPos.cFlowData.log[rango ],
1624:                  main= "Después compensación - log", mult=1.2,
1625:                 my.layout=laycol,
1626:                 filename="Rplot%d.png",
1627:                 imageW=21, imageH=29.7, paneldesc = panel.desc[[panelFile ]],
1628:                 my.scales= list(x = list(axs = "i", cex=0.4),
1629:                 y = list(axs = "i", cex=0.4))
1630:                 )
1631:
1632:   tp2<- fnvisualizarScatervivasScreen (flowData.orig[rango ],
1633:                  fun=FL8 ~ FL3 ,Pos.flowData.orig.N[rango ],
1634:                  main="Antes compensación - log", mult=1.2,
1635:                 my.layout=laycol,
1636:                 filename="Rplot%d.png",
1637:                 imageW=21, imageH=29.7, paneldesc = panel.desc[[panelFile ]],
1638:                 my.scales= list(x = list(axs = "i", cex=0.4),
1639:                 y = list(axs = "i", cex=0.4))
1640:                 )
1641:
1642:
1643:     #x11(width = 25, height = 15, pointsize = 12)
1644:       print( plot(tp1, position = c(0.1, 0, 0.5 , 1), more = TRUE))
1645:       print( plot(tp2, position = c(0.5, 0 ,1 , 1), more = FALSE))
1646:
1647:   dev.off()
1648:
1649:
1650:
1651:
1652:   pdfName<- paste(dataDirOut,"/" , ExpList ,
1653:                  "- CompScatersVisFL6FL8%03d.pdf", sep="")
1654:   pdf(file=pdfName, onefile=TRUE, width = 15, height=10 )
1655:
1656:
1657:   ####Scatters con medianas despues de la compensación
1658:   #X11(width = 15, height = 17)
1659:        tp1<- fnvisualizarScatervivasScreen (cFlowData.log[rango ],
1660:                 fun=`FL8` ~ `FL6` ,cPos.cFlowData.log[rango ],
1661:                 main= "Después compensación - log", mult=1.2,
1662:                 my.layout=laycol,
1663:                 filename="Rplot%d.png",
1664:                 imageW=21, imageH=29.7,
1665:                 paneldesc = panel.desc[[panelFile ]],
1666:                 my.scales= list(x = list(axs = "i", cex=0.4),
1667:                 y = list(axs = "i", cex=0.4))
1668:                 )
1669:
1670:    tp2<- fnvisualizarScatervivasScreen (flowData.orig[rango ],
1671:                  fun=`FL8` ~ `FL6` ,Pos.flowData.orig.N[rango ],
1672:                  main="Antes compensación - log", mult=1.2,
1673:                 my.layout=laycol,
1674:                 filename="Rplot%d.png",
1675:                 imageW=21, imageH=29.7,
1676:                 paneldesc = panel.desc[[panelFile ]],
1677:                 my.scales= list(x = list(axs = "i", cex=0.4),
1678:                 y = list(axs = "i", cex=0.4))
1679:                 )
1680:       print( plot(tp1, position = c(0.1, 0, 0.5 , 1), more = TRUE))
1681:       print( plot(tp2, position = c(0.5, 0 ,1 , 1), more = FALSE))
1682:
1683:
1684:   dev.off()
1685:
1686:   ###############################################################################
1687:   }
1688: ###############################################################################
1689: }
```

```
1690:
1691: ########################################################
1692: ###### fin de iteracion por matPanles
1693: ########################################################
1694:
1695:
1696: ##Guardamos a disco la matriz de medianas
1697:  txtFilename <- paste(dataDirOut,"/" , "finMediansData.lin.N" , ".txt", sep="")
1698:  write.table(finMediansData.lin.N ,file=txtFilename ,
1699:              row.names=TRUE,sep="\t",eol="\n",dec = ",")
1700:
1701:
1702:
1703: ##Guardamos a disco la matriz de medias
1704:  txtFilename <- paste(dataDirOut,"/" , "finMeansData.lin.N" , ".txt", sep="")
1705:  write.table(finMeansData.lin.N ,file=txtFilename ,
1706:              row.names=TRUE,sep="\t",eol="\n",dec = ",")
1707:
1708:
1709: ##Guardamos a disco la matriz de medianas
1710:  txtFilename <- paste(dataDirOut,"/" , "finMediansData.orig.N" , ".txt", sep="")
1711:  write.table(finMediansData.orig.N ,file=txtFilename ,
1712:              row.names=TRUE,sep="\t",eol="\n",dec = ",")
1713:
1714:
1715: ##Guardamos a disco la matriz de medias
1716:  txtFilename <- paste(dataDirOut,"/" , "finMeansData.orig.N" , ".txt", sep="")
1717:  write.table(finMeansData.orig.N ,file=txtFilename ,
1718:              row.names=TRUE,sep="\t",eol="\n",dec = ",")
1719:
1720:
1721: ##Guardamos a disco la matriz de medianas
1722:  txtFilename <- paste(dataDirOut,"/" , "cfinMediansData.lin.N" , ".txt", sep="")
1723:  write.table(cfinMediansData.lin.N ,file=txtFilename ,
1724:             row.names=TRUE,sep="\t",eol="\n",dec = ",")
1725:
1726:
1727: ##Guardamos a disco la matriz de medias
1728:  txtFilename <- paste(dataDirOut,"/" , "cfinMeansData.lin.N" , ".txt", sep="")
1729:  write.table(cfinMeansData.lin.N ,file=txtFilename ,
1730:              row.names=TRUE,sep="\t",eol="\n",dec = ",")
1731:
1732:
1733:
1734: ##Guardamos a disco la matriz de medianas
1735:  txtFilename <- paste(dataDirOut,"/" , "cfinMediansData.log" , ".txt", sep="")
1736:  write.table(cfinMediansData.log ,file=txtFilename ,
1737:              row.names=TRUE,sep="\t",eol="\n",dec = ",")
1738:
1739:
1740: ##Guardamos a disco la matriz de medias
1741:  txtFilename <- paste(dataDirOut,"/" , "cfinMeansData.log" , ".txt", sep="")
1742:  write.table(cfinMeansData.log ,file=txtFilename ,
1743:              row.names=TRUE,sep="\t",eol="\n",dec = ",")
1744:
1745:
1746:
1747:
```

```
 1:
 2:
 3:   ### PROGRAMA analisis.compensated_rev0.R
 4:   ### R. Tamarit , Junio 2010
 5:   ### Analisis de las medianas de los frames compensados
 6:   ########################################################
 7:   ### chunk: Load Libraries
 8:   ########################################################
 9:
10:
11:    library(flowCore)
12:    library(flowQ)
13:    library(flowViz)
14:    library(flowStats)
15:    library(flowUtils)
16:    library(geneplotter)
17:    library(colorspace)
18:    library(grid)
19:    library(MASS)
20:    library(flowFP)
21:    library("geneplotter")
22:    require("RColorBrewer")
23:    library(MASS)
24:    library(gplots)
25:
26:    ######################## remover objetos ##########
27:   # print(Objlist <- ls())
28:   # rm(Objlist )
29:
30:    ########################
31:    memory.size()
32:        memory.size(TRUE)
33:        memory.limit()
34:        memory.size(max = 3000)
35:        memory.size()
36:        memory.size(TRUE)
37:        memory.limit()
38:
39:      # source("C:/Users/ramon/DataPrj/CdHg/funciones.propias.R",echo = TRUE)
40:
41: ########################################################
42: ### chunk number 0: Set working dir
43: ########################################################
44: ###workingDir <- getwd()
45: ## Directorio de partida
46: dataDir <- "C:/Users/ramon/DataPrj/CdHg/MedianasCalculos"
47: dataDir <- "C:/Compartida/DataPrj/CdHg/MedianasCalculos"
48: setwd(dataDir)
49:
50:
51: ###salida de la compensacion
52: dataDirOut <-"C:/Users/ramon/DataPrj/CdHg/MedianasCalculos"
53: dataDirOut <-"C:/Compartida/DataPrj/CdHg/MedianasCalculos"
54:
55:
56:
57:
58: ########################################################
59: ### chunk : Variables
60: ########################################################
61:  VisualizarResultado <- "N"
62:   print(paste("### VisualizarResultado <- " , VisualizarResultado) )
63:  ifDebug <- FALSE
64:   print(paste("### ifDebug <- " , ifDebug ) )
65:
66:
67: ########################################################
68: ### chunk :Leer el flowSet
```

```
 69: ######################################################
 70:
 71:  ##Lista de Experimentos
 72:  ExpList <- "full.annot.compensated.txt"
 73:
 74:
 75:   ### Descripción de los fluorocromos de cada panel
 76:   panel.desc <- list()
 77:   panel.desc[[1]] <- c("FSC", "SSC", "FL1.DCF"   ,  "FL3.TMRM", "FL6.Indo1",
    "FL8.BDP675")
 78:   panel.desc[[2]] <- c("FSC", "SSC", "FL1.DIBAC3",  "FL3.MTSX", "FL6.MCB"  ,
    "FL8.BDP675")
 79:   panel.desc[[3]] <- c("FSC", "SSC", "FL1.DAF"   ,  "FL3.TMRM", "FL6.MCB"  ,
    "FL8.BDP675")
 80:   for (r in 1:length(panel.desc)){names(panel.desc[[r]]) <- c("FSC", "SSC", "FL1"
    ,  "FL3", "FL6", "FL8")}
 81:
 82:
 83:
 84:   flowData.log <- read.flowSet(path = ".", phenoData = ExpList,
    transformation=FALSE)
 85:   sampleNames(flowData.log) <- as.character((pData(flowData.log)[, "Id"]))
 86:   flowData.lin <- read.flowSet(path = ".", phenoData = ExpList,
    transformation="linearize")
 87:   sampleNames(flowData.lin) <- as.character((pData(flowData.lin)[, "Id"]))
 88:
 89:   print("###############Experimentos################")
 90:   Titulos<-sampleNames(flowData.log)
 91:   sampleNames(flowData.lin)
 92:   Titulos
 93:   pData(phenoData(flowData.log))
 94:   head(pData(flowData.log))
 95:   sampleNames(flowData.log)
 96:   Tclist<- sampleNames(flowData.log)
 97:   factores <-pData(flowData.log)
 98:   columnas<-colnames(flowData.log[[1]])
 99:   factores <-pData(flowData.log)
100:   Channels<-colnames(flowData.log[[1]])
101:
102:
103:
104:   ######################################################
105:   ### chunk : Selección de la población de células
106:   ###     viables con filtros rectagulares
107:   ###     y de densidad.
108:   ######################################################
109:
110:
111:   print("
112:   ######################################################################
113:   ### chunk : Definición de Filtros para selección de las celulas vivas
114:   ######################################################################")
115:
116:   ######################################################################
117:   ##### Configuracion de filtro rectangular
118:   print(minFSC <- 500)
119:   print(maxFSC <- 2200)
120:   print(minSSC <- 200)
121:   print(maxSSC <- 2000)
122:   rectGate <- rectangleGate(filterId="rectangleGate",
123:             "FSC"=c(minFSC , maxFSC),
124:             "SSC"=c(minSSC , maxSSC))
125:
126:   ######################################################################
127:   ##### Configuracion de filtro morphGateScale
128:   #####  norm2filter
129:   print(morphGateScale <- 2.5)
130:     morphGate <-norm2Filter(filterId = "MorphologyGate",
```

```
131:                "FSC",  "SSC",
132:                scale = morphGateScale )
133:   ######################################################################
134:
135:
136:   ##########################################################################
137:   print("
138:   ##########################################################################
139:   ### Seleccion de positivos en el flowSet
140:   ##########################################################################
141:   ##########################################################################")
142:   ###filtramos primero con rectGate
143:   ##########################################################################
144:   frcfs.log <- filter(flowData.log, rectGate)
145:   rcPos.flowData.log<-Subset(flowData.log, rectGate)
146:
147:   frcfs.lin <- filter(flowData.lin , rectGate)
148:   rcPos.flowData.lin <-Subset(flowData.lin , rectGate)
149:
150:   if (ifDebug){
151:     prepareOutput(mult=0.8)
152:     print(xyplot(SSC ~ FSC , rcPos.flowData.log,smooth=TRUE, filter=rectGate ,
153:       layout=c(6,1), subset = (CTox == 0), xlab="",
154:       scales = list(x = list(axs = "i"), y = list(draw = FALSE)),
155:       main=paste("Seleccion- minFSC:", minFSC,
156:                  " maxFSC:", maxFSC,
157:                  " minSSC:", minSSC,
158:                  " maxSSC:", maxSSC)))
159:       }
160:   ### Seleccion de la población con mayor densidad.
161:   ### con un filtro norm2Filter
162:        ##########################################################################
163:   fPosfs.log <- filter(rcPos.flowData.log, morphGate )
164:   Pos.flowData.log  <-Subset(rcPos.flowData.log, morphGate )
165:
166:   fPosfs.lin <- filter(rcPos.flowData.lin , morphGate )
167:   Pos.flowData.lin <-Subset(rcPos.flowData.lin , morphGate )
168:
169:   if (ifDebug){
170:     prepareOutput(mult=0.8)
171:     print(xyplot(SSC ~ FSC , rcPos.flowData.log ,smooth=TRUE, filter=morphGate ,
172:       layout=c(6,1), subset = (CTox == 0), xlab="",
173:       scales = list(x = list(axs = "i"), y = list(draw = FALSE)),
174:       main=paste("Seleccion- morphGateScale <- ", morphGateScale )))
175:       }
176:
177:
178:   print("
179:   ##########################################################
180:   ### chunk : ### Cálculo de las estadisticas de selección
181:   ##########################################################")
182:
183:
   ##############################################################################################
184:
185:   Total <-as.numeric(fsApply(flowData.lin, nrow, use.exprs = TRUE))
186:   Pos <-as.numeric(fsApply(Pos.flowData.lin, nrow, use.exprs = TRUE))
187:   data1 <-data.frame(Files = Titulos, "Total Cells" = Total, "Live Cells" = Pos )
188:   data1 <-transform(data1, Percent = round(data1[, 3] * 100/data1[,2], 0))
189:   EstatSel <-as.matrix(data1)
190:   EstatSel
191:   factores.EstatSel <- pData(flowData.lin)
192:   EstatSel <- cbind(EstatSel,factores.EstatSel )
193:
   ##############################################################################################
194:    print("##Guardamos a disco la matriz de estadisticas")
195:    txtFilename <- paste(dataDirOut,"/" , ExpList ,".EstadisticasSeleccion", ".txt",
   sep="")
```

```
196:    write.table(EstatSel ,file=txtFilename ,row.names=TRUE,sep="\t",eol="\n",dec = ",
    ")
197:    print(txtFilename )
198:
199:
200:
201:
202:
203:   ######################################################
204:   ### chunk : ## Separar por paneles      ###
205:   ######################################################
206:
207:   ## Separamos el flowSet por Paneles
208:   splitPanelsFlowData.log <- split(Pos.flowData.log, factores[,"panel"])
209:   splitPanelsFlowData.lin <- split(Pos.flowData.lin, factores[,"panel"])
210:
211:   panels <- names(splitPanelsFlowData.log )
212:   print(paste("Paneles: ", panels ))
213:
214:   ## cada panel es un elemento de la lista
215:   ## splitPanelsFlowData
216:
217:
218:   ######################################################
219:   ### chunk : ## Analisis de densidad
220:   ######################################################
221:   #### Solo de los datos en log
222:   ## Iteramos para cada panel ....
223:   for (panelId in 1:length(panels ) ){
224:   ###For Debug panelId<-1
225:   print(paste("Iterando panel: ", panels[panelId ] ))
226:   PanelFlowData.log <- splitPanelsFlowData.log [[panelId]]
227:   factoresSplit.log <-pData(PanelFlowData.log )
228:   ### Los controles son los que no tienen toxico CTox=0
229:     splitFlowData.log <- split(PanelFlowData.log , factoresSplit.log[,"CTox"])
230:     print(names(splitFlowData.log ))
231:     ControlFlowData.log <- splitFlowData.log[[1]]
232:     print(sampleNames(ControlFlowData.log ))
233:
234:     X11()
235:
236:     trellis.par.set(theme = col.whitebg())
237:     lw <- list(ylab.axis.padding = list(x = 0.5), left.padding = list(x = 0.5,
238:        units = "inches"), right.padding = list(x = 0, units = "inches"),
239:        panel = list(x = 1.4, units = "inches"))
240:     lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
241:       list(x = 0, units = "inches"), panel = list(x = 4, units = "inches"))
242:
243:     lattice.options(layout.widths = lw, layout.heights = lh,las=2)
244:
245:     X11(width = 14, height = 10, pointsize = 12)
246:     print(densityplot( ~ `FL1` + `FL3` +  `FL6` + `FL8` , xlim=c(-500,4500),
247:         ControlFlowData.log , overlap=0.5,  main=paste("Panel ",panelId  )),
    darg=c(bw=0.2, n=200))
248:
249:
250:     X11()
251:
252:     trellis.par.set(theme = col.whitebg())
253:     lw <- list(ylab.axis.padding = list(x = 0.5), left.padding = list(x = 0.5,
254:        units = "inches"), right.padding = list(x = 0, units = "inches"),
255:        panel = list(x = 1.4, units = "inches"))
256:     lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
257:       list(x = 0, units = "inches"), panel = list(x = 6, units = "inches"))
258:
259:     lattice.options(layout.widths = lw, layout.heights = lh,las=2)
260:
261:     X11(width = 50, height = 35, pointsize = 12)
```

```
262:     print(densityplot(~  `FL1` + `FL3` +  `FL6` + `FL8` ,
263:         PanelFlowData.log , overlap=0.1,  main=paste("Panel ",panelId  )),
     darg=c(bw=0.2))
264:
265:     fDensityTest2(ControlFlowData.log , texto="rcPosTFS.log", iminVs=0, imaxVs=2500,

266:                   intMax=0.04,isXlog=TRUE)
267:
268:     #viewPars(ControlFlowData.log )
269:
270:
271:   };## fin de la iteracion por el panel
272:
273:   ##### las intensidades basales no son comparables ....
274:
275:
276:
277:
     print("############################################################################################ )
278:   print("#Medias y medianas de los frames  en escala log")
279:
     print("############################################################################################ )
280:   print("#Calculamos las medianas de los experimentos")
281:
282:   TMedians.log<- NULL
283:   data.TMedians.log<- NULL
284:   MediansData.TMedians.log<- NULL
285:
286:   TMeans.log<- NULL
287:   Data.TMedians.log<- NULL
288:   MediansData.log<- NULL
289:   ########## Cálculo de medianas ###################################
290:   TMedians.log<-as.matrix(fsApply(Pos.flowData.log, each_col, median))
291:   TMedians.log
292:   data.TMedians.log<-data.frame(Files = rownames(TMedians.log),
293:                       "FSC" = TMedians.log[, 1],
294:              "SSC" = TMedians.log[, 2],
295:              "FL1" = TMedians.log[, 3],
296:              "FL3" = TMedians.log[, 4],
297:              "FL6" = TMedians.log[, 5],
298:              "FL8" = TMedians.log[, 6])
299:
300:   factores.ord <- NULL
301:   factores.ord <- pData(phenoData(Pos.flowData.log))
302:
303:
304:
305:   MediansData.log<- cbind(data.TMedians.log,factores.ord )
306:   finMediansData.log<- MediansData.log
307:
308:
309:    print("##Guardamos a disco la matriz de medianas")
310:    txtFilename <- paste(dataDirOut,"/" , ExpList ,".MatrizMedianasOriginales.log.",
     ".txt", sep="")
311:    write.table(MediansData.log,file=txtFilename ,row.names=TRUE,sep="\t",eol="\n",
     dec = ",")
312:    print(txtFilename )
313:
314:
315:
316:
     ##############################################################################################
317:
318:
319:
     print("############################################################################################ )
320:   print("#Medias y medianas de los frames  en escala Lin")
321:
     print("############################################################################################ )
```

```
322:   print("#Calculamos las medianas de los experimentos")
323:
324:   TMedians.lin <- NULL
325:   data.TMedians.lin<- NULL
326:   MediansData.TMedians.lin<- NULL
327:
328:   TMeans.lin<- NULL
329:   Data.TMedians.lin <- NULL
330:   MediansData.lin<- NULL
331:   ########## Cálculo de medianas ####################################
332:   TMedians.lin<-as.matrix(fsApply(Pos.flowData, each_col, median))
333:   TMedians.lin
334:   data.TMedians.lin<-data.frame(Files = rownames(TMedians.lin),
335:                       "FSC" = TMedians.lin[, 1],
336:              "SSC" = TMedians.lin[, 2],
337:              "FL1" = TMedians.lin[, 3],
338:              "FL3" = TMedians.lin[, 4],
339:              "FL6" = TMedians.lin[, 5],
340:              "FL8" = TMedians.lin[, 6])
341:
342:   factores.ord <- NULL
343:   factores.ord <- pData(phenoData(Pos.flowData))
344:
345:
346:
347:   MediansData.lin<- cbind(data.TMedians.lin,factores.ord )
348:   finMediansData.lin<- MediansData.lin
349:
350:
351:    print("##Guardamos a disco la matriz de medianas")
352:    txtFilename <- paste(dataDirOut,"/" , ExpList ,".MatrizMedianasOriginaleslin.",
    ".txt", sep="")
353:    write.table(MediansData.lin,file=txtFilename ,row.names=TRUE,sep="\t",eol="\n",
    dec = ",")
354:    print(txtFilename )
355:
356:
357:
358:   ###########################################################################
359:   ### chunk: CALCULO DE LAS POBLACIONES POSITIVAS DE LA SELECCIÓN DE CELULAS VIVAS
360:   ###########################################################################
361:   splitPoints.log <- matrix(0, nrow = 3, ncol = 4, byrow = TRUE,
362:         dimnames = list(c(1:3),c("FL1","FL3","FL6","FL8")))
363:
364:
365:
366:   ## Iteramos para cada panel ....
367:   for (panelId in 1:length(panels ) ){
368:   ###For Debug panelId<-2
369:   print(paste("Iterando panel: ", panels[panelId ] ))
370:   PanelFlowData.log <- splitPanelsFlowData.log [[panelId]]
371:   factoresSplit.log <-pData(PanelFlowData.log )
372:   ### Los controles son los que no tienen toxico CTox=0
373:     splitFlowData.log <- split(PanelFlowData.log , factoresSplit.log[,"CTox"])
374:     print(names(splitFlowData.log ))
375:     ControlFlowData.log <- splitFlowData.log[[1]]
376:     print(sampleNames(ControlFlowData.log ))
377:
378:   # El calculo se raliza con las celulas seleccionadas en eslala lineal##
379:   ## La funcion rangeGate crea un filtro que separa las poblaciones positivas y
    negativas
380:   x11(width = 4, height=4)
381:   rg1 <- rangeGate(ControlFlowData.log ,filterId="fltFL1","FL1",plot=TRUE,
    absolute=FALSE,positive=TRUE,alpha=0.5)
382:   x11(width = 4, height=4)
383:   rg3 <- rangeGate(ControlFlowData.log ,filterId="fltFL3","FL3",plot=TRUE,
    absolute=FALSE,positive=TRUE,alpha=0.5)
384:   x11(width = 4, height=4)
```

```
385:   rg6 <- rangeGate(ControlFlowData.log ,filterId="fltFL6","FL6",plot=TRUE,
     absolute=FALSE,positive=TRUE,alpha=0.5)
386:   x11(width = 4, height=4)
387:   rg8 <- rangeGate(ControlFlowData.log ,filterId="fltFL8","FL8",plot=TRUE,
     absolute=FALSE,positive=TRUE,alpha=0.5)
388:
389:   ControlFlowData.log.rg1 <- filter(ControlFlowData.log, rg1 )
390:   a1.orig<-filterDetails(ControlFlowData.log.rg1[[1]])$fltFL1$filter@min
391:   a1.orig
392:   splitPoints.log[panelId, "FL1"]<- a1.orig
393:
394:   ControlFlowData.log.rg3 <- filter(ControlFlowData.log, rg3 )
395:   a3.orig<-filterDetails(ControlFlowData.log.rg3[[1]])$fltFL3$filter@min
396:   a3.orig
397:   splitPoints.log[panelId, "FL3"]<- a3.orig
398:
399:   ControlFlowData.log.rg6 <- filter(ControlFlowData.log, rg6 )
400:   a6.orig<-filterDetails(ControlFlowData.log.rg6[[1]])$fltFL6$filter@min
401:   a6.orig
402:   splitPoints.log[panelId, "FL6"]<- a6.orig
403:
404:   ControlFlowData.log.rg8 <- filter(ControlFlowData.log, rg8 )
405:   a8.orig<-filterDetails(ControlFlowData.log.rg8[[1]])$fltFL8$filter@min
406:   a8.orig
407:   splitPoints.log[panelId, "FL8"]<- a8.orig
408:   }
409:
410:   splitPoints.log[2:3, "FL6"] <- 2500
411:
412:   splitPoints.lin <- 10^(splitPoints.log*4/4096)
413:
414:   PosCels <- list()
415:   NegCels <- list()
416:   PercPos <- list()
417:   TMedians.pos.lin <- list()
418:   TMedians.neg.lin <- list()
419:
420:   for (panelId in 1:length(panels ) ){
421:   ###For Debug panelId<-2
422:   print(paste("Iterando panel: ", panels[panelId ] ))
423:   factores.calc <- NULL
424:   factores.calc <-pData(splitPanelsFlowData.lin[[panelId]] )
425:   ### Filtramos el flowSet lineal
426:
427:   rg1 <- rectangleGate(filterId="fltFL1", "FL1"=c(splitPoints.lin[panelId, "FL1"],
     Inf))
428:   Pos.flowData.lin.rg1 <- filter(splitPanelsFlowData.lin[[panelId]], rg1 )
429:   FL1.pos.lin <-split(splitPanelsFlowData.lin[[panelId]], Pos.flowData.lin.rg1
     )$'fltFL1+'
430:   FL1.neg.lin <-split(splitPanelsFlowData.lin[[panelId]], Pos.flowData.lin.rg1
     )$'fltFL1-'
431:   TMedians.lin.FL1 <- as.matrix(fsApply(FL1.pos.lin, each_col, median))[,"FL1"]
432:   TMedians.neg.lin.FL1 <- as.matrix(fsApply(FL1.neg.lin, each_col, median))[,"FL1"]
433:   Pos.FL1 <-as.numeric(fsApply(FL1.pos.lin, nrow, use.exprs = TRUE))
434:   Neg.FL1 <-as.numeric(fsApply(FL1.neg.lin, nrow, use.exprs = TRUE))
435:   perc.pos.FL1 <- round(100*Pos.FL1/(Pos.FL1+Neg.FL1),2)
436:
437:
438:   rg3 <- rectangleGate(filterId="fltFL3", "FL3"=c(splitPoints.lin[panelId, "FL3"],
     Inf))
439:   Pos.flowData.lin.rg3 <- filter(splitPanelsFlowData.lin[[panelId]], rg3 )
440:   FL3.pos.lin<-split(splitPanelsFlowData.lin[[panelId]], Pos.flowData.lin.rg3
     )$'fltFL3+'
441:   FL3.neg.lin<-split(splitPanelsFlowData.lin[[panelId]], Pos.flowData.lin.rg3
     )$'fltFL3-'
442:   TMedians.lin.FL3 <- as.matrix(fsApply(FL3.pos.lin, each_col, median))[,"FL3"]
443:   TMedians.neg.lin.FL3 <- as.matrix(fsApply(FL3.neg.lin, each_col, median))[,"FL3"]
444:   Pos.FL3<-as.numeric(fsApply(FL3.pos.lin, nrow, use.exprs = TRUE))
```

```
445:   Neg.FL3 <-as.numeric(fsApply(FL3.neg.lin, nrow, use.exprs = TRUE))
446:   perc.pos.FL3 <- round(100*Pos.FL3/(Pos.FL3+Neg.FL3),2)
447:
448:
449:   rg6 <- rectangleGate(filterId="fltFL6", "FL6"=c(splitPoints.lin[panelId, "FL6"],
      Inf))
450:   Pos.flowData.lin.rg6 <- filter(splitPanelsFlowData.lin[[panelId]], rg6 )
451:   FL6.pos.lin<-split(splitPanelsFlowData.lin[[panelId]], Pos.flowData.lin.rg6
      )$'fltFL6+'
452:   FL6.neg.lin<-split(splitPanelsFlowData.lin[[panelId]], Pos.flowData.lin.rg6
      )$'fltFL6-'
453:   TMedians.lin.FL6 <- as.matrix(fsApply(FL6.pos.lin, each_col, median))[,"FL6"]
454:   TMedians.neg.lin.FL6 <- as.matrix(fsApply(FL6.neg.lin, each_col, median))[,"FL6"]
455:   Pos.FL6 <-as.numeric(fsApply(FL6.pos.lin, nrow, use.exprs = TRUE))
456:   Neg.FL6 <-as.numeric(fsApply(FL6.neg.lin, nrow, use.exprs = TRUE))
457:   perc.pos.FL6 <- round(100*Pos.FL6/(Pos.FL6+Neg.FL6),2)
458:
459:
460:   rg8 <- rectangleGate(filterId="fltFL8", "FL8"=c(splitPoints.lin[panelId, "FL8"],
      Inf))
461:   Pos.flowData.lin.rg8 <- filter(splitPanelsFlowData.lin[[panelId]], rg8 )
462:   FL8.pos.lin<-split(splitPanelsFlowData.lin[[panelId]],
      Pos.flowData.lin.rg8)$'fltFL8+'
463:   FL8.neg.lin<-split(splitPanelsFlowData.lin[[panelId]],
      Pos.flowData.lin.rg8)$'fltFL8-'
464:   TMedians.lin.FL8 <- as.matrix(fsApply(FL8.pos.lin, each_col, median))[,"FL8"]
465:   TMedians.neg.lin.FL8 <- as.matrix(fsApply(FL8.neg.lin, each_col, median))[,"FL8"]
466:   Pos.FL8<-as.numeric(fsApply(FL8.pos.lin, nrow, use.exprs = TRUE))
467:   Neg.FL8 <-as.numeric(fsApply(FL8.neg.lin, nrow, use.exprs = TRUE))
468:   perc.pos.FL8 <- round(100*Pos.FL8/(Pos.FL8+Neg.FL8),2)
469:
470:
471:
472:
473:   TMedians.pos.lin[[panelId]] <-cbind(TMedians.lin.FL1,TMedians.lin.FL3,
      TMedians.lin.FL6,TMedians.lin.FL8,factores.calc)
474:   TMedians.neg.lin[[panelId]] <-cbind(TMedians.neg.lin.FL1,TMedians.neg.lin.FL3,
      TMedians.neg.lin.FL6,TMedians.neg.lin.FL8,factores.calc)
475:   PosCels[[panelId]] <- cbind(Pos.FL1,Pos.FL3,Pos.FL6,Pos.FL8,factores.calc)
476:   NegCels[[panelId]] <- cbind(Neg.FL1,Neg.FL3,Neg.FL6,Neg.FL8,factores.calc)
477:   PercPos[[panelId]] <- cbind(perc.pos.FL1,perc.pos.FL3,perc.pos.FL6,perc.pos.FL8,
      factores.calc)
478:   rownames(PercPos[[panelId]])<- rownames(TMedians.pos.lin[[panelId]])
479:   colnames(PercPos[[panelId]])<- colnames(TMedians.pos.lin[[panelId]])
480:
481:   colnames(TMedians.pos.lin[[panelId]])<- c("FL1","FL3","FL6","FL8","Id","Date",
      "IDate","panel","StainId","Stains","CTox","Tox","name","split")
482:   colnames(TMedians.neg.lin[[panelId]])<- c("FL1","FL3","FL6","FL8","Id","Date",
      "IDate","panel","StainId","Stains","CTox","Tox","name","split")
483:   colnames(PercPos[[panelId]])<- c("FL1","FL3","FL6","FL8","Id","Date","IDate",
      "panel","StainId","Stains","CTox","Tox","name","split")
484:
485:
486:   }
487:   PorcPos.lin <-  rbind(PercPos[[1]],PercPos[[2]],PercPos[[3]])
488:
489:   MediansData.pos.lin <- rbind(TMedians.pos.lin[[1]],TMedians.pos.lin[[2]],
      TMedians.pos.lin[[3]])
490:   MediansData.neg.lin <- rbind(TMedians.neg.lin[[1]],TMedians.neg.lin[[2]],
      TMedians.neg.lin[[3]])
491:   MediansData.pos.lin
492:   MediansData.neg.lin
493:
494:
495:   ### Calcular el ratio de variación respecto del control
496:
497:   controles.pos <- MediansData.pos.lin [ which (MediansData.pos.lin ["CTox"]==0), ]
498:   controles.neg <- MediansData.neg.lin [ which (MediansData.neg.lin ["CTox"]==0), ]
```

```
499:   Expers <- MediansData.pos.lin [ which (MediansData.pos.lin["CTox"]!=0), ]
500:   ContrId <- Expers[,"name"]
501:   Expers <- cbind(Expers, ContrId )
502:   porcVar <- cbind(Expers, ContrId )
503:
504:   for (i in 1:nrow(Expers)){
505:
506:       ## debug i=1
507:       st <- Expers [i,"StainId"]
508:       tx <- Expers [i,"Tox"]
509:       pn <- Expers [i,"panel"]
510:       isControl <- which (controles.pos["StainId"]==st  & controles.pos["Tox"]==tx
    & controles.pos["panel"]==pn )
511:       isBaseLine <- which (controles.neg["StainId"]==st  & controles.pos["Tox"]==tx
    & controles.neg["panel"]==pn )
512:       control <- controles.pos[ isControl  , ]
513:       baseLine <-  controles.neg[ isBaseLine  , ]
514:
515:       Expers [i,"FL1"] <- round((Expers [i,"FL1"] - baseLine[1,"FL1"])/ (control[1,
    "FL1"]-baseLine[1,"FL1"]),2)
516:       Expers [i,"FL3"] <- round((Expers [i,"FL3"] - baseLine[1,"FL3"])/ (control[1,
    "FL3"]-baseLine[1,"FL3"]),2)
517:       Expers [i,"FL6"] <- round((Expers [i,"FL6"] - baseLine[1,"FL6"])/ (control[1,
    "FL6"]-baseLine[1,"FL6"]),2)
518:       Expers [i,"FL8"] <- round((Expers [i,"FL8"] - baseLine[1,"FL8"])/ (control[1,
    "FL8"]-baseLine[1,"FL8"]),2)
519:
520:
521:       porcVar [i,"FL1"] <- round((((porcVar [i,"FL1"] - baseLine[1,"FL1"])/
    (control[1,"FL1"]-baseLine[1,"FL1"]))-1)*100,0)
522:       porcVar [i,"FL3"] <- round((((porcVar [i,"FL3"] - baseLine[1,"FL3"])/
    (control[1,"FL3"]-baseLine[1,"FL3"]))-1)*100,0)
523:       porcVar [i,"FL6"] <- round((((porcVar [i,"FL6"] - baseLine[1,"FL6"])/
    (control[1,"FL6"]-baseLine[1,"FL6"]))-1)*100,0)
524:       porcVar [i,"FL8"] <- round((((porcVar [i,"FL8"] - baseLine[1,"FL8"])/
    (control[1,"FL8"]-baseLine[1,"FL8"]))-1)*100,0)
525:
526:       }
527:
528:     porcVar
529:
530:    print("##Guardamos a disco la matriz de variaciones")
531:    txtFilename <- paste(dataDirOut,"/" , ExpList ,
    ".MatrizVariacionesCompensadas.lin.", ".txt", sep="")
532:    write.table(Expers,file=txtFilename ,row.names=TRUE,sep="\t",eol="\n",dec = ",")
533:    print(txtFilename )
534:
535:
536:   print("##Guardamos a disco la matriz de variaciones")
537:    txtFilename <- paste(dataDirOut,"/" , ExpList ,
    ".MatrizVariacionesCompensadas.porc.lin.", ".txt", sep="")
538:    write.table(porcVar ,file=txtFilename ,row.names=TRUE,sep="\t",eol="\n",dec = ",
    ")
539:    print(txtFilename )
540:
541:
542:   print("##Guardamos a disco la matriz de porcentajes")
543:    txtFilename <- paste(dataDirOut,"/" , ExpList ,
    ".MatrizPorcentajes.positivas.lin.", ".txt", sep="")
544:    write.table(PorcPos.lin,file=txtFilename ,row.names=TRUE,sep="\t",eol="\n",dec =
    ",")
545:    print(txtFilename )
546:
547:
548:
549:   Titulos
550:   factores
551:   columnas
```

```
552:   Channels
553:   fluoChannels <- Channels[-c(1:2)]
554:   Toxs <- c("Cd", "Hg")
555:   stainsInt <- as.matrix(c("A", "B", "C", "D", "E"))
556:   rownames(stainsInt ) <- c("full",  "FL1", "FL3", "FL6", "FL8")
557:   revStains <- as.matrix(c("full",  "FL1", "FL3", "FL6", "FL8"))
558:   rownames(revStains ) <- c("A", "B", "C", "D", "E")
559:   descript.fluoPanel <- list()
560:   descript.fluoPanel[[1]] <-c("FSC", "SSC",
561:             "FL1.DCF - Actividad peroxidasa",
562:             "FL3.TMRM - Potencial de membrana mitocondrial",
563:             "FL6.Indo1 - Calcio intracelular",
564:             "FL8.BDP675 - Peroxidación lipídica")
565:
566:   descript.fluoPanel[[2]] <- c("FSC", "SSC",
567:             "FL1.DIBAC3 - Potencial de membrana plasmática",
568:             "FL3.MTSX - Superoxido mitocondrial",
569:             "FL6.MCB - Tioles libres - glutation",
570:             "FL8.BDP675 - Peroxidación lipídica")
571:
572:   descript.fluoPanel[[3]] <- c("FSC", "SSC",
573:             "FL1.DAF - Niveles de óxido nítrico",
574:             "FL3.TMRM - Potencial de membrana mitocondrial",
575:             "FL6.MCB - Tioles libres - glutation",
576:             "FL8.BDP675 - Peroxidación lipídica")
577:
578:
579:   for (r in 1:length(descript.fluoPanel)) {
580:       names(descript.fluoPanel[[r]]) <- c("FSC", "SSC", "FL1", "FL3", "FL6", "FL8")}
581:
582:
583:   ########################################################
584:   ### Graficos de % de variación respecto del control
585:   ########################################################
586:   for (panelId in 1:length(panels) ){
587:
588:   x11(width=80, height=160)
589:   def.par <- par(no.readonly = TRUE) # save default, for resetting...
590:       mat<-matrix(c(1:16), 4, 1,  byrow=TRUE)
591:       layout(mat, widths = rep(1, ncol(mat)),
592:       heights = rep(1, nrow(mat)), respect = FALSE)
593:   par(mar=c(4.5, 4.5, 3.0, 0.5))
594:     ###For Debug panelId<-2
595:     print(paste("Iterando panel: ", panels[panelId ] ))
596:
597:     for (st in stainsInt[2:5]){
598:       #fordebug st<-2
599:       Cualesfullstains <- which(porcVar [,"StainId"]=="A" &
600:                 porcVar [,"panel"]==panelId )
601:       fullstains <- porcVar [Cualesfullstains ,]
602:
603:       cualestox <-  which(porcVar [,"panel"]==panelId &
604:               porcVar [,"StainId"]==st  )
605:       grupotox <- porcVar [cualestox ,]
606:
607:       grupo <- rbind(fullstains ,grupotox  )
608:       grupoOrdenado <- sort.data.frame(grupo, by= ~ "Tox")
609:       #print(grupoOrdenado )
610:       chann <- revStains[[st,1]]
611:       print(chann )
612:           tit <- descript.fluoPanel[[panelId ]][[chann ]]
613:       tity <- "% control"
614:       barplot(grupoOrdenado [,chann ],
615:           names.arg=rownames(grupoOrdenado ),
616:           ylab=tity, main=paste(tit, " -Pnl.", panelId , sep="" ),
617:           las=2, cex.names=0.9, cex.main=1)
618:       abline( h=0 , col="blue")
619:       abline( h=15 , col="red")
```

```
620:       abline( h=-15 , col="red")
621:     }
622:   }
623:
624:   ########################################################
625:   ### Graficos de % de positivos
626:   ########################################################
627:   for (panelId in 1:length(panels) ){
628:
629:   x11(width=80, height=160)
630:   def.par <- par(no.readonly = TRUE) # save default, for resetting...
631:       mat<-matrix(c(1:16), 4, 1,  byrow=TRUE)
632:       layout(mat, widths = rep(1, ncol(mat)),
633:       heights = rep(1, nrow(mat)), respect = FALSE)
634:   par(mar=c(4.5, 4.5, 3.0, 0.5))
635:
636:     ###For Debug panelId<-2
637:     print(paste("Iterando panel: ", panels[panelId ] ))
638:
639:     for (st in stainsInt[2:5]){
640:       #fordebug st<-2
641:       Cualesfullstains <- which(PorcPos.lin[,"StainId"]=="A" &
642:                 PorcPos.lin[,"panel"]==panelId )
643:       fullstains <- PorcPos.lin[Cualesfullstains ,]
644:
645:       cualestox <-  which(PorcPos.lin[,"panel"]==panelId &
646:               PorcPos.lin[,"StainId"]==st  )
647:       grupotox <- PorcPos.lin[cualestox ,]
648:
649:       grupo <- rbind(fullstains ,grupotox  )
650:       grupoOrdenado <- sort.data.frame(grupo, by= ~ "Tox")
651:       #print(grupoOrdenado )
652:       chann <- revStains[[st,1]]
653:       print(chann )
654:           tit <- descript.fluoPanel[[panelId ]][[chann ]]
655:       tity <- "% control"
656:       barplot(grupoOrdenado [,chann ],
657:           names.arg=rownames(grupoOrdenado ),
658:           ylab=tity, main=paste(tit, " -Pnl.", panelId , sep="" ),
659:           las=2, cex.names=1, cex.main=1, ylim=c(0,100))
660:       abline( h=0 , col="blue")
661:       abline( h=15 , col="red")
662:       abline( h=-15 , col="red")
663:     }
664:   }
665:
666:
667:
668:   ########################################################
669:   ### Graficos de % de celulas viables
670:   ########################################################
671:
672:   x11(width=160, height=80)
673:   def.par <- par(no.readonly = TRUE) # save default, for resetting...
674:       mat<-matrix(c(1:2), 2, 1,  byrow=TRUE)
675:       layout(mat, widths = rep(1, ncol(mat)),
676:       heights = rep(1, nrow(mat)), respect = FALSE)
677:   par(mar=c(4.5, 4.5, 3.0, 0.5))
678:
679:     for (tx in Toxs ){
680:       #fordebug tx <-"Cd"
681:       cualesTox <- which(EstatSel[,"Tox"]==tx )
682:
683:       grupotox <- EstatSel[cualesTox ,]
684:       cualesControl <- which(grupotox [,"CTox"]==0 & grupotox [,"Tox"]==tx )
685:
686:       matcol <- matrix(data="grey" ,nrow =nrow(grupotox ) , ncol = 1)
687:       for (l in 1:length(cualesControl )) {
```

```
688:           print(l)
689:           print(cualesControl[[l]])
690:           matcol[cualesControl[[l]] ,] <- "green"
691:           }
692:       grupoOrdenado <- sort.data.frame(grupotox , by= ~ "Files")
693:       #print(grupoOrdenado )
694:
695:           tit <- paste("Toxico ", tx)
696:       tity <- "% viables"
697:       matriz <- data.frame(grupoOrdenado)
698:       barplot(as.numeric(matriz [,"Percent"]),
699:           names.arg=rownames(grupoOrdenado ),
700:           ylab=tity, main=tit, col = matcol ,
701:           las=2, cex.names=0.9, cex.main=1.2, ylim=c(0,100))
702:       abline( h=0 , col="blue")
703:       abline( h=20 , col="red")
704:
705:     }
706:
707:
708:
709:
710:   ##########################################################
711:   ### Graficos de medianas positivas y negativas
712:   ##########################################################
713:   for (panelId in 1:length(panels) ){
714:
715:   x11(width=80, height=160)
716:   def.par <- par(no.readonly = TRUE) # save default, for resetting...
717:       mat<-matrix(c(1:16), 4, 1,  byrow=TRUE)
718:       layout(mat, widths = rep(1, ncol(mat)),
719:       heights = rep(1, nrow(mat)), respect = FALSE)
720:   par(mar=c(4.5, 4.5,2.5, 2.5))
721:     ###For Debug panelId<-2
722:     print(paste("Iterando panel: ", panels[panelId ] ))
723:
724:     for (st in stainsInt[2:5]){
725:       #fordebug st<-2
726:       Cualesfullstains <- which(MediansData.pos.lin[,"StainId"]=="A" &
727:                 MediansData.pos.lin[,"panel"]==panelId )
728:       fullstains <- MediansData.pos.lin[Cualesfullstains ,]
729:       cualestox <-  which(MediansData.pos.lin[,"panel"]==panelId &
730:               MediansData.pos.lin[,"StainId"]==st  )
731:       grupotox <- MediansData.pos.lin[cualestox ,]
732:       grupo <- rbind(fullstains ,grupotox  )
733:       grupoOrdenado <- sort.data.frame(grupo, by= ~ "Tox")
734:
735:       fullstains.neg <- MediansData.neg.lin[Cualesfullstains ,]
736:       grupotox.neg <-  MediansData.neg.lin[cualestox ,]
737:       grupo.neg <- rbind(fullstains.neg ,grupotox.neg  )
738:       grupoOrdenado.neg <- sort.data.frame(grupo.neg, by= ~ "Tox")
739:
740:
741:
742:       #print(grupoOrdenado )
743:       chann <- revStains[[st,1]]
744:       print(chann )
745:           tit <- descript.fluoPanel[[panelId ]][[chann ]]
746:       tity <- "Mediana Int.lin."
747:
748:
749:       grupoOrdenado[is.na(grupoOrdenado [,chann ]),chann] <- 0
750:       ymax <- signif(max(grupoOrdenado [,chann ]),2) + 50
751:       plot(grupoOrdenado [,chann ], ylim=c(0,ymax ),xaxt="n",  col="blue", pch=19,
752:         main= paste ("pnl",panelId ," - " , tit), ylab=tity, xlab="" ,cex.main=1 )
753:
754:       for( punto in 1:length(rownames(grupoOrdenado ))){
755:       #debug punto<-1
```

```
756:
757:       pointsmat <- cbind(grupoOrdenado.neg[punto ,chann ],grupoOrdenado [punto ,
    chann ])
758:       lines(c(punto,punto), c(grupoOrdenado.neg[punto ,chann ],
759:         grupoOrdenado[punto ,chann ] ), col = "red" )
760:         }
761:       points(grupoOrdenado.neg[,chann ], col="red", pch=19)
762:       axis(1, at=1:length(rownames(grupoOrdenado)), cex.axis=0.9, las=2,
    lab=rownames(grupoOrdenado))
763:     }
764:   }
765:
766:
767:
768:
```

```
 1:
 2:
 3:   ### PROGRAMA analisis.positive.ctrl.R
 4:   ### R. Tamarit , Junio 2010
 5:
 6:   ######################################################
 7:   ### chunk: Load Libraries
 8:   ######################################################
 9:
10:
11:    library(flowCore)
12:    library(flowQ)
13:    library(flowViz)
14:    library(flowStats)
15:    library(flowUtils)
16:    library(geneplotter)
17:    library(colorspace)
18:    library(grid)
19:    library(MASS)
20:    library(flowFP)
21:    library("geneplotter")
22:    require("RColorBrewer")
23:    library(MASS)
24:    library(gplots)
25:
26:    ######################## remover objetos ##########
27:   # print(Objlist <- ls())
28:   # rm(Objlist )
29:
30:    ########################
31:    memory.size()
32:        memory.size(TRUE)
33:        memory.limit()
34:        memory.size(max = 3000)
35:        memory.size()
36:        memory.size(TRUE)
37:        memory.limit()
38:
39: ######################################################
40: ### chunk number 0: Set working dir
41: ######################################################
42: ###workingDir <- getwd()
43: ## Directorio de partida
44: dataDir <- "C:/Compartida/DataPrj/Fulldata"
45: setwd(dataDir)
46:
47:
48: ###salida de la compensacion
49: dataDirOut <-"C:/Compartida/DataPrj/Fulldata/Outrev3"
50:
51:
52:
53: ######################################################
54: ### chunk : Variables
55: ######################################################
56:
57:  ifDebug <- FALSE
58:   print(paste("### ifDebug <- " , ifDebug ) )
59:
60:
61: ######################################################
62: ### chunk : Filtar el experimento
63: ####         "PARAQ0 2A.FCS" que tiene demasiados eventos
64: ######################################################
65:
66: filtraPARAQ02A <- FALSE
67: if (filtraPARAQ02A) {
68:     dat<-read.FCS("PARAQ0 2A.FCS",transformation=FALSE )
```

```
69:     sf <- sampleFilter(filterId="mySampleFilter", size=5000)
70:     sf
71:
72:     fres <- filter(dat, sf)
73:     fres
74:     summary(fres)
75:
76:     Subset(dat, fres)
77:
78:     res<-(split(dat, fres))$`mySampleFilter+`
79:     write.FCS(res, "PARAQ0 2A.FCS",what="integer" )
80:     splom(res)
81:   }
82:
83:
84: ######################################################
85: ### chunk :Leer el flowSet
86: ######################################################
87:
88:  ##Lista de Experimentos
89:  ExpList <- "full.annotation.txt"
90:
91:
92:   ### Descripción de los fluorocromos de cada panel
93:   panel.desc <- list()
94:   panel.desc[[1]] <- c("FSC", "SSC", "FL1.DCF"   ,  "FL3.TMRM", "FL6.Indo1",
    "FL8.BDP675")
95:   panel.desc[[2]] <- c("FSC", "SSC", "FL1.DIBAC3",  "FL3.MTSX", "FL6.MCB"  ,
    "FL8.BDP675")
96:   panel.desc[[3]] <- c("FSC", "SSC", "FL1.DAF"   ,  "FL3.TMRM", "FL6.MCB"  ,
    "FL8.BDP675")
97:   for (r in 1:length(panel.desc)){names(panel.desc[[r]]) <- c("FSC", "SSC", "FL1"
    ,  "FL3", "FL6", "FL8")}
98:
99:
100:   flowData <- read.flowSet(path = ".", phenoData = ExpList,
    transformation="linearize")
101:   sampleNames(flowData) <- as.character((pData(flowData)[, "Id"]))
102:   print("###############Experimentos###############")
103:   Titulos<-sampleNames(flowData)
104:   Titulos
105:   pData(phenoData(flowData))
106:   head(pData(flowData))
107:   sampleNames(flowData)
108:   Tclist<- sampleNames(flowData)
109:   factores <-pData(flowData)
110:   columnas<-colnames(flowData[[1]])
111:   factores <-pData(flowData)
112:   Channels<-colnames(flowData[[1]])
113:
114:
115:
116:   ######################################################
117:   ### chunk : Selección de la población de células
118:   ###     viables con filtros rectagulares
119:   ###     y de densidad.
120:   ######################################################
121:
122:
123:   print("
124:   ######################################################################
125:   ### chunk : Definición de Filtros para selección de las celulas vivas
126:   ######################################################################")
127:
128:   ######################################################################
129:   ##### Configuracion de filtro rectangular
130:   print(minFSC <- 500)
131:   print(maxFSC <- 2200)
```

```
132:   print(minSSC <- 200)
133:   print(maxSSC <- 2000)
134:   rectGate <- rectangleGate(filterId="rectangleGate",
135:            "FSC"=c(minFSC , maxFSC),
136:            "SSC"=c(minSSC , maxSSC))
137:
138:   ######################################################################
139:   ##### Configuracion de filtro morphGateScale
140:   #####  norm2filter
141:   print(morphGateScale <- 2)
142:     morphGate <-norm2Filter(filterId = "MorphologyGate",
143:                "FSC",  "SSC",
144:                scale = morphGateScale )
145:   ######################################################################
146:
147:
148:   ##############################################################################
149:   print("
150:   ##############################################################################
151:   ### Seleccion de positivos en el flowSet
152:   ##############################################################################
153:   ##############################################################################")
154:   ###filtramos primero con rectGate
155:   ##############################################################################
156:   frcfs <- filter(flowData, rectGate)
157:   rcPos.flowData<-Subset(flowData, rectGate)
158:   if (ifDebug){
159:     prepareOutput(mult=0.8)
160:     print(xyplot(SSC ~ FSC , rcPos.flowData,smooth=TRUE, filter=rectGate ,
161:       layout=c(6,1), subset = (CTox == 0), xlab="",
162:       scales = list(x = list(axs = "i"), y = list(draw = FALSE)),
163:       main=paste("Seleccion- minFSC:", minFSC,
164:                " maxFSC:", maxFSC,
165:                " minSSC:", minSSC,
166:                " maxSSC:", maxSSC)))
167:       }
168:   ### Seleccion de la población con mayor densidad.
169:   ### con un filtro norm2Filter
170:        ##############################################################################
171:   fPosfs <- filter(rcPos.flowData, morphGate )
172:   Pos.flowData  <-Subset(rcPos.flowData, morphGate )
173:
174:   if (ifDebug){
175:     prepareOutput(mult=0.8)
176:     print(xyplot(SSC ~ FSC , rcPos.flowData ,smooth=TRUE, filter=morphGate ,
177:       layout=c(6,1), subset = (CTox == 0), xlab="",
178:       scales = list(x = list(axs = "i"), y = list(draw = FALSE)),
179:       main=paste("Seleccion- morphGateScale <- ", morphGateScale )))
180:       }
181:
182:
183:
184:
185:
186: ####################################################
187: ### chunk : ## Separar por paneles.
188: ####################################################
189:
190:   ## Separamos el flowSet por Paneles
191:   splitPanelsFlowData <- split(Pos.flowData, factores[,"panel"])
192:   panels <- names(splitPanelsFlowData )
193:   print(paste("Paneles: ", panels ))
194:
195:   ## cada panel es un elemento de la lista
196:   ## splitPanelsFlowData
197:
198:
199:   ####################################################
```

```
200:   ### chunk : ## Analisis de densidad
201:   ######################################################
202:
203:
204:
205:   ## Iteramos para cada panel ....
206:   for (panelId in 1:length(panels ) ){
207:
208:   print(paste("Iterando panel: ", panels[panelId ] ))
209:   PanelFlowData <- splitPanelsFlowData [[panelId]]
210:   factoresSplit <-pData(PanelFlowData )
211:   ### Los controles son los que no tienen toxico CTox=0
212:     splitFlowData <- split(PanelFlowData , factoresSplit [,"CTox"])
213:     print(names(splitFlowData ))
214:     ControlFlowData <- splitFlowData[[1]]
215:     print(sampleNames(ControlFlowData ))
216:
217:
218:
219:     trellis.par.set(theme = col.whitebg())
220:     lw <- list(ylab.axis.padding = list(x = 0.5), left.padding = list(x = 0.5,
221:        units = "inches"), right.padding = list(x = 0, units = "inches"),
222:        panel = list(x = 1.4, units = "inches"))
223:     lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
224:       list(x = 0, units = "inches"), panel = list(x = 4, units = "inches"))
225:
226:     lattice.options(layout.widths = lw, layout.heights = lh,las=2)
227:
228:     X11(width = 12, height = 10, pointsize = 12)
229:     print(densityplot( ~ `FL1` + `FL3` +  `FL6` + `FL8` ,
230:         ControlFlowData , overlap=0.5,  main=paste("Panel ",panelId  )),
    darg=c(bw=0.2, n=200))
231:
232:     X11()
233:
234:     trellis.par.set(theme = col.whitebg())
235:     lw <- list(ylab.axis.padding = list(x = 0.5), left.padding = list(x = 0.5,
236:        units = "inches"), right.padding = list(x = 0, units = "inches"),
237:        panel = list(x = 1.4, units = "inches"))
238:     lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
239:       list(x = 0, units = "inches"), panel = list(x = 6, units = "inches"))
240:
241:     lattice.options(layout.widths = lw, layout.heights = lh,las=2)
242:
243:     X11(width = 50, height = 35, pointsize = 12)
244:     print(densityplot(~  `FL1` + `FL3` +  `FL6` + `FL8` ,
245:         PanelFlowData , overlap=0.1,  main=paste("Panel ",panelId  )),
    darg=c(bw=0.2))
246:
247:     fDensityTest2(ControlFlowData , texto="rcPosTFS", iminVs=0, imaxVs=2500,
248:                   intMax=0.04,isXlog=TRUE)
249:
250:   };## fin de la iteracion por el panel
251:
252:
253:
254:
    print("############################################################################################ )
255:   print("#Medias y medianas de los frames  en escala lineal")
256:
    print("############################################################################################ )
257:   print("#Calculamos las medianas de los experimentos")
258:
259:   TMedians.lin <- NULL
260:   data.TMedians.lin<- NULL
261:   MediansData.TMedians.lin<- NULL
262:
263:   TMeans.lin<- NULL
```

```
264:   Data.TMedians.lin <- NULL
265:   MediansData.lin<- NULL
266:   ########## Cálculo de medianas ####################################
267:   TMedians.lin<-as.matrix(fsApply(Pos.flowData, each_col, median))
268:   TMedians.lin
269:   data.TMedians.lin<-data.frame(Files = rownames(TMedians.lin),
270:                       "FSC" = TMedians.lin[, 1],
271:              "SSC" = TMedians.lin[, 2],
272:              "FL1" = TMedians.lin[, 3],
273:              "FL3" = TMedians.lin[, 4],
274:              "FL6" = TMedians.lin[, 5],
275:              "FL8" = TMedians.lin[, 6])
276:
277:   factores.ord <- NULL
278:   factores.ord <- pData(phenoData(Pos.flowData))
279:
280:
281:
282:   MediansData.lin<- cbind(data.TMedians.lin,factores.ord )
283:   finMediansData.lin<- MediansData.lin
284:
285:
286:    print("##Guardamos a disco la matriz de medianas")
287:    txtFilename <- paste(dataDirOut,"/" , ExpList ,".MatrizMedianasOriginaleslin.",
    ".txt", sep="")
288:    write.table(MediansData.lin,file=txtFilename ,row.names=TRUE,sep="\t",eol="\n",
    dec = ",")
289:    print(txtFilename )
290:
291:
292:   ########## Cálculo de medias ####################################
293:   TMeans.lin<-as.matrix(fsApply(Pos.flowData, each_col, mean))
294:   TMeans.lin
295:   data.TMeans.lin<-data.frame(Files = rownames(TMeans.lin),
296:                       "FSC" = TMeans.lin[, 1],
297:              "SSC" = TMeans.lin[, 2],
298:              "FL1" = TMeans.lin[, 3],
299:              "FL3" = TMeans.lin[, 4],
300:              "FL6" = TMeans.lin[, 5],
301:              "FL8" = TMeans.lin[, 6])
302:   factores.ord <- NULL
303:   factores.ord <- pData(phenoData(Pos.flowData))
304:
305:
306:
307:   MeansData.lin<- cbind(data.TMeans.lin,factores.ord)
308:
309:   finMeansData.lin<- MeansData.lin
310:
311:    print("##Guardamos a disco la matriz de medias")
312:    txtFilename <- paste(dataDirOut,"/" , ExpList ,".MatrizMediasOriginaleslin.",
    ".txt", sep="")
313:    write.table(MeansData.lin,file=txtFilename ,row.names=TRUE,sep="\t",eol="\r\n",
    dec = ",")
314:    print(txtFilename )
315:
    ##############################################################################################
316:
317:
318:
319:
320:
321:
322:
323:   ##########################################################################
324:   ### chunk: COMPENSACION
325:   ##########################################################################
326:   # El calculo se realiza con las celulas seleccionadas en eslala lineal##")
```

```
327:
328:
329:   # Cargar la matriz de compensacion ¡¡¡ Se compensan solo los full stain
330:
331:
332:
333:   cPanList <- list()
334:   ## Iteramos para cada panel ....
335:   for (panelId in 1:length(panels ) ){
336:   ## debug panelId =1
337:
338:
339:   print(paste("Iterando panel: ", panels[panelId ] ))
340:   pantocomp <- NULL
341:   pantocomp <- splitPanelsFlowData [[panelId]]
342:   factSplit <- pData(pantocomp )
343:   forCompensate <- which(factSplit[,"StainId"]=="A")
344:   noCompensate <- which(factSplit[,"StainId"]!="A")
345:
346:   SplitForComp <- pantocomp [forCompensate]
347:   SplitForNoComp <- pantocomp [noCompensate ]
348:
349:   matrixCompName <- paste(dataDir,"/" , "MatrizCompensacionPnl", panelId
350:                                       , ".txt", sep="")
351:
352:   spillmat <- t(read.table(matrixCompName , header = TRUE,sep="\t",dec = ","))
353:   print(spillmat )
354:   SplitForComp  <- compensate(SplitForComp , spillmat)
355:   finalFlowSet <- rbind2(SplitForComp ,SplitForNoComp  )
356:   sampleNames(finalFlowSet ) <- as.character((pData(finalFlowSet )[, "Id"]))
357:   cPanList[[panelId]] <- finalFlowSet
358:
     print("################################################################################################ )
359:   }
360:   cPos.flowData.N.12 <- rbind2(cPanList[[1]],cPanList[[2]])
361:   cPos.flowData.N <- rbind2(cPos.flowData.N.12,cPanList[[3]])
362:
363:   sampleNames(cPos.flowData.N) <- as.character((pData(cPos.flowData.N)[, "Id"]))
364:   summary(cPos.flowData.N)
365:   viewPars(cPos.flowData.N)
366:
367:
368:
     print("################################################################################################ )
369:   print("#Medias y medianas de los frames compensados")
370:
     print("################################################################################################ )
371:   print("#Calculamos las medianas de los experimentos")
372:
373:   cTMedians.N <- NULL
374:   cdata.TMedians.N<- NULL
375:   cMediansData.TMedians.N<- NULL
376:
377:   cTMeans.N<- NULL
378:   cData.TMedians.N <- NULL
379:   cMediansData.N<- NULL
380:   ########## Cálculo de medianas ######################################
381:   cTMedians.N<-as.matrix(fsApply(cPos.flowData.N, each_col, median))
382:   cTMedians.N
383:   head(cTMedians.N)
384:   cdata.TMedians.N<-data.frame(Files = rownames(cTMedians.N),
385:                         "FSC" = cTMedians.N[, 1],
386:              "SSC" = cTMedians.N[, 2],
387:              "FL1" = cTMedians.N[, 3],
388:              "FL3" = cTMedians.N[, 4],
389:              "FL6" = cTMedians.N[, 5],
390:              "FL8" = cTMedians.N[, 6])
391:
```

```
392:   factores.ord <- NULL
393:   factores.ord <- data.frame(pData(phenoData(cPos.flowData.N)))
394:
395:
396:   cMediansData.N<- cbind(cdata.TMedians.N,factores.ord )
397:
398:   cfinMediansData.N<- cMediansData.N
399:
400:
401:    print("##Guardamos a disco la matriz de medianas")
402:    txtFilename <- paste(dataDirOut,"/" , ExpList ,".MatrizMedianasCompensadaslin.",
    ".txt", sep="")
403:    write.table(cMediansData.N ,file=txtFilename ,row.names=TRUE,sep="\t",eol="\n",
    dec = ",")
404:    print(txtFilename )
405:
406:   summary(cPos.flowData.N["3B-NOR001"])
407:       summary(cPos.flowData.N1["3B-NOR001"])
408:   summary(flowData["3B-NOR001"])
409:   summary(Pos.flowData["3B-NOR001"])
410:
411:   ########## Cálculo de medias ####################################
412:   cTMeans.N<-as.matrix(fsApply(cPos.flowData.N, each_col, mean))
413:   cTMeans.N
414:   cdata.TMeans.N<-data.frame(Files = rownames(cTMeans.N),
415:                       "FSC" = cTMeans.N[, 1],
416:              "SSC" = cTMeans.N[, 2],
417:              "FL1" = cTMeans.N[, 3],
418:              "FL3" = cTMeans.N[, 4],
419:              "FL6" = cTMeans.N[, 5],
420:              "FL8" = cTMeans.N[, 6])
421:   factores.ord <- NULL
422:   factores.ord <- data.frame(pData(phenoData(cPos.flowData.N)))
423:
424:   cMeansData.N <- NULL
425:   cMeansData.N<- cbind(cdata.TMeans.N,factores.ord)
426:
427:   cfinMeansData.N <- NULL
428:   cfinMeansData.N<- cMeansData.N
429:
430:
431:    print("##Guardamos a disco la matriz de medias")
432:    txtFilename <- paste(dataDirOut,"/" , ExpList ,".MatrizMediasCompensadaslin.",
    ".txt", sep="")
433:    write.table(cMeansData.N ,file=txtFilename ,row.names=TRUE,sep="\t",eol="\r\n",
    dec = ",")
434:    print(txtFilename )
435:
    ##############################################################################################
436:
437:
438:
439:   ### Calcular el ratio de variación respecto del control
440:
441:   controles <- cMediansData.N[ which (cMediansData.N["CTox"]==0), ]
442:
443:   Expers <- cMediansData.N[ which (cMediansData.N["CTox"]!=0), ]
444:   ContrId <- Expers[,"name"]
445:   Expers <- cbind(Expers, ContrId )
446:
447:   for (i in 1:nrow(Expers)){
448:
449:       ## debug i=1
450:       st <- Expers [i,"StainId"]
451:       tx <- Expers [i,"Tox"]
452:       pn <- Expers [i,"panel"]
453:       isControl <- which (controles["StainId"]==st  & controles["Tox"]==tx &
    controles["panel"]==pn )
```

```
454:       control <- controles[ isControl  , ]
455:       Expers [i,"FL1"] <- (Expers [i,"FL1"] / control[1,"FL1"])
456:       Expers [i,"FL3"] <- (Expers [i,"FL3"] / control[1,"FL3"])
457:       Expers [i,"FL6"] <- (Expers [i,"FL6"] / control[1,"FL6"])
458:       Expers [i,"FL8"] <- (Expers [i,"FL8"] / control[1,"FL8"])
459:       Expers [i,"ContrId"] <- as.character(control[1,"Id"])
460:       }
461:
462:
463:
464:    print("##Guardamos a disco la matriz de variaciones")
465:    txtFilename <- paste(dataDirOut,"/" , ExpList ,
    ".MatrizVariacionesCompensadaslin.", ".txt", sep="")
466:    write.table(Expers,file=txtFilename ,row.names=TRUE,sep="\t",eol="\n",dec = ",")
467:    print(txtFilename )
468:
469:
470:
471: ##############################################################################################
    ##############################
472:
473:
474:   Titulos
475:   factores
476:   columnas
477:   Channels
478:   fluoChannels <- Channels[-c(1:2)]
479:   Toxs <- c("Cd", "Hg")
480:   stainsInt <- as.matrix(c("A", "B", "C", "D", "E"))
481:   rownames(stainsInt ) <- c("full",  "FL1", "FL3", "FL6", "FL8")
482:   revStains <- as.matrix(c("full",  "FL1", "FL3", "FL6", "FL8"))
483:   rownames(revStains ) <- c("A", "B", "C", "D", "E")
484:   descript.fluoPanel <- list()
485:   descript.fluoPanel[[1]] <-c("FSC", "SSC",
486:             "FL1.DCF - Actividad peroxidasa",
487:             "FL3.TMRM - Potencial de membrana mitocondrial",
488:             "FL6.Indo1 - Calcio intracelular",
489:             "FL8.BDP675 - Peroxidación lipídica")
490:
491:   descript.fluoPanel[[2]] <- c("FSC", "SSC",
492:             "FL1.DIBAC3 - Potencial de membrana plasmática",
493:             "FL3.MTSX - Superoxido mitocondrial",
494:             "FL6.MCB - Tioles libres - glutation",
495:             "FL8.BDP675 - Peroxidación lipídica")
496:
497:   descript.fluoPanel[[3]] <- c("FSC", "SSC",
498:             "FL1.DAF - Niveles de óxido nítrico",
499:             "FL3.TMRM - Potencial de membrana mitocondrial",
500:             "FL6.MCB - Tioles libres - glutation",
501:             "FL8.BDP675 - Peroxidación lipídica")
502:
503:
504:   for (r in 1:length(descript.fluoPanel)) {
505:       names(descript.fluoPanel[[r]]) <- c("FSC", "SSC", "FL1", "FL3", "FL6", "FL8")}
506:
507:   ########################################################
508:   ### Graficos  Graficos de % variacion
509:   ########################################################
510:   porcVar <- Expers
511:   for (panelId in 1:length(panels) ){
512:
513:       x11(width=150, height=100)
514:       def.par <- par(no.readonly = TRUE) # save default, for resetting...
515:           mat<-matrix(c(1:4), 2, 2,  byrow=TRUE)
516:           layout(mat, widths = rep(1, ncol(mat)),
517:           heights = rep(1, nrow(mat)), respect = FALSE)
518:     par(mar=c(8.5,5.5,8.8,2.5))
519:
```

```
520:
521:     ###For Debug panelId<-2
522:     print(paste("Iterando panel: ", panels[panelId ] ))
523:
524:     for (st in stainsInt[2:5]){
525:
526:       #fordebug st<-"FL3"
527:       Cualesfullstains <- which(porcVar [,"StainId"]=="A" &
528:                 porcVar [,"panel"]==panelId & porcVar [,"FL3"]<10 )
529:       fullstains <- porcVar [Cualesfullstains ,]
530:
531:       cualestox <-  which(porcVar [,"panel"]==panelId &
532:               porcVar [,"StainId"]==st & porcVar [,"FL3"]<10)
533:       grupotox <- porcVar [cualestox ,]
534:
535:       grupo <- rbind(fullstains ,grupotox  )
536:       grupoOrdenado <- sort.data.frame(grupo, by= ~ "Tox")
537:       #print(grupoOrdenado )
538:       chann <- revStains[[st,1]]
539:       print(chann )
540:           tit <- descript.fluoPanel[[panelId ]][[chann ]]
541:       tity <- "Iobs/Icontrol"
542:
543:
544:       barplot(grupoOrdenado [,chann ],
545:           names.arg=rownames(grupoOrdenado ),
546:           ylab=tity, main=paste(tit, "-Pnl.", panelId , sep="" ),
547:           las=2, cex.names=1.4, cex.main=1.2)
548:
549:       abline( h=0 , col="blue")
550:       abline( h=1 , col="red")
551:       #abline( h=-15 , col="red")
552:
553:     }
554:   }
555:
556:
557:
558:   ##########################################################
559:   ### Graficos de Intensidad de mediana
560:   ##########################################################
561:   porcVar <- MediansData.lin
562:
563:
564:   for (panelId in 1:length(panels) ){
565:
566:   x11(width=120, height=160)
567:   def.par <- par(no.readonly = TRUE) # save default, for resetting...
568:       mat<-matrix(c(1:16), 4, 1,  byrow=TRUE)
569:       layout(mat, widths = rep(1, ncol(mat)),
570:       heights = rep(1, nrow(mat)), respect = FALSE)
571:   par(mar=c(4.5, 5.5, 3.5, 0.5))
572:     ###For Debug panelId<-2
573:     print(paste("Iterando panel: ", panels[panelId ] ))
574:
575:     for (st in stainsInt[2:5]){
576:       #fordebug st<-3
577:       Cualesfullstains <- which(porcVar [,"StainId"]=="A" &
578:                 porcVar [,"panel"]==panelId )
579:       fullstains <- porcVar [Cualesfullstains ,]
580:
581:       cualestox <-  which(porcVar [,"panel"]==panelId &
582:               porcVar [,"StainId"]==st  )
583:       grupotox <- porcVar [cualestox ,]
584:
585:       grupo <- rbind(fullstains ,grupotox  )
586:       grupoOrdenado <- sort.data.frame(grupo, by= ~ "Tox")
587:       #print(grupoOrdenado )
```

```
588:       chann <- revStains[[st,1]]
589:       print(chann )
590:           tit <- descript.fluoPanel[[panelId ]][[chann ]]
591:       tity <- "Int. Chann. num."
592:       barplot(grupoOrdenado [,chann ],
593:           names.arg=rownames(grupoOrdenado ),
594:           ylab=tity, main=paste(tit, " -Pnl.", panelId , sep="" ),
595:           las=2, cex.names=0.9, cex.main=1)
596:       abline( h=0 , col="blue")
597:       #abline( h=15 , col="red")
598:       #abline( h=-15 , col="red")
599:     }
600:   }
601:
602:
603:
604:
605:
606:
607:
608:
609:
```

```

  ### Libreria: utils.functions.R
  ### VERSION: 6.10.07.2010


#####################################################
### chunk: Load Libraries
#####################################################
 library(flowCore)
 library(flowQ)
 library(flowViz)
 library(flowStats)
 library(flowUtils)
 library(geneplotter)
 library(colorspace)
 library(grid)
 library(MASS)
 library(flowFP)
 library(Geneplotter)
 require(RColorBrewer)
 library(MASS)
 library("KernSmooth")

#####################################################
### chunk : Funcion viewPars
#####################################################

  viewPars <- function(viewParsFlowSet){
        print("############################################")
        for (frame in 1:length(viewParsFlowSet)){
          print(viewParsFlowSet[[frame ]])
          print("############################################")
        }}
  #viewPars(flowData)



  ######################################################################
  ### chunk : funciones de Transformacion antilogaritmica     ####
       ###  AllgT <- antilogTransform (transformId="Alog",a=4,r=4096)#
  ######################################################################

  #revierte la transformacion del amplificador logaritmico
  antilogTransform <- function(transformId, a=4,r=4096){
        t = new("transform", .Data = function(x) 10^(x*a/r))
        t@transformationId = transformId
        t
        }

  AllgT <- antilogTransform (transformId="Alog",a=4,r=4096)


  ######################################################################
  ### chunk : funciones de Transformacion linlog              ####
  ### linlogTransform = function(transformationId, median = 0, dist = 1, ...)#
  ######################################################################


  linlogTransform = function(transformationId, median = 0, dist = 1, ...)
  {
     tr <- new("transform", .Data = function(x) {
     idx = which(x <= median + dist)
     idx2 = which(x > median + dist)
     if (length(idx2) > 0) {
       x[idx2] = log10(x[idx2] - median) - log10(dist/exp(1))
       }
     if (length(idx) > 0) {
       x[idx] = 1/dist * log10(exp(1)) * (x[idx] - median)
```

```
69:         }
70:         x
71:       })
72:       tr@transformationId = transformationId
73:       tr
74:   }
75:
76:   lnlgT <- linlogTransform(transformationId = "splitscale",
77:               median = 0, dist = 100)
78:
79:
80:   fnlinlog <- function(x,medianx,distx ) {
81:       idx = which(x <= medianx + distx)
82:       idx2 = which(x > medianx + distx)
83:       if (length(idx2) > 0) {
84:         x[idx2] = log10(x[idx2] - medianx) - log10(distx/exp(1))
85:         }
86:       if (length(idx) > 0) {
87:         x[idx] = 1/dist * log10(exp(1)) * (x[idx] - medianx)
88:         }
89:         x
90:       }
91:
92:   ## See test.linlog.R
93:
94:
95:
96:   ##########################################################################
97:   ### chunk : Reescalado de flowsets                                       #
98:   ###   rescaling <- function(resFlowSet, minScale=0, maxScale=10000)#
99:   ##########################################################################
100:
101:  ###modificar el escalado
102:  ###Reajustamos los maximos y minimos para mejorar la
103:  ### visualizacion
104:
105:  rescaling <- function(resFlowSet, minScale=0, maxScale=10000){
106:  nFrames <- length(resFlowSet)
107:  # Solo cambiamos los de los canales de fluorescencia
108:  for (i in 1:nFrames)
109:    {
110:      ###maxRange
111:          pData(parameters(resFlowSet[[i]]))[5][3:6,1]<- maxScale
112:      ###minRange
113:      pData(parameters(resFlowSet[[i]]))[4][3:6,1]<- minScale
114:    }
115:  resFlowSet
116:  }
117:
118:
119:
120:  ##########################################################################
121:  ### chunk : Modificar la descripcion de marcadores en cada canal   #
122:  ###   changeDescNames <- function(resFlowSet, newNames)#
123:  ##########################################################################
124:  changeDescNames <- function(resFlowSet, newNames){
125:  nFrames <- length(resFlowSet)
126:
127:  for (i in 1:nFrames)
128:    {
129:
130:          namesDesc <- pData(parameters(resFlowSet[[i]]))$desc
131:      namesDesc <- newNames
132:      pData(parameters(resFlowSet[[i]]))$desc <- namesDesc
133:    }
134:  resFlowSet
135:  }
136:
```

```
137:   ##########################################################################
138:   ### chunk : Modificar nombres de       marcadores en cada canal   #
139:   ###   changeFluoNames <- function(resFlowSet, newNames)#
140:   ##########################################################################
141:   changeFluoNames <- function(resFlowSet, newNames){
142:   nFrames <- length(resFlowSet)
143:   colnames(resFlowSet)<-newNames
144:   for (i in 1:nFrames)
145:     {
146:       namesDesc <- pData(parameters(resFlowSet[[i]]))$name
147:       namesDesc <- newNames
148:       pData(parameters(resFlowSet[[i]]))$name<- namesDesc
149:     }
150:   resFlowSet
151:   }
152:
153:
154: ######################################################################
155: ### chunk : VisualizarHistogramas                              ##
156: ### En funciones.propias.R                                      ##
157: ### visualizar histogramas                                   ##
158: ######################################################################
159:
160:
161:   VisualizarHistogramas <- function(ff,main="Hist", xmin=0, xmax=2000)
162:   {
163:     Expresionff <-(exprs(ff))
164:     X11(width = 14, height = 7)
165:     mat<-matrix(c(1:6), 2, 3,  byrow=TRUE)
166:     layout(mat, widths = rep(1, ncol(mat)),heights = rep(1, nrow(mat)), respect =
    FALSE)
167:     for (i in 1:6){
168:       hist(Expresionff [,i], breaks=4096, col="blue",
169:            xlab=(colnames(Expresionff )[i]), ylab="",
170:             main=main,xlim = c(xmin,xmax))}
171:   }
172:
173:
174:   ##### Funcion fDensityTest## Genera un output con una comparacion de las
175:   # funciona de densidad de los flowSet.
176:   fDensityTest2 <- function(FlowSetImp, texto="DensFunTest", showSt=TRUE,
177:               iminVs=0, imaxVs=3500,intMax=1,isXlog=FALSE)
178:   {
179:
180:     chkFlowData <- FlowSetImp
181:
182:
183:     x11(width=120, height=110)
184:     mat<-matrix(c(1:6), 2, 3,  byrow=TRUE)
185:     layout(mat, widths = rep(1, ncol(mat)),heights = rep(1, nrow(mat)), respect =
    FALSE)
186:
187:     nombrescol   <- colnames(chkFlowData[[1]])
188:     print(nombrescol)
189:     pData(chkFlowData[1])
190:     nombrescol
191:     nExp <- length(chkFlowData)
192:     nChann <- length(nombrescol    )
193:       leyenda<-sampleNames(chkFlowData)
194:       print(leyenda)
195:        cls <- densCols(c(1:40), nbin=400,colramp = colorRampPalette(rainbow(20)))
196:
197:             cl<-cls
198:     for (icanal in 1:nChann)  {
199:     for (iexp in 1:nExp ) {
200:
201:         if (icanal<3){
202:             minVs=0
```

```
203:             maxVs=3500
204:           } else {
205:             minVs=iminVs
206:             maxVs=imaxVs
207:           }
208:
209:
210:         matrixExpresion <-(exprs(chkFlowData[[iexp]]))
211:         media<-mean(matrixExpresion[,icanal])
212:         mediana<-median(matrixExpresion[,icanal])
213:         valdens <- density(matrixExpresion[,icanal ])
214:         if (iexp==1){
215:           print(plot(valdens ,xlim=c(minVs,maxVs),
216:                   ylim=c(0,intMax), ann=TRUE,
217:                   xlab=nombrescol[icanal],
218:                 xlog=isXlog,
219:                           main=nombrescol[icanal]))
220:         } else {
221:           print(lines(valdens , col=cl[iexp]))}
222:         if (showSt){
223:           print(points(mediana, intMax-0.005, pch=2, col=cl[iexp], cex=2, bty="n"))
224:           print(points(media, intMax-0.008, pch=3, col=cl[iexp], cex=2, bty="n"))
225:           }
226:           xmn <- sampleNames(chkFlowData[iexp])
227:         #SI QUEREMOS MEDIANS DATA
228:           xm <- mediana
229:           coor <-rbind(c(xm,0.003),c(xm,0.0038))
230:           print(coor)
231:           print(lines(coor, col=cl[iexp],lwd=1.5))
232:         }
233:         print(legend("topright", leyenda, pch=19, col=cl[1:iexp], cex=0.8))
234:     }
235:
236:
237:   }
238:
239:
240:
241:
242:     ## fDensityTest(cPosTFS, "Compensados.Vivas.","CTox")
243:  ##############################################################################
244:
245:
246:
247:   ##### Funcion fDensityTestSSCFSC ## Genera un output con una comparacion de las
248:   ##### funciona de densidad de los flowSet. en Los canales FSC y SSC
249:
250:   fDensityTestSSCFSC <- function(FlowSetImp, texto="DensFunTest", factorAg="CTox",
    Stainf="A",icanal=1,
251:               iminVs=0, imaxVs=3500,intMax=1,isXlog=FALSE, showSt=FALSE)
252:   {
253:     locFactor <- factorAg
254:     anlFlowData <- FlowSetImp
255:     factores <- pData(anlFlowData )
256:     print("#######################################")
257:     print(factores)
258:     print("#######################################")
259:     chkFFlowData <- split(anlFlowData , factores[,locFactor])
260:     IdFactores <- names(chkFFlowData)
261:     print("#######################################")
262:     print(IdFactores)
263:     print("#######################################")
264:
265:     pdfName<- paste(dataDirOut,"/" , ExpList ,".", locFactor,".",
266:                                ".", texto , ".", "- Compdens%03d.pdf", sep="")
267:
268:
269:     print(pdfName)
```

```
270:
271:     grupo <- Stainf
272:
273:     chkFlowData <- chkFFlowData[[grupo]]
274:     print("######ITERANDO FACTORES ################################")
275:     print(paste("#########GRUPO: ", grupo ))
276:     nombrescol   <- colnames(chkFlowData[[1]])
277:     print(nombrescol)
278:     pData(chkFlowData[1])
279:     nombrescol
280:     nExp <- length(chkFlowData)
281:     nChann <- length(nombrescol    )
282:       leyenda<-sampleNames(chkFlowData)
283:       print(leyenda)
284:         YlOrBr <- c("#FED98E", "#FE9929", "#D95F0E", "#993404")
285:         clramp <- colorRampPalette(YlOrBr, space = "Lab")
286:             cls <- densCols(c(1:40), nbin=40,colramp = clramp )
287:       cl<- c("black", "blue", "green", "violet", "red",
288:                "#FED98E", "#FE9929", "#D95F0E", "#993404")
289:
290:
291:     for (iexp in 1:nExp ) {
292:
293:
294:             minVs=iminVs
295:             maxVs=imaxVs
296:
297:
298:
299:         matrixExpresion <-(exprs(chkFlowData[[iexp]]))
300:         media<-mean(matrixExpresion[,icanal])
301:         mediana<-median(matrixExpresion[,icanal])
302:         valdens <- density(matrixExpresion[,icanal ])
303:         if (iexp==1){
304:           print(plot(valdens ,xlim=c(minVs,maxVs),
305:                   ylim=c(0,intMax), ann=TRUE,
306:                   xlab=nombrescol[icanal],
307:                  ylab="",
308:                  xlog=isXlog,
309:                            main=""))
310:         } else {
311:         print(lines(valdens , col=cl[iexp]))
312:         if (showSt){
313:           print(points(mediana, intMax-0.005, pch=2, col=cl[iexp], cex=2, bty="n"))
314:           print(points(media, intMax-0.008, pch=3, col=cl[iexp], cex=2, bty="n"))
315:           }
316:           xmn <- sampleNames(chkFlowData[iexp])
317:         #SI QUEREMOS MEDIANS DATA
318:           xm <- mediana
319:           coor <-rbind(c(xm,0.0035),c(xm,0.0038))
320:           print(coor)
321:           print(lines(coor, col=cl[iexp],lwd=1.5))
322:
323:         print(legend("topright", leyenda, pch=19, col=cl[1:iexp], cex=0.9))
324:           }
325:
326:     #dev.off()
327:   }
328: }
329:
330:     ## fDensityTest(cPosTFS, "Compensados.Vivas.","CTox")
331:  ##############################################################################
332:
333:
334:
335:
336:
337: ####################################################################
```

```
338: ######################################################################
339: ### chunk : GRAFICOS DE DENSIDAD.          ##
340: ### En funciones.propias.R             ##
341: ##
342: ######################################################################
343:
344: ####################GRAFICOS DE DENSIDAD#############################
345:   pdfName<- paste(dataDirOut,"/" , ExpList , "- CompDensityMed%03d.pdf", sep="")
346:   #pdf(file=pdfName, onefile=TRUE, width = 15, height=10 )
347:
348:
349: visualizarDensityRec <- function (vsFlowSet, main="Graficos PDF", xlim=c(0,4500)){
350:
351:   trellis.par.set(theme = col.whitebg())
352:    lw <- list(ylab.axis.padding = list(x = 0.5), left.padding = list(x = 0.5,
353:      units = "inches"), right.padding = list(x = 0, units = "inches"),
354:      panel = list(x = 1.4, units = "inches"))
355:   lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
356:      list(x = 0, units = "inches"), panel = list(x = 4, units = "inches"))
357:
358:   lattice.options(layout.widths = lw, layout.heights = lh,las=2)
359:   X11(width = 14, height = 7, pointsize = 12)
360:
361:   print(densityplot(~ `FL1` + `FL3` +  `FL6` + `FL8` ,xlim=xlim,
362:         vsFlowSet, overlap=0.1, main=main))
363: }
364:
365:
366: visualizarDensity <- function (vsFlowSet, main="Graficos PDF"){
367:
368:   trellis.par.set(theme = col.whitebg())
369:    lw <- list(ylab.axis.padding = list(x = 0.5), left.padding = list(x = 0.5,
370:      units = "inches"), right.padding = list(x = 0, units = "inches"),
371:      panel = list(x = 1.4, units = "inches"))
372:   lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
373:      list(x = 0, units = "inches"), panel = list(x = 4, units = "inches"))
374:
375:   lattice.options(layout.widths = lw, layout.heights = lh,las=2)
376:
377:   X11(width = 14, height = 10, pointsize = 12)
378:   print(densityplot(~ `SSC` + `FSC` + `FL1` + `FL3` +  `FL6` + `FL8` ,
379:         vsFlowSet, overlap=0.1, main=main), darg=c(bw=0.2))
380: }
381:
382:
383:   ######################################################################
384:   ### Función : scatersComparados       ##
385:   ######################################################################
386:
387:
388:   scatersComparados <- function(noCompFlowSet,CompFlowSet,mult=1,
389:             MediansData=MediansData,Texto="ScatersComparados"){
390:
391:   trellis.par.set(theme = col.whitebg())
392:   lw <- list(ylab.axis.padding = list(x = 0.5), left.padding = list(x = 0.5,
393:     units = "inches"), right.padding = list(x = 0, units = "inches"),
394:     panel = list(x = 1*mult, units = "inches"))
395:   lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
396:     list(x = 0, units = "inches"), panel = list(x = 1*mult, units = "inches"))
397:
398:     lattice.options(layout.widths = lw, layout.heights = lh)
399:
400:
401:     mt = trellis.par.get("par.main.text")
402:     mt$cex = 0.7
403:       trellis.par.set("par.main.text",mt)
404:     #X11(width = 15, height = 7, pointsize = 12)
405:      tp1 <- xyplot(`FL1` ~ `FL3` , noCompFlowSet[1:6] ,
```

```
406:       main=paste("Scater comparado - Antes compensación ", Texto),
407:           pch = 21, cex = 0.1,
408:       layout = c(6, 1),
409:             panel = function(x, frames, channel.x, channel.y, ...) {
410:               nm <- as.character(x)
411:       print(nm)
412:               x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
413:               y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
414:       my.panel(x, y, frame = frames[[nm]], ...)
415:       panel.abline(v= median(x),col="red", alpha=0.8)
416:       panel.abline(h= median(y),col="green", alpha=0.8)
417:       panel.abline(v= mean(x),col="yellow", alpha=0.8)
418:       panel.abline(h= mean(y),col="yellow", alpha=0.8)
419:
420:             })
421:
422:       tp2 <- xyplot(`FL1` ~ `FL3` , CompFlowSet[1:6],
423:       main=paste("Scater comparado  - Después compensación", Texto),
424:
425:       pch = 21, cex = 0.1,
426:       layout = c(6, 1),
427:             panel = function(x, frames, channel.x, channel.y, ...) {
428:               nm <- as.character(x)
429:       print(nm)
430:               x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
431:               y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
432:       my.panel(x, y, frame = frames[[nm]], ...)
433:       panel.abline(v= median(x),col="red", alpha=0.8)
434:       panel.abline(h= median(y),col="green", alpha=0.8)
435:       panel.abline(v= mean(x),col="yellow", alpha=0.8)
436:       panel.abline(h= mean(y),col="yellow", alpha=0.8)
437:
438:             })
439:
440:       x11(width = 15, height = 7, pointsize = 12)
441:        print(plot(tp1, position = c(0, 0.4, 1 , 1), more = TRUE))
442:        print(plot(tp2, position = c(0, 0 , 1 , 0.5), more = FALSE))
443:
444:
445:     X11(width = 15, height = 10, pointsize = 12)
446:       tp1 <- print(xyplot(`FL6` ~ `FL8` ,noCompFlowSet[1:6] ,
447:       main=paste("Scater comparado - Antes compensación ",Texto),
448:           pch = 21, cex = 0.3,
449:       layout = c(6, 1),
450:             panel = function(x, frames, channel.x, channel.y, ...) {
451:               nm <- as.character(x)
452:       print(nm)
453:               x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
454:               y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
455:       my.panel(x, y, frame = frames[[nm]], ...)
456:       panel.abline(v= median(x),col="red", alpha=0.5)
457:       panel.abline(h= median(y),col="green", alpha=0.5)
458:       panel.abline(v= mean(x),col="yellow", alpha=0.8)
459:       panel.abline(h= mean(y),col="yellow", alpha=0.8)
460:
461:             }))
462:       tp2 <- xyplot(`FL6` ~ `FL8` , CompFlowSet[1:6],
463:       main=paste("Scater comparado  - Después compensación", Texto),
464:
465:       pch = 21, cex = 0.1,
466:       layout = c(6, 1),
467:             panel = function(x, frames, channel.x, channel.y, ...) {
468:               nm <- as.character(x)
469:       print(nm)
470:               x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
471:               y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
472:       my.panel(x, y, frame = frames[[nm]], ...)
473:       panel.abline(v= median(x),col="red", alpha=0.8)
```

```
474:       panel.abline(h= median(y),col="green", alpha=0.8)
475:       panel.abline(v= mean(x),col="yellow", alpha=0.8)
476:       panel.abline(h= mean(y),col="yellow", alpha=0.8)
477:       })
478:
479:       x11(width = 15, height = 7, pointsize = 12)
480:        print(plot(tp1, position = c(0, 0.4, 1 , 1), more = TRUE))
481:        print(plot(tp2, position = c(0, 0 , 1 , 0.5), more = FALSE))
482:
483:   #dev.off()
484:
485:   ##########################################################################3
486:
487:   }
488:
489:   ###################################################################
490:   ### Función : prepareOutput       ##
491:   ###################################################################
492:   ### Prepara el output por pantalla de scaters   ##
493:   ###################################################################
494:
495:   prepareOutput <- function(mult=1,width=90,height=50){
496:
497:   trellis.par <- NULL
498:   trellis.par.set(theme = col.whitebg())
499:   lw <- list(ylab.axis.padding = list(x = 0.4), left.padding = list(x = 0.4,
500:     units = "inches"), right.padding = list(x = 0, units = "inches"),
501:     panel = list(x = 1.9*mult, units = "inches"))
502:   lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
503:     list(x = 0, units = "inches"), panel = list(x = 1.9*mult, units = "inches"))
504:   lattice.options(layout.widths = lw, layout.heights = lh,las=2)
505:
506:   no <- TRUE
507:   if (no){
508:   my.panel <- function(x, y, ..., frame)
509:   {
510:         cols <- densCols(x, y, nbin=200,colramp = colorRampPalette(rainbow(10)))
511:         panel.xyplot(x, y,col=cols, ...)
512:
513:   }}
514:
515:   x11(width=width, height=height)
516:   }
517:
518:
519:   ###################################################################
520:   ### chunk : Graficos con seleccion de viables                ##
521:   ###################################################################
522:
523:
524:   ###################################################################
525:   ### Función : visualizarScatervivas.FL      ##
526:   ###################################################################
527:
528:   visualizarScatervivas.FL <- function(fcompTodas,fcompVivas,
529:                               main="Visualizacion",mult=1,my.layout=c(5,5)) {
530:
531:   trellis.par.set(theme = col.whitebg())
532:   lw <- list(ylab.axis.padding = list(x = 0.5), left.padding = list(x = 0.5,
533:     units = "inches"), right.padding = list(x = 0, units = "inches"),
534:     panel = list(x = 1*mult, units = "inches"))
535:   lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
536:     list(x = 0, units = "inches"), panel = list(x = 1*mult, units = "inches"))
537:
538:     lattice.options(layout.widths = lw, layout.heights = lh)
539:
540:   x11(width=90, height=50)
541:
```

```
  print( xyplot( FL1 ~ FSC , fcompTodas,smooth=TRUE, pch = 21, cex = 0.3,
    main=paste( "Seleccion- morphGateScale:", morphGateScale) ,
    layout=my.layout,
        panel = function(x, frames, channel.x, channel.y, ...) {
            nm <- as.character(x)
            print(nm)
            x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
            y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
    my.panel(x, y, frame = frames[[nm]], ...)
    print( x.intname <- as.character(channel.x))
    print( y.intname <- as.character(channel.y))
    xyVivas <- exprs(fcompVivas[[nm ]])
    panel.abline(v= median(xyVivas[,x.intname]),col="red", alpha=0.5)
    panel.abline(h= median(xyVivas[,y.intname]),col="green", alpha=0.5)
    panel.points(  xyVivas [,x.intname],  xyVivas [,y.intname] , col="blue"  ,
                   alpha=0.3, pch=".", cex=1)
    intMedianV <- median(xyVivas[,x.intname])
    maxMedianV <- max(xyVivas[,x.intname])
    intMedianH <- median(xyVivas[,y.intname])
    maxMedianH <- max(xyVivas[,y.intname])
    panel.text(maxMedianH ,maxMedianV,pos=1 ,
              format(intMedianV ,digits=5),col="red", cex=0.6)
    panel.text(maxMedianH ,maxMedianV,pos=2 ,
              format(intMedianH ,digits=5),col="green", cex=0.6)

        }))
  x11(width=90, height=50)


  print( xyplot(FL3 ~ FL1 , fcompTodas,smooth=TRUE, pch = 21, cex = 0.3,
    main=paste( "Seleccion- morphGateScale:", morphGateScale) ,
    layout=my.layout,

        panel = function(x, frames, channel.x, channel.y, ...) {
            nm <- as.character(x)
            print(nm)
            x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
            y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
    my.panel(x, y, frame = frames[[nm]], ...)
    print( x.intname <- as.character(channel.x))
    print( y.intname <- as.character(channel.y))
    xyVivas <- exprs(fcompVivas[[nm ]])
    panel.abline(v= median(xyVivas[,x.intname]),col="red", alpha=0.5)
    panel.abline(h= median(xyVivas[,y.intname]),col="green", alpha=0.5)
    panel.points(  xyVivas [,x.intname],  xyVivas [,y.intname] , col="blue"  ,
alpha=0.3, pch=".", cex=1)
         }))

  x11(width=90, height=50)


  print( xyplot(FL6 ~ FL1   , fcompTodas,smooth=TRUE, pch = 21, cex = 0.3,
    main=paste( "Seleccion- morphGateScale:", morphGateScale) ,
    layout=my.layout,

        panel = function(x, frames, channel.x, channel.y, ...) {
            nm <- as.character(x)
            print(nm)
            x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
            y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
    my.panel(x, y, frame = frames[[nm]], ...)
    print( x.intname <- as.character(channel.x))
    print( y.intname <- as.character(channel.y))
    xyVivas <- exprs(fcompVivas[[nm ]])
    panel.abline(v= median(xyVivas[,x.intname]),col="red", alpha=0.5)
    panel.abline(h= median(xyVivas[,y.intname]),col="green", alpha=0.5)
    panel.points(  xyVivas [,x.intname],  xyVivas [,y.intname] , col="blue"  ,
alpha=0.3, pch=".", cex=1)
```

```
608:          }))
609:
610:   x11(width=90, height=50)
611:
612:
613:   print( xyplot(FL8 ~ FL1 , fcompTodas,smooth=TRUE, pch = 21, cex = 0.3,
614:     main=paste( "Seleccion- morphGateScale:", morphGateScale) ,
615:     layout=my.layout,
616:
617:         panel = function(x, frames, channel.x, channel.y, ...) {
618:             nm <- as.character(x)
619:             print(nm)
620:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
621:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
622:     my.panel(x, y, frame = frames[[nm]], ...)
623:         print( x.intname <- as.character(channel.x))
624:     print( y.intname <- as.character(channel.y))
625:     xyVivas <- exprs(fcompVivas[[nm ]])
626:     panel.abline(v= median(xyVivas[,x.intname]),col="red", alpha=0.5)
627:     panel.abline(h= median(xyVivas[,y.intname]),col="green", alpha=0.5)
628:     panel.points(  xyVivas [,x.intname],  xyVivas [,y.intname] , col="blue"  ,
     alpha=0.3, pch=".", cex=1)
629:        }))
630:
631:   x11(width=90, height=50)
632:
633:
634:   print( xyplot( FL1 ~ FL3   , fcompTodas,smooth=TRUE, pch = 21, cex = 0.3,
635:     main=paste( "Seleccion- morphGateScale:", morphGateScale) ,
636:     layout=my.layout,
637:
638:         panel = function(x, frames, channel.x, channel.y, ...) {
639:             nm <- as.character(x)
640:             print(nm)
641:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
642:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
643:     my.panel(x, y, frame = frames[[nm]], ...)
644:         print( x.intname <- as.character(channel.x))
645:     print( y.intname <- as.character(channel.y))
646:     xyVivas <- exprs(fcompVivas[[nm ]])
647:     panel.abline(v= median(xyVivas[,x.intname]),col="red", alpha=0.5)
648:     panel.abline(h= median(xyVivas[,y.intname]),col="green", alpha=0.5)
649:     panel.points(  xyVivas [,x.intname],  xyVivas [,y.intname] , col="blue"  ,
     alpha=0.3, pch=".", cex=1)
650:         }))
651:   x11(width=90, height=50)
652:
653:
654:   print( xyplot(FL6 ~ FL3 , fcompTodas,smooth=TRUE, pch = 21, cex = 0.3,
655:     main=paste( "Seleccion- morphGateScale:", morphGateScale) ,
656:     layout=my.layout,
657:
658:         panel = function(x, frames, channel.x, channel.y, ...) {
659:             nm <- as.character(x)
660:             print(nm)
661:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
662:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
663:     my.panel(x, y, frame = frames[[nm]], ...)
664:         print( x.intname <- as.character(channel.x))
665:     print( y.intname <- as.character(channel.y))
666:     xyVivas <- exprs(fcompVivas[[nm ]])
667:     panel.abline(v= median(xyVivas[,x.intname]),col="red", alpha=0.5)
668:     panel.abline(h= median(xyVivas[,y.intname]),col="green", alpha=0.5)
669:     panel.points(  xyVivas [,x.intname],  xyVivas [,y.intname] , col="blue"  ,
     alpha=0.3, pch=".", cex=1)
670:
671:          }))
672:
```

```
673:   x11(width=90, height=50)
674:
675:
676:   print( xyplot(FL8 ~ FL3 , fcompTodas,smooth=TRUE, pch = 21, cex = 0.3,
677:     main=paste( "Seleccion- morphGateScale:", morphGateScale) ,
678:     layout=my.layout,
679:
680:         panel = function(x, frames, channel.x, channel.y, ...) {
681:             nm <- as.character(x)
682:             print(nm)
683:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
684:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
685:     my.panel(x, y, frame = frames[[nm]], ...)
686:         print( x.intname <- as.character(channel.x))
687:     print( y.intname <- as.character(channel.y))
688:     xyVivas <- exprs(fcompVivas[[nm ]])
689:     panel.abline(v= median(xyVivas[,x.intname]),col="red", alpha=0.5)
690:     panel.abline(h= median(xyVivas[,y.intname]),col="green", alpha=0.5)
691:     panel.points(  xyVivas [,x.intname],  xyVivas [,y.intname] , col="blue"  ,
    alpha=0.3, pch=".", cex=1)
692:
693:          }))
694:
695:   x11(width=90, height=50)
696:
697:
698:   print( xyplot(FL8 ~ FL6, fcompTodas,smooth=TRUE, pch = 21, cex = 0.3,
699:     main=paste( "Seleccion- morphGateScale:", morphGateScale) ,
700:     layout=my.layout,
701:
702:         panel = function(x, frames, channel.x, channel.y, ...) {
703:             nm <- as.character(x)
704:     print(nm)
705:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
706:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
707:     my.panel(x, y, frame = frames[[nm]], ...)
708:     print( x.intname <- as.character(channel.x))
709:     print( y.intname <- as.character(channel.y))
710:     xyVivas <- exprs(fcompVivas[[nm ]])
711:     panel.abline(v= median(xyVivas[,x.intname]),col="red", alpha=0.5)
712:     panel.abline(h= median(xyVivas[,y.intname]),col="green", alpha=0.5)
713:     panel.points(  xyVivas [,x.intname],  xyVivas [,y.intname] , col="blue"  ,
    alpha=0.3, pch=".", cex=1)
714:          }))
715:
716: }
717:
718:
719:
720:   ######################################################################
721:   ### chunk : Funcion fnvisualizarScatervivas.FL                       #
722:   ######################################################################
723:
724:   fnvisualizarScatervivas.FL <- function(fcompTodas, fun= FL1 ~ FSC ,
725:                 fcompVivas, main="Visualizacion",
726:                     mult=1,my.layout=c(5,5)) {
727:
728:   trellis.par.set(theme = col.whitebg())
729:   lw <- list(ylab.axis.padding = list(x = 0.5), left.padding = list(x = 0.5,
730:     units = "inches"), right.padding = list(x = 0, units = "inches"),
731:     panel = list(x = 1*mult, units = "inches"))
732:   lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
733:     list(x = 0, units = "inches"), panel = list(x = 1*mult, units = "inches"))
734:
735:     lattice.options(layout.widths = lw, layout.heights = lh)
736:
737:
738:   x11(width=90, height=50)
```

```
739:
740:   print( xyplot( fun , fcompTodas,smooth=TRUE, pch = 21, cex = 0.3,
741:     main=main ,
742:     layout=my.layout,
743:          panel = function(x, frames, channel.x, channel.y, ...) {
744:             nm <- as.character(x)
745:             print(nm)
746:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
747:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
748:     my.panel(x, y, frame = frames[[nm]], ...)
749:     print( x.intname <- as.character(channel.x))
750:     print( y.intname <- as.character(channel.y))
751:     xyVivas <- exprs(fcompVivas[[nm ]])
752:     panel.abline(v= median(xyVivas[,x.intname]),col="red", alpha=0.5)
753:     panel.abline(h= median(xyVivas[,y.intname]),col="green", alpha=0.5)
754:     panel.points(xyVivas [,x.intname],  xyVivas [,y.intname] , col="blue",
755:          alpha=0.3, pch=".", cex=1)
756:     intMedianV <- median(xyVivas[,x.intname])
757:     maxMedianV <- max(x)
758:     intMedianH <- median(xyVivas[,y.intname])
759:     maxMedianH <- max(y)
760:     panel.text(intMedianV ,maxMedianH*1.4 ,pos=1 ,
761:         format(intMedianV ,digits=3),col="red", cex=0.6)
762:     panel.text(maxMedianV ,intMedianH*1.1 ,pos=1 ,
763:         format(intMedianH ,digits=3),col="green", cex=0.6)
764:
765:         }))
766:
767:   }
768:
769:
770:
771:
772:
773:
774:   ########################################################################
775:   ###  Funcion visualizarScater6
776:   ########################################################################
777:
778:   visualizarScater6 <-  function (fp, main="Visualizacion",mult=0.5,layout=c(6,1)){
779:
780:
781:   trellis.par.set(theme = col.whitebg())
782:   lw <- list(ylab.axis.padding = list(x = 0.5), left.padding = list(x = 0.5,
783:     units = "inches"), right.padding = list(x = 0, units = "inches"),
784:     panel = list(x = 1*mult, units = "inches"))
785:   lh <- list(bottom.padding = list(x = 0, units = "inches"), top.padding <-
786:     list(x = 0, units = "inches"), panel = list(x = 1*mult, units = "inches"))
787:
788:     lattice.options(layout.widths = lw, layout.heights = lh)
789:
790:     #fp<-flowData
791:     #x11(width=90, height=50)
792:       g1<- (xyplot(`SSC` ~ `FSC`, fp, main=main, pch = 21, cex = 0.3,
793:          scales=list(x=list(draw=FALSE, xlab.cex=0.5),y=list(draw=FALSE) ),
794:          layout=c(6,1),par.strip.text=list(cex=0.5),
795:
796:              panel = function(x, frames, channel.x, channel.y, ...) {
797:                  nm <- as.character(x)
798:                  x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
799:                  y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
800:              my.panel(x, y, frame = frames[[nm]], ...)
801:          panel.abline(v= median(x),col="red", alpha=0.5)
802:          panel.abline(h= median(y),col="green", alpha=0.5) }))
803:
804:     #x11(width=90, height=50)
805:       g2<- (xyplot(`FL3` ~ `FL1`, fp,  pch = 21, cex = 0.3,
806:          scales=list(x=list(draw=FALSE, xlab.cex=0.5),y=list(draw=FALSE) ),
```

```
807:          layout=c(6,1),par.strip.text=list(cex=0.5),
808:
809:              panel = function(x, frames, channel.x, channel.y, ...) {
810:                  nm <- as.character(x)
811:                  x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
812:                  y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
813:              my.panel(x, y, frame = frames[[nm]], ...)
814:          panel.abline(v= median(x),col="red", alpha=0.5)
815:          panel.abline(h= median(y),col="green", alpha=0.5)}))
816:
817:
818:      #x11(width=90, height=50)
819:        g3 <-(xyplot(`FL6` ~ `FL1`, fp,  pch = 21, cex = 0.3,
820:          scales=list(x=list(draw=FALSE, xlab.cex=0.5),y=list(draw=FALSE) ),
821:          layout=c(6,1),par.strip.text=list(cex=0.5),
822:
823:              panel = function(x, frames, channel.x, channel.y, ...) {
824:                  nm <- as.character(x)
825:                  x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
826:                  y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
827:              my.panel(x, y, frame = frames[[nm]], ...)
828:          panel.abline(v= median(x),col="red", alpha=0.5)
829:          panel.abline(h= median(y),col="green", alpha=0.5)}))
830:
831:
832:      #x11(width=90, height=50)
833:        g4<- (xyplot(`FL8` ~ `FL1`, fp, pch = 21, cex = 0.3,
834:          scales=list(x=list(draw=FALSE, xlab.cex=0.5),y=list(draw=FALSE) ),
835:          layout=c(6,1),par.strip.text=list(cex=0.5),
836:
837:              panel = function(x, frames, channel.x, channel.y, ...) {
838:                  nm <- as.character(x)
839:                  x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
840:                  y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
841:              my.panel(x, y, frame = frames[[nm]], ...)
842:          panel.abline(v= median(x),col="red", alpha=0.5)
843:          panel.abline(h= median(y),col="green", alpha=0.5)}))
844:
845:
846:      #x11(width=90, height=50)
847:        g5<- (xyplot(`FL6`~ `FL3`, fp, pch = 21, cex = 0.3,
848:          scales=list(x=list(draw=FALSE, xlab.cex=0.5),y=list(draw=FALSE) ),
849:          layout=c(6,1),par.strip.text=list(cex=0.5),
850:
851:              panel = function(x, frames, channel.x, channel.y, ...) {
852:                  nm <- as.character(x)
853:                  x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
854:                  y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
855:              my.panel(x, y, frame = frames[[nm]], ...)
856:          panel.abline(v= median(x),col="red", alpha=0.5)
857:          panel.abline(h= median(y),col="green", alpha=0.5)}))
858:
859:
860:      #x11(width=90, height=50)
861:        g6<- (xyplot(`FL8`~ `FL3`, fp,  pch = 21, cex = 0.3,
862:          scales=list(x=list(draw=FALSE, xlab.cex=0.5),y=list(draw=FALSE) ),
863:          layout=c(6,1),par.strip.text=list(cex=0.5),
864:
865:              panel = function(x, frames, channel.x, channel.y, ...) {
866:                  nm <- as.character(x)
867:                  x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
868:                  y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
869:              my.panel(x, y, frame = frames[[nm]], ...)
870:          panel.abline(v= median(x),col="red", alpha=0.5)
871:          panel.abline(h= median(y),col="green", alpha=0.5)}))
872:
873:
874:      #x11(width=190, height=100)
```

```
875:       g7<- ( xyplot(`FL8` ~ `FL6`, fp, pch = 21,
876:          scales=list(x=list(draw=FALSE),y=list(draw=FALSE)),
877:          layout=c(6,1),par.strip.text=list(cex=0.5),
878:              panel = function(x, frames, channel.x, channel.y, ...) {
879:                  nm <- as.character(x)
880:                  x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
881:                  y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
882:              my.panel(x, y, frame = frames[[nm]], ...)
883:          panel.abline(v= median(x),col="red", alpha=0.5)
884:          panel.abline(h= median(y),col="green", alpha=0.5)}))
885:     x11(width=120, height=130)
886:        print(plot(g1, position = c(0,0,1,1.8), more = TRUE))
887:        print(plot(g2, position = c(0,0,1,1.55), more = TRUE))
888:        print(plot(g3, position = c(0,0,1,1.3), more = TRUE))
889:        print(plot(g4, position = c(0,0,1,1.05), more = TRUE))
890:        print(plot(g5, position = c(0,0,1,0.8), more = TRUE))
891:        print(plot(g6, position = c(0,0,1,0.55), more = TRUE))
892:        print(plot(g7, position = c(0,0,1,0.3), more = FALSE))
893:
894:   }
895:
896:
897:
898:   fnvisualizarScatervivasScreen <- function(fcompTodas, fun= FL1 ~ FSC ,fcompVivas,
     main="", mult=0.7,
899:                  my.layout=c(5,5), paneldesc=NULL ,
900:                  filename="Rplot%d.png",
901:                  imageW=21, imageH=29.7,
902:                  my.scales= list(x = list(axs = "i", cex=0.5), y = list(axs = "i",
     cex=0.5)),
903:                  my.colorsSup = colorRampPalette(rainbow(10)),
904:                  my.colorsInf = colorRampPalette(YlOrBr , space = "Lab")
905:                  ) {
906:
907:     ### Para imprimir las etiquetas con los nombres de los fluorocromos
908:     variables <- all.vars(fun, functions = FALSE, unique = TRUE)
909:     if (is.null(paneldesc)) {
910:         my.ylab=variables [[1]]
911:         my.xlab=variables [[2]]
912:
913:       } else {
914:         ### Si enviamos la matriz con las descripciones .....
915:         my.ylab=paneldesc [variables [[1]]]
916:         my.xlab=paneldesc [variables [[2]]]
917:
918:       }
919:     ###
920:
921:
922:     YlOrBr <- c("#FFFFD4", "#FED98E", "#FE9929", "#D95F0E", "#993404")
923:     colors <- colorRampPalette(c("white", brewer.pal(9, "Blues")))
924:     colors1 <- colorRampPalette(YlOrBr , space = "Lab")
925:     colors_sup <- colors
926:     ##my.colorsSup
927:     #colors_inf <- my.colorsInf
928:     #colors2 <- rev(heat_hcl(33, c = c(80, 30), l = c(30, 90), power = c(1/5,
     1.3)))
929:
930:   my.panel_inf <- function(x, y, ..., frame)
931:   {
932:     ###densCols produces a vector containing colors which encode the
933:     ####local densities at each point in a scatterplot.
934:     cols <- densCols(x, y, nbin=120,colramp = colorRampPalette(YlOrBr , space =
     "Lab"))
935:     print("debug.... my.panel")
936:         panel.xyplot(x, y,col=cols, ...)
937:         #glpolygon(filter)
938:   }
```

```
939:
940:   my.panel_sup <- function(x, y, ..., frame)
941:   {
942:         cols <- densCols(x, y, nbin=120,colramp = colorRampPalette(rainbow(10))  )
943:     print("debug.... my.panelsup")
944:         panel.xyplot(x, y,col=cols, ...)
945:         #glpolygon(filter)
946:   }
947:
948:
949:   lw <- list(ylab.axis.padding = list(x = 0.5), left.padding = list(x = 0.5,
950:     units = "inches"), right.padding = list(x = 0, units = "inches"),
951:     panel = list(x = 1*mult, units = "inches"))
952:   lh <- list(bottom.padding = list(x = 0.4, units = "inches"), top.padding <-
953:     list(x = 0, units = "inches"), panel = list(x = 1*mult, units = "inches"))
954:
955:     lattice.options(layout.widths = lw, layout.heights = lh)
956:
957:   my.plot<-( xyplot( fun , fcompTodas, pch = ".", cex = 0.6,
958:     main=main , xlab = my.xlab, ylab=my.ylab,
959:     layout=my.layout, scales=my.scales, par.strip.text = list(cex=0.6),
960:          panel = function(x, frames, channel.x, channel.y, ...) {
961:             nm <- as.character(x)
962:     print(nm)
963:             x <- flowViz:::evalInFlowFrame(channel.x, frames[[nm]])
964:             y <- flowViz:::evalInFlowFrame(channel.y, frames[[nm]])
965:     print("debug.... xyplot.")
966:     my.panel_inf(x, y, frame = frames[[nm]], ...)
967:     print( x.intname <- as.character(channel.x))
968:     print( y.intname <- as.character(channel.y))
969:     print("debug.... xyplot..")
970:     xyVivas <- exprs(fcompVivas[[nm ]])
971:     print("debug.... xyplot...")
972:     panel.abline(v= median(xyVivas[,x.intname]),col="red", alpha=0.5)
973:     panel.abline(h= median(xyVivas[,y.intname]),col="green", alpha=1, cex=1)
974:     print("debug.... xyplot....")
975:     #panel.points(  xyVivas [,x.intname],  xyVivas [,y.intname] , col="blue"  ,
     alpha=0.3, pch=".", cex=1)
976:         xsup <- flowViz:::evalInFlowFrame(channel.x, fcompVivas[[nm]])
977:             ysup <- flowViz:::evalInFlowFrame(channel.y, fcompVivas[[nm]])
978:     print("debug.... xyplot.....")
979:     my.panel_sup(xsup,ysup, frame=fcompVivas[[nm]],... )
980:     intMedianV <- median(xyVivas[,x.intname])
981:     maxMedianV <- max(xyVivas[,x.intname])
982:     intMedianH <- median(xyVivas[,y.intname])
983:     maxMedianH <- max(xyVivas[,y.intname])
984:     print("debug.... xyplot.....")
985:     nombres<-as.matrix(parameters(fcompVivas[[nm]])$name)
986:     print("debug.... xyplot......")
987:     idxXscale <- which(nombres==x.intname)
988:     maxXScaleVals<-as.matrix(parameters(fcompVivas[[nm]])$maxRange)[idxXscale]
989:     minXScaleVals<-as.matrix(parameters(fcompVivas[[nm]])$minRange)[idxXscale]
990:     print("debug.... xyplot.......")
991:     idxYscale <- which(nombres==y.intname)
992:     maxYScaleVals<-as.matrix(parameters(fcompVivas[[nm]])$maxRange)[idxYscale]
993:     minYScaleVals<-as.matrix(parameters(fcompVivas[[nm]])$minRange)[idxYscale]
994:     print("debug.... xyplot.......")
995:     print(maxYScaleVals);print(maxYScaleVals)
996:     print("********************")
997:     print(intMedianH)
998:     print("debug.... xyplot.......")
999:     panel.text(maxXScaleVals*0.85,maxYScaleVals*0.95 ,pos=1 ,format(intMedianV ,
     digits=4),col="red", cex=0.6)
1000:     #panel.text(maxMedianV ,intMedianH*0.93 ,pos=1 ,format(intMedianH ,digits=5),
     col="green", cex=0.6)
1001:     panel.text(maxXScaleVals*0.85,maxYScaleVals*0.80 ,pos=1 ,format(intMedianH ,
     digits=4),col="green", cex=0.6)
1002:     print("debug.... xyplot........")
```

```
1003:         }))
1004:   my.plot
1005:   #dev.off()
1006:   }
1007:
1008:
1009:
1010:   ######################################################################
1011:   ###  Funcion sort.data.frame                  #
1012:   ######################################################################
1013: sort.data.frame <- function(x, by){
1014:     # Author: Kevin Wright
1015:     # with some ideas from Andy Liaw
1016:     # http://tolstoy.newcastle.edu.au/R/help/04/07/1076.html
1017:
1018:     # x: A data.frame
1019:     # by: A one-sided formula using + for ascending and - for descending
1020:     #     Sorting is left to right in the formula
1021:
1022:     # Useage is:
1023:     # library(nlme);
1024:     # data(Oats)
1025:     # sort(Oats, by= ~nitro-Variety)
1026:
1027:     if(by[[1]] != "~")
1028:         stop("Argument 'by' must be a one-sided formula.")
1029:
1030:     # Make the formula into character and remove spaces
1031:     formc <- as.character(by[2])
1032:     formc <- gsub(" ", "", formc)
1033:     # If the first character is not + or -, add +
1034:     if(!is.element(substring(formc, 1, 1), c("+", "-")))
1035:         formc <- paste("+", formc, sep = "")
1036:
1037:     # Extract the variables from the formula
1038:     vars <- unlist(strsplit(formc, "[\\+\\-]"))
1039:     vars <- vars[vars != ""] # Remove any extra "" terms
1040:
1041:     # Build a list of arguments to pass to "order" function
1042:     calllist <- list()
1043:     pos <- 1 # Position of + or -
1044:     for(i in 1:length(vars)){
1045:         varsign <- substring(formc, pos, pos)
1046:         pos <- pos + 1 + nchar(vars[i])
1047:         if(is.factor(x[, vars[i]])){
1048:             if(varsign == "-") {
1049:                 calllist[[i]] <- -rank(x[, vars[i]])
1050:             } else {
1051:                 calllist[[i]] <- rank(x[, vars[i]])
1052:             }
1053:         } else {
1054:             if(varsign == "-") {
1055:                 calllist[[i]] <- -x[, vars[i]]
1056:             } else {
1057:                 calllist[[i]] <- x[,vars[i]]
1058:             }
1059:         }
1060:     }
1061:     return(x[do.call("order", calllist), ])
1062: }
1063:
1064:
```

```
 1:
 2:
 3:   ### VERSION: 5.11.07.2010
 4:   ### Chuunk: positive.selecction.R
 5:   ##############################################################################
 6:   ### Chunk: Divide una población enpositivos y negativos
 7:   ##############################################################################
 8:   # El calculo se raliza con las celulas seleccionadas en eslala lineal##
 9:   ## La funcion rangeGate crea un filtro que separa las poblaciones positivas y
    negativas
10:   x11(width = 4, height=4)
11:   rg1 <- rangeGate(Pos.flowData.orig.N[1:6],filterId="fltFL1","FL1",
12:                    plot=TRUE,absolute=FALSE,positive=TRUE,alpha=0.6)
13:   x11(width = 4, height=4)
14:   rg3 <- rangeGate(Pos.flowData.orig.N[1:6],filterId="fltFL3","FL3",
15:                    plot=TRUE,absolute=FALSE,positive=TRUE,alpha=0.9)
16:   x11(width = 4, height=4)
17:   rg6 <- rangeGate(Pos.flowData.orig.N[1:6],filterId="fltFL6","FL6",
18:                    plot=TRUE,absolute=FALSE,positive=TRUE,alpha=0.6)
19:   x11(width = 4, height=4)
20:   rg8 <- rangeGate(Pos.flowData.orig.N[1:6],filterId="fltFL8","FL8",
21:                    plot=TRUE,absolute=FALSE,positive=TRUE,alpha=0.6)
22:
23:   Pos.flowData.orig.N.rg1 <- filter(Pos.flowData.orig.N, rg1 )
24:   summFL1<-summary(Pos.flowData.orig.N.rg1)
25:   FL1.pos.orig<-split(Pos.flowData.orig.N, Pos.flowData.orig.N.rg1 )$'fltFL1+'
26:   a1.orig<-filterDetails(Pos.flowData.orig.N.rg1[[1]])$fltFL1$filter@min
27:   a1.lin <- 10^(a1.orig*4/4096)
28:   Medians.orig.pos.FL1 <-(as.matrix(fsApply(FL1.pos.orig, each_col, median)))[,"FL1"]
29:   rg1.lin <- rectangleGate(filterId="rg1.lin", "FL1"=c(a1.lin , Inf))
30:   Pos.flowData.lin.N.rg1 <- filter(Pos.flowData.lin.N, rg1.lin )
31:   FL1.pos.lin<-split(Pos.flowData.lin.N, Pos.flowData.lin.N.rg1 )$'rg1.lin+'
32:   Medians.lin.pos.FL1<-(as.matrix(fsApply(FL1.pos.lin, each_col, median)))[,"FL1"]
33:   FL1.neg.lin<-split(Pos.flowData.lin.N, Pos.flowData.lin.N.rg1 )$'rg1.lin-'
34:   Medians.lin.neg.FL1<-(as.matrix(fsApply(FL1.neg.lin, each_col, median)))[,"FL1"]
35:   if (panelFile==2){
36:
37:   rg3 <- rectangleGate(filterId="fltFL3", "FL3"=c(2000 , Inf))
38:   Pos.flowData.orig.N.rg3 <- filter(Pos.flowData.orig.N, rg3 )
39:   FL3.pos.orig<-split(Pos.flowData.orig.N, Pos.flowData.orig.N.rg3 )$'fltFL3+'
40:   a3.orig<-2000
41:   a3.lin <- 10^(a3.orig*4/4096)
42:   Medians.orig.pos.FL3 <-(as.matrix(fsApply(FL3.pos.orig, each_col, median)))[,"FL3"]
43:   rg3.lin <- rectangleGate(filterId="rg3.lin", "FL3"=c(a3.lin , Inf))
44:   Pos.flowData.lin.N.rg3 <- filter(Pos.flowData.lin.N, rg3.lin )
45:   FL3.pos.lin<-split(Pos.flowData.lin.N, Pos.flowData.lin.N.rg3 )$'rg3.lin+'
46:   Medians.lin.pos.FL3<-(as.matrix(fsApply(FL3.pos.lin, each_col, median)))[,"FL3"]
47:   FL3.neg.lin<-split(Pos.flowData.lin.N, Pos.flowData.lin.N.rg3 )$'rg3.lin-'
48:   Medians.lin.neg.FL3<-(as.matrix(fsApply(FL3.neg.lin, each_col, median)))[,"FL3"]
49:   visualizarDensity(FL3.pos.orig)
50:
51:
52:   } else {
53:
54:   Pos.flowData.orig.N.rg3 <- filter(Pos.flowData.orig.N, rg3 )
55:   FL3.pos.orig<-split(Pos.flowData.orig.N, Pos.flowData.orig.N.rg3 )$'fltFL3+'
56:   a3.orig<-filterDetails(Pos.flowData.orig.N.rg3[[1]])$fltFL3$filter@min
57:   a3.lin <- 10^(a3.orig*4/4096)
58:   Medians.orig.pos.FL3 <-(as.matrix(fsApply(FL3.pos.orig, each_col, median)))[,"FL3"]
59:   rg3.lin <- rectangleGate(filterId="rg3.lin", "FL3"=c(a3.lin , Inf))
60:   Pos.flowData.lin.N.rg3 <- filter(Pos.flowData.lin.N, rg3.lin )
61:   FL3.pos.lin<-split(Pos.flowData.lin.N, Pos.flowData.lin.N.rg3 )$'rg3.lin+'
62:   Medians.lin.pos.FL3<-(as.matrix(fsApply(FL3.pos.lin, each_col, median)))[,"FL3"]
63:   FL3.neg.lin<-split(Pos.flowData.lin.N, Pos.flowData.lin.N.rg3 )$'rg3.lin-'
64:   Medians.lin.neg.FL3<-(as.matrix(fsApply(FL3.neg.lin, each_col, median)))[,"FL3"]
65:   visualizarDensity(FL3.pos.orig)
66:
67:
```

```
68:   }
69:
70:
71:   Pos.flowData.orig.N.rg6 <- filter(Pos.flowData.orig.N, rg6 )
72:   FL6.pos.orig<-split(Pos.flowData.orig.N, Pos.flowData.orig.N.rg6 )$'fltFL6+'
73:   a6.orig<-filterDetails(Pos.flowData.orig.N.rg6[[1]])$fltFL6$filter@min
74:   a6.lin <- 10^(a6.orig*4/4096)
75:   Medians.orig.pos.FL6 <-(as.matrix(fsApply(FL6.pos.orig, each_col, median)))[,"FL6"]
76:   rg6.lin <- rectangleGate(filterId="rg6.lin", "FL6"=c(a6.lin , Inf))
77:   Pos.flowData.lin.N.rg6 <- filter(Pos.flowData.lin.N, rg6.lin )
78:   FL6.pos.lin<-split(Pos.flowData.lin.N, Pos.flowData.lin.N.rg6 )$'rg6.lin+'
79:   Medians.lin.pos.FL6<-(as.matrix(fsApply(FL6.pos.lin, each_col, median)))[,"FL6"]
80:   FL6.neg.lin<-split(Pos.flowData.lin.N, Pos.flowData.lin.N.rg6 )$'rg6.lin-'
81:   Medians.lin.neg.FL6<-(as.matrix(fsApply(FL6.neg.lin, each_col, median)))[,"FL6"]
82:   visualizarDensity(FL6.pos.orig)
83:
84:
85:   Pos.flowData.orig.N.rg8 <- filter(Pos.flowData.orig.N, rg8 )
86:   FL8.pos.orig<-split(Pos.flowData.orig.N, Pos.flowData.orig.N.rg8 )$'fltFL8+'
87:   a8.orig<-filterDetails(Pos.flowData.orig.N.rg8[[1]])$fltFL8$filter@min
88:   a8.lin <- 10^(a8.orig*4/4096)
89:   Medians.orig.pos.FL8 <-(as.matrix(fsApply(FL8.pos.orig, each_col, median)))[,"FL8"]
90:   rg8.lin <- rectangleGate(filterId="rg8.lin", "FL6"=c(a8.lin , Inf))
91:   Pos.flowData.lin.N.rg8 <- filter(Pos.flowData.lin.N, rg8.lin )
92:   FL8.pos.lin<-split(Pos.flowData.lin.N, Pos.flowData.lin.N.rg8 )$'rg8.lin+'
93:   Medians.lin.pos.FL8<-(as.matrix(fsApply(FL8.pos.lin, each_col, median)))[,"FL8"]
94:   FL8.neg.lin<-split(Pos.flowData.lin.N, Pos.flowData.lin.N.rg8 )$'rg8.lin-'
95:   Medians.lin.neg.FL8<-(as.matrix(fsApply(FL8.neg.lin, each_col, median)))[,"FL8"]
96:   visualizarDensity(FL8.neg.lin)
97:
98:
```