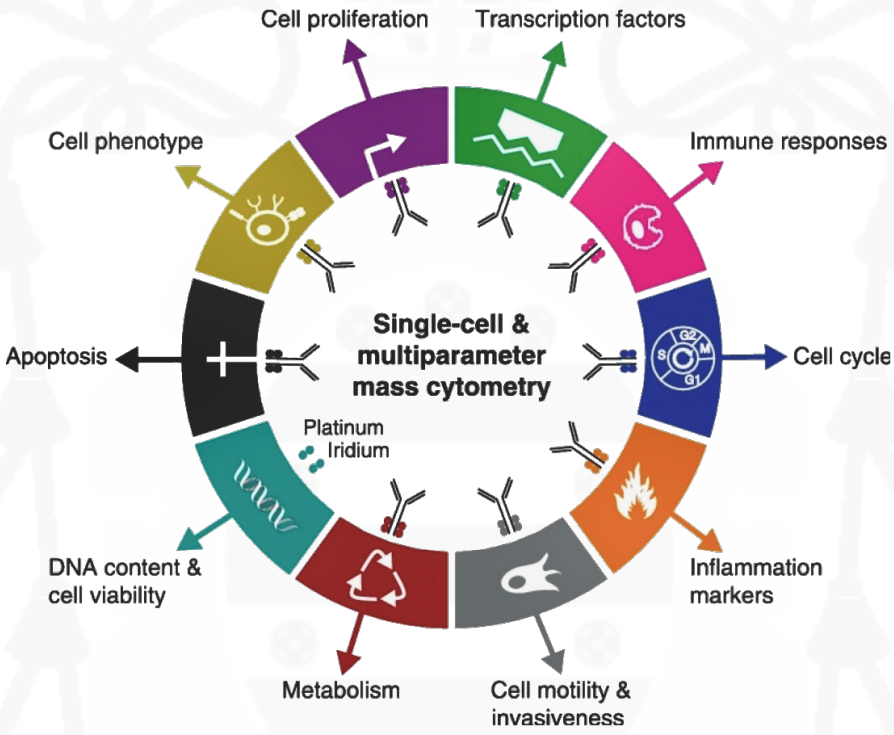
TRABAJO FIN DE GRADO

BIOTECNOLOGÍA

DESARROLLO DE UN MÉTODO BIOINFORMÁTICO DE TRATAMIENTO DE DATOS DE SEÑALES TRANSITORIAS MULTIELEMENTALES PARA ESPECTROMETRÍA DE MASAS



Darío Bagüés Castro

Departmento de Química Física y Analítica

Julio 2024

UNIVERSIDAD DE OVIEDO

FACULTAD DE BIOLOGÍA

Resumen

El análisis de células individuales es un tema de investigación en auge, ya que detectar diferencias a nivel de las células individuales permiten diferenciar pequeñas poblaciones celulares que se comportan de forma diferente a la población global. Este hecho puede tener importantes implicaciones en estudios bioquímicos, biotecnológicos, clínicos, farmacológicos, etc. Aunque la técnica más común de análisis de células individuales es la citometría de flujo, en los últimos años han aparecido nuevas técnicas analíticas con capacidad para llevar a cabo esta tarea y, en concreto, la espectrometría de masas con fuente de ionización de plasma por acoplamiento inductivo (ICP-MS) ha surgido como una alternativa adecuada para el análisis tanto del contenido elemental como de la presencia de ciertas biomoléculas a través del etiquetado con anticuerpos marcados con elementos lantánidos en células individuales en la forma de *single-cell* ICP-MS (SC-ICP-MS).

La novedosa técnica de citometría de masas se basa en los mismos fundamentos que el SC-ICP-MS con el uso de un analizador de masas que permite la medida casi simultánea de varios isótopos en una misma célula lo que posibilita la detección simultánea de decenas de marcadores en cada célula individual. Este tipo de técnicas generan una gran cantidad de datos complejos, que hacen necesario el desarrollo de nuevas herramientas bioinformáticas que permitan su procesamiento y visualización para extraer las conclusiones necesarias a partir de ellos.

Los objetivos de este TFG serán, en primer lugar, familiarizar al estudiante con las técnicas de análisis elemental de células individuales para que, desde el conocimiento de los fundamentos de la técnica, pueda desarrollar, a continuación, una herramienta bioinformática que facilite el procesamiento de los datos obtenidos mediante la técnica de *single cell*-ICP-MS y citometría de masas y su representación gráfica siguiendo los algoritmos y estándares más utilizados por la comunidad científica.

Abstract

The analysis of individual cells is a growing research topic, as detecting differences at the individual cell level allows distinguishing small cellular populations that behave differently from the overall population. This fact can have significant implications in biochemical, biotechnological, clinical, pharmacological, and other studies. Although the most common technique for single-cell analysis is flow cytometry, in recent years, new analytical techniques with the capability to perform this task have emerged. Specifically, inductively coupled plasma mass spectrometry (ICP-MS) has arisen as a suitable alternative for the analysis of both elemental content and the presence of certain biomolecules through lanthanide-labeled antibody tagging in individual cells, in the form of single-cell ICP-MS (SC-ICP-MS).

The novel mass cytometry technique is based on the same principles as SC-ICP-MS, using a mass analyzer that allows for the nearly simultaneous measurement of multiple isotopes in a single cell. This enables the simultaneous detection of dozens of markers in each individual cell. These techniques generate a large amount of complex data, making it necessary to develop new bioinformatics tools that facilitate the processing and visualization of this data to draw the necessary conclusions from it.

The objectives of this bachelor's thesis will, first of all, familiarize the student with the techniques of elemental analysis of individual cells so that, with a knowledge of the technique's fundamentals, they can subsequently develop a bioinformatics tool that facilitates the processing of data obtained through the single-cell ICP-MS and mass cytometry techniques and their graphical representation following the algorithms and standards most commonly used by the scientific community.

Declaración de originalidad

Darío Bagüés Castro, estudiante del Grado en Biotecnología en la Facultad de Biología de la Universidad de Oviedo, en relación con el Trabajo Fin de Grado *Desarrollo de un método bioinformático de tratamiento de datos de señales transitorias multielementales para espectrometría de masas* presentado para su defensa y evaluación en el curso 2023/2024, declara que asume la originalidad de dicho trabajo, entendida en el sentido de que no ha utilizado fuentes sin citarlas debidamente.

Índice

1. Introducción

El análisis de células individuales ha emergido como una herramienta fundamental en numerosos campos científicos y biomédicos debido a su capacidad para revelar información detallada sobre la heterogeneidad celular y las complejas interacciones dentro de poblaciones celulares. Esta capacidad de análisis a nivel individual es esencial para comprender fenómenos biológicos, identificar biomarcadores relevantes y avanzar en el desarrollo de terapias personalizadas.

Un enfoque comúnmente utilizado en este ámbito es la citometría de flujo, que permite un análisis rápido y preciso de múltiples parámetros celulares en células individuales mediante la detección y cuantificación de la fluorescencia emitida por sondas específicas con las que se han marcado previamente las células. Recientemente, ha surgido la posibilidad de utilizar la espectrometría de masas elemental, en particular la técnica de espectrometría de masas de plasma acoplado inductivamente (ICP-MS, por sus siglas en inglés), como una alternativa que ofrece ventajas sobre la citometría convencional al proporcionar información detallada sobre la composición elemental de las células y permitir la detección simultánea de múltiples elementos en una única célula.

La citometría de flujo es una técnica creada por Wallace H. Coulter en 1953 utilizando el principio de Coulter basado en la impedancia. En su origen, la técnica consistía en pasar células a través de una pequeña abertura donde los cambios en la impedancia eléctrica son proporcionales al volumen celular, lo que permite contar y dimensionar las células (Shapiro, 2004). Mack Fulwyler integró la fluorescencia a la citometría de flujo en la década de 1960. Este avance permitió a los investigadores etiquetar células con marcadores fluorescentes, lo que hizo posible estudiar componentes celulares y funciones específicas (Fulwyler, 1965). En un citómetro de flujo moderno, las células son propulsadas a través de un haz de láser enfocado. A medida que las células interactúan con el láser, absorben y dispersan la luz y emiten fluorescencia a determinadas longitudes de onda en función de las moléculas empleadas como marcas fluorescentes. Los detectores capturan estas señales, proporcionando información sobre el tamaño celular, la granularidad y la presencia de biomarcadores (McKinnon, 2018), véase la figura 1.A

Una parte esencial del proceso es el uso de anticuerpos conjugados con fluorocromos para determinar la presencia de marcadores específicos en las células. Estos anticuerpos fluorescentes se unen específicamente a sus antígenos, que pueden encontrarse tanto en la superficie como dentro de las células. Esto permite la identificación y cuantificación de diversas subpoblaciones celulares mediante la detección de la fluorescencia emitida por los fluorocromos. Además, la

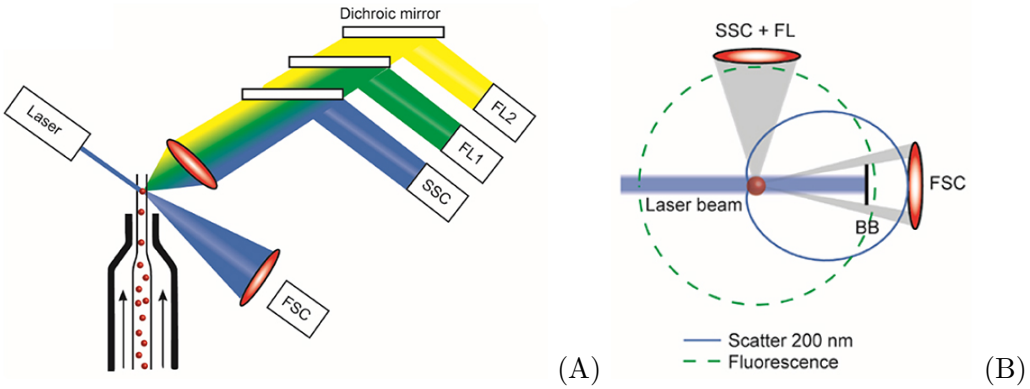


Figura 1: Análisis por citometría de flujo de muestras de vesículas extracelulares. (A) Durante la citometría de flujo, las partículas en suspensión son enfocadas hidrodinámicamente por un fluido de vaina (indicado por flechas) para intersectar con un láser. La luz dispersada por cada partícula es detectada por detectores de dispersión frontal (FSC) y lateral (SSC), junto con múltiples detectores de fluorescencia (FL1, FL2, etc.). (B) La fluorescencia (línea discontinua verde) es isotrópica y permite determinar la expresión de antígenos y el origen celular. La dispersión (línea continua azul) tiene una distribución angular influenciada por el tamaño y el índice de refracción de la partícula. Adaptado de Rikkert et al. (2020), bajo Creative Commons Attribution License (CC BY).

luz dispersada proporciona información adicional: la dispersión frontal (FSC, por sus siglas en inglés) está correlacionada con el tamaño celular, mientras que la dispersión lateral (SSC, por sus siglas en inglés) está relacionada con la granularidad o complejidad interna de la célula (Maecker et al., 2012), véase la figura 1.B.

La técnica ha avanzado significativamente, especialmente en su capacidad para realizar análisis multiparamétrico. Permite la medición simultánea de múltiples marcadores en células individuales, permitiendo un perfil exhaustivo de las características celulares. Los citómetros de flujo de alta gama ahora son capaces de analizar entre 30 y 50 parámetros diferentes por célula, lo que incluye una combinación de proteínas de superficie e intracelulares, así como diversas otras características celulares (Liu et al., 2022). Esta capacidad multiparamétrica es crucial para identificar y caracterizar poblaciones celulares complejas, como diferentes subconjuntos de células inmunitarias, y para comprender los estados funcionales de las células en salud y enfermedad (Schmit et al., 2021).

1.1. Espetrometría de masas elemental

La espectrometría de masas comienza con la ionización de la muestra para generar especies cargadas eléctricamente que, a continuación, son separadas en función de su relación masa/carga (m/z) en un analizador de masas mediante la aplicación de campos eléctricos y/o magnéticos.

El plasma acoplado inductivamente (ICP) es una fuente de ionización dura que utiliza un plasma generado por inducción electromagnética para ionizar los átomos del analito. Este método es particularmente efectivo para el análisis de elementos en muestras debido a su alta eficiencia de ionización, alta tolerabilidad a la introducción de muestras líquidas e incluso sólidas y su capacidad para manejar una amplia gama de matrices (Hou et al., 2021).

El plasma en un sistema ICP se genera al aplicar una corriente alterna de alta frecuencia a una bobina de inducción, creando un campo electromagnético que ioniza el gas (generalmente argón) presente en la cámara de plasma. Este plasma alcanza temperaturas extremadamente altas, típicamente entre 6000 y 10000 K, lo que permite la desolvatación, vaporización, atomización e ionización del analito, generando principalmente iones elementales con una carga positiva. Tiene una alta sensibilidad y capacidad para manejar matrices complejas con la posibilidad de realizar análisis multielementales de manera simultánea (Douvris et al., 2023).

Existen varios tipos de analizadores de masas utilizados en los espectrómetros de masas comerciales. El más convencional es el cuadrupolo, que utiliza campos eléctricos y magnéticos para filtrar y seleccionar iones basándose en su relación masa/carga. Funciona aplicando campos eléctricos oscilantes a cuatro barras metálicas paralelas. Estos campos hacen que los iones se desplacen a través del cuadrupolo, y solo los iones con una relación m/z específica pueden atravesar el dispositivo y llegar al detector de manera secuencial.

Sin embargo, existen otros analizadores de masas considearados casi simultáneos, es decir que pueden analizar varias m/z casi de manera simultánea. El analizador de masas de tiempo de vuelo (TOF) se basa en los diferentes tiempos que iones de distinta masa acelerados con la misma energía cinética tardan en recorrer una determinada distancia. Como todos los iones son acelerados con la misma energía cinética, su velocidad solo depende de su relación m/z. Por tanto, el tiempo recorrido en recorrer una distancia constante dependerá de la relación m/z de cada ión con diferencias muy pequeñas entre ellos, de tal manera que la medida de todos los iones puede considerarse casi simultánea.

Aunque los analizadores de TOF pueden proporcionar una alta resolución en el tiempo de vuelo, y, por lo tanto, en la determinación de la m/z, a menudo tienen una menor sensibilidad que los analizadores de cuadrupolo. Esto puede resultar en una capacidad de identificación y cuantificación ligeramente inferior para compuestos muy similares. A cambio, los analizadores de TOF permiten tiempos de integración muy cortos, obteniendo espectros completos de manera simultánea cada pocos microsegundos.

La elevada sensibilidad que proporciona el ICP-MS ha permitido, en los últimos años la

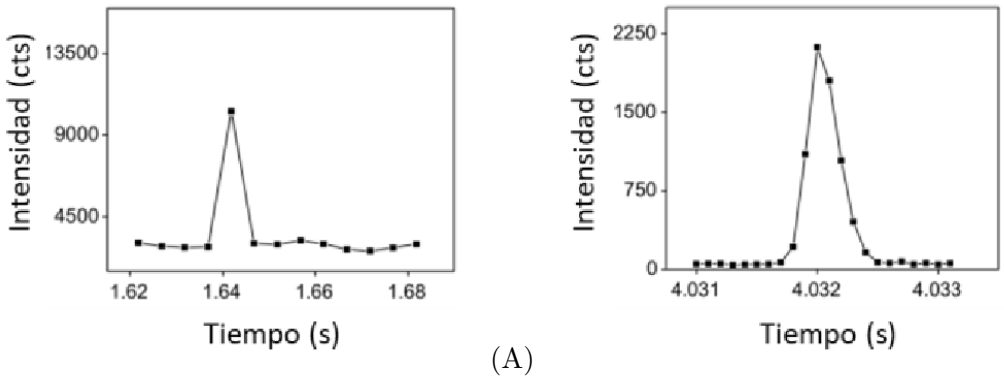


Figura 2: Influencia del tiempo de integración en la relación señal/ruido. (A) Gráfica de intensidad frente a tiempo de una medición con alto tiempo de integración, con máximo del pico y señal de fondo más elevados. (B) Gráfica de intensidad frente a tiempo de una medición con bajo tiempo de integración, con un pico más bajo, definido por más puntos y con menor fondo. Adaptado de Álvarez-Fernández García (2020)

detección de células individuales mediante esta técnica en lo que se conoce como "*single-cell* ICP-MS"(SC-ICP-MS). De esta forma, cuando se introduce una suspensión de células suficientemente diluida, cada vez que una célula individual llega al plasma es ionizada, generando una nube de iones que llega al detector en un periodo de tiempo muy corto, del orden de 500 µs. La llegada de cada célula al detector genera una señal transitoria de muy corta duración que se denomina evento o *spike*. Siempre que la suspensión celular esté suficientemente diluida para que la probabilidad de detectar dos células a la vez sea muy baja, la intensidad del evento será proporcional a la masa del elemento en la célula. Además, la frecuencia de los eventos será proporcional a la concentración celular en la muestra. Por otro lado, el fondo continuo viene dado por la introducción de manera constante con el tiempo de analito en forma disuelta.

Dada la corta duración de los eventos celulares en SC-ICP-MS, el tiempo de integración es un parámetro importante a tener en cuenta para la medida. El tiempo de integración consiste en el intervalo de tiempo durante el que el detector acumula los iones que le llegan. Mientras que el tiempo de integración típicamente utilizado para medidas normales en ICP-MS es de 100 ms, este debe ser mucho más corto en estudios de *single cell*, puesto que como se dijo anteriormente los eventos celulares son de unos 500 µs, de manera que debe reducirse por debajo de los 5 ms. En el caso de detectores que permitan tiempos de integración por debajo de los 500 µs, además, se podrán diferenciar los perfiles de cada evento generado por una célula, ya que con tiempos de integración suficientemente cortos estarán definidos por varios puntos de medida.

1.2. Cuantificación en ICP-MS

La cuantificación en ICP-MS implica la determinación precisa de la concentración de elementos presentes en una muestra. Dos estrategias utilizadas para este propósito son el calibrado externo y la dilución isotópica.

El calibrado externo es un método que implica la preparación de una serie de estándares con concentraciones conocidas del analito de interés. Estos estándares se utilizan para crear una curva de calibración, la cual es una representación gráfica de la respuesta del detector frente a la concentración del analito. La concentración del analito en la muestra desconocida se determina comparando su respuesta con la curva de calibración obtenida.

Esta técnica es ampliamente utilizada debido a su simplicidad y eficacia en condiciones controladas. Sin embargo, es susceptible a interferencias de matriz, las cuales pueden alterar la respuesta del detector y, por ende, la precisión del análisis (D. C. Harris, 2010). De haberlas, estas son corregibles utilizando calibrados por adiciones estándar o con patrón interno. Sin embargo, el ICP-MS es conocido por su robustez frente a distintos tipos de matrices, por lo que los efectos de matriz son escasos.

Para el análisis cuantitativo por SC-ICP-MS, las medidas se realizan por introducción de los estándares líquidos mediante una bomba de jeringa a bajo flujo y utilizando tiempos de integración cortos (menores a 5 ms). Para posteriormente determinar la cantidad de metal, se realiza una calibración externa con estándares elementales del metal deseado. Para ello, se introducen patrones iónicos de concentraciones conocidas, generando una señal continua solamente afectada por el ruido instrumental. Una vez calculada la intensidad media obtenida para cada concentración, se consigue una regresión que permite interpolar la intensidad de cada punto para hallar la concentración del metal en la célula individual.

A pesar de utilizar sistemas de alta eficiencia de transporte, la mayoría de los sistemas de introducción de muestras en ICP-MS se caracterizan por ser capaces de hacer llegar al plasma solo una pequeña fracción de la muestra. Mientras que los sistemas convencionales tienen eficiencias de transporte por debajo del 5 %, los de alta eficiencia pueden superar el 80 %. Esta eficiencia de transporte afecta a la masa de analito que llega al detector, ya que no toda la disolución de este se transporta hasta el plasma (Pereira et al., 2023). Sin embargo, cuando una célula alcanza el plasma, sus elementos se ionizan completamente. Por lo tanto, la eficiencia de transporte debe ser corregida, bien en los patrones de calibrado, o bien durante la interpolación de los eventos en la recta de calibrado. Así, si la eficiencia de transporte se corrige durante el calibrado, la masa de analito que llega al plasma durante la duración de un tiempo de integración

será:

$$m = [C] \cdot \nu \cdot dt \cdot TE \tag{1}$$

donde m será la masa de metal en la célula, $[C]$ la concentración detectada, ν el flujo, dt el tiempo de integración y TE la eficiencia de transporte.

1.3. Análisis multiparamétrico de células mediante ICP-MS: citometría de masas

La citometría de flujo es una herramienta valiosa para analizar células individuales, pero tiene desventajas notables, principalmente debido a limitaciones ópticas. Estas incluyen la amortiguación de fluorescencia y la autofluorescencia, así como problemas con los amplios espectros de emisión y absorción de las etiquetas fluorescentes, lo que lleva a solapamientos espectrales y limita el número de proteínas que se pueden analizar simultáneamente.

Para abordar estas limitaciones, Tanner et al. (2008) desarrollaron la citometría de masas, una técnica para el análisis multiparamétrico de células individuales basada en la detección por ICP-MS. Este método utiliza etiquetas de metal (lantánidos) para modificar los anticuerpos en lugar de las etiquetas fluorescentes tradicionales. La versión comercial, CyTOF™, fue introducida en 2009 por DVS Sciences Inc. (ahora Fluidigm) y es una adaptación del ICP-MS con un analizador de masas de tiempo de vuelo (TOF) (Bandura et al., 2009).

En CyTOF, el sistema de introducción de muestras incluye un nebulizador neumático microcéntrico conectado a una cámara de nebulización con un flujo de gas argón y un sistema de calentamiento. Este sistema seca las gotas que contienen las células y las transporta al plasma, donde se atomizan e ionizan. La nube de iones resultante de cada célula viaja a través de la óptica de iones y los filtros, hacia el analizador de masas TOF, donde los datos son recolectados y registrados por el detector. Véase en la figura 3 un esquema general de todo el proceso de la citometría de masas.

Cada célula en la citometría de masas produce una nube de iones que genera una señal transitoria al llegar al detector, durando hasta 500 µs y proporcionando espectros con información sobre todas las etiquetas metálicas asociadas a las células. Los datos de la citometría de masas pueden visualizarse de manera similar a la citometría de flujo tradicional, pero debido al alto número de parámetros, interpretar estos datos en gráficos bidimensionales es un desafío. Por lo tanto, a menudo se requieren métodos sofisticados de posprocesamiento de datos y métodos estadísticos (Kimball et al., 2018).

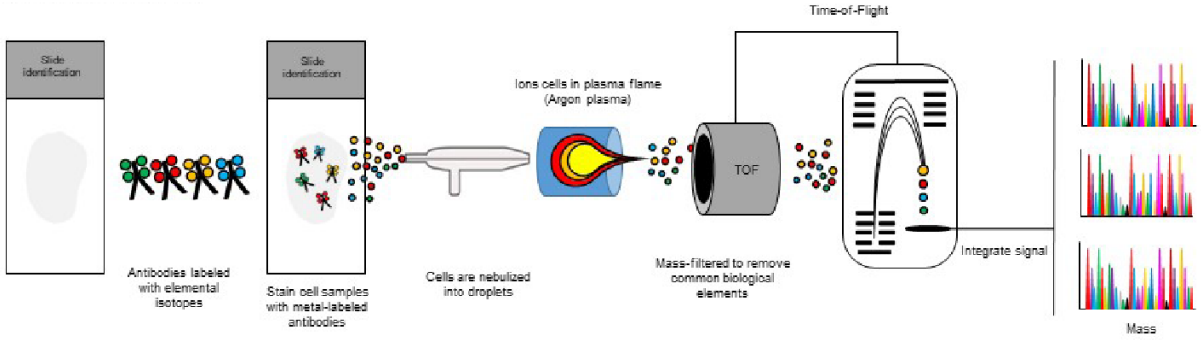


Figura 3: Flujo de trabajo esquemático de citometría de masas. Las muestras se etiquetan con combinaciones únicas de metales pesados y se agrupan para reducir la variabilidad. Luego, se incuban con anticuerpos dirigidos a proteínas específicas. Las células se nebulizan e introducen en un citómetro de masas. En un plasma de argón acoplado inductivamente, se rompen los enlaces covalentes y se liberan iones. La nube de iones se filtra para enriquecer los iones de metales pesados, que son cuantificados mediante espectrometría de masas de tiempo de vuelo, produciendo mediciones de células individuales. Adaptado de Parra (2018), bajo Creative Commons Attribution License (CC BY)

Sin embargo, la citometría de masas está limitada por su tasa de introducción de células, que es de alrededor de 1000 células por segundo, significativamente menor que las aproximadamente 10,000 células por segundo alcanzables con la citometría de flujo fluorescente. A pesar de esto, CyTOF puede analizar hasta 100 biomarcadores simultáneamente en una sola célula, superando con creces las hasta 15 etiquetas diferentes que los instrumentos de citometría de flujo más avanzados pueden lograr después de extensas optimizaciones (Theiner et al., 2022).

1.4. Manejo de datos en citometría de masas

El *Flow Cytometry Standard* (FCS) es un estándar de archivos de datos generados en experimentos de citometría de flujo. Anteriormente, el FCS solía ser el único formato de archivo ampliamente adoptado en la citometría de flujo, aunque en la actualidad hay otros también aceptados. La especificación del FCS (Spidlen et al., 2010) ha sido tradicionalmente desarrollada y mantenida por la Sociedad Internacional para el Avance de la Citometría (ISAC).

Este formato almacena la intensidad integrada de los eventos celulares, siendo estos en la citometría de flujo tradicional una célula biológica real, partículas empleadas como calibrantes u otra masa lo suficientemente grande como para activar el dispositivo de adquisición de datos del instrumento de citometría de flujo. Sin embargo, en SC-ICP-MS se obtiene una señal continua con el tiempo de la instensidad de cada ión de m/z correspondiente y a continuación se lleva a cabo filtrado posterior para determinar qué es un evento, por lo que los archivos `.fcs` generados

por la citometría de masas, pese a representar una rica fuente de datos para la investigación en biología celular, sufren una pérdida de información sustancial en comparación con los datos crudos adquiridos durante la medición. Por ejemplo, la información sobre el fondo, la duración y la forma de los eventos, así como su distribución durante la medida, se ve comprometida en este formato. Esta pérdida de información representa un desafío significativo para los investigadores, ya que limita la capacidad de realizar análisis detallados y precisos de los datos.

Con los bajísimos tiempos de integración de CyTOF, que obtiene una medida cada 13 µs de todo el espectro de masas entre m/z 75 y 210, unos minutos de medición implican millones de lecturas, cada una pudiendo aportar más de un centenar de valores para distintas intensidades de señal y la medida de señales de m/z donde no se espera ningún evento (canales cuya información es innecesaria). Por ello, el manejo de los datos crudos presenta sus propios obstáculos, como el tamaño y la complejidad de los mismos, así como la necesidad de herramientas especializadas para su procesamiento y análisis (Newell & Cheng, 2016). En este sentido, surge la urgencia de disponer de sistemas que permitan el tratamiento independiente de los datos de citometría de masas, especialmente para aplicaciones cuantitativas que requieren una precisión y reproducibilidad rigurosas.

1.5. Uso de *software* libre en la ciencia

El *software* no libre, a menudo referido como *software* privativo, impone restricciones a los usuarios que socavan las libertades fundamentales (Stallman, 2021). Estas limitaciones son inherentemente injustas y particularmente perjudiciales para el avance científico al constreñir a los usuarios de varias maneras: no pueden acceder al código fuente, lo que les impide entender, modificar o mejorar el programa; están coercionados en su capacidad para redistribuirlo, lo que imposibilita los esfuerzos colaborativos y la difusión del conocimiento, y dependen del proveedor para actualizaciones, correcciones de seguridad y continuidad del servicio, lo cual puede ser problemático si el proveedor deja de ofrecer soporte o quiebra. Estas restricciones violan los principios de libertad y autonomía, haciendo que el *software* no libre sea injusto por naturaleza (Stallman, 2002).

La ciencia prospera gracias a la transparencia, la reproducibilidad y la colaboración, todos los cuales se ven obstaculizados por el *software* no libre (Morin et al., 2012). Sin acceso al código fuente, los científicos no pueden verificar la exactitud del código utilizado en la investigación, lo que puede llevar a que errores o sesgos en los resultados pasen desapercibidos. La incapacidad de modificar y redistribuir los programas empleados dificulta la reproducibilidad de los experimen-

tos y resultados científicos, haciendo imposible que otros investigadores verifiquen los hallazgos de manera independiente. Además, la naturaleza colaborativa de la ciencia se ve obstruida, ya que no puede ser compartido ni adaptado libremente para satisfacer diferentes necesidades de investigación.

Las implicaciones éticas del *software* no libre son significativas. Puede crear disparidades en el acceso a la tecnología, ya que las instituciones y los investigadores que no tengan capacidad para desarrollar un programa específico para sus necesidades se encuentran en desventaja, exacerbando la desigualdad en la investigación científica. Además, puede incluir características que monitorean y controlan a los usuarios, limitando su privacidad y autonomía (Stallman, 2013), o introducir de forma forzosa modificaciones en el programa instalado que pueden privar de funcionalidad o incluir alguna no deseada (Free Software Foundation [F.S.F.], 2024). Esta vigilancia es incompatible con los estándares éticos de la integridad científica y la dignidad del usuario.

El *software* libre aborda estos problemas garantizando varias libertades esenciales (Stallman, 2002). Los usuarios tienen derecho a ejecutar el programa para cualquier propósito, promoviendo la accesibilidad y la inclusión. Con acceso al código fuente, es posible entender cómo funciona y realizar mejoras, fomentando la innovación y el aprendizaje. También pueden compartirlo, facilitando la colaboración y la difusión del conocimiento. Al estar permitido compartir las contribuciones con la comunidad, se impulsan el progreso y el perfeccionamiento colectivos. En el contexto de la ciencia, estas libertades son indispensables. Aseguran que las herramientas empleadas sean transparentes, reproducibles y equitativas, y permite a los investigadores colaborar abiertamente, verificar resultados rigurosamente y adaptar el código a sus necesidades específicas sin restricciones. El *software* libre no solo rectifica estas injusticias sino que también mejora el proceso científico, convirtiéndolo en la única opción viable.

2. Objetivos

Dada la necesidad de acceder a los datos crudos obtenidos mediante medidas de citometría de masas, el objetivo general de este Trabajo Fin de Grado es el de desarrollar una solución bioinformática que permita visualizar y analizar los datos obtenidos mediante esta técnica antes de ningún tipo de procesamiento. Para ello, este objetivo principal se divide en cuatro objetivos parciales:

2.1. Reducción del tamaño de los archivos de datos

Desarrollar un método informático innovador para optimizar el tamaño de los archivos de datos generados, permitiendo una gestión eficiente y reduciendo la carga computacional.

2.2. Visualización de los datos crudos en gráficas intensidad frente a tiempo

Implementar una herramienta de visualización que muestre los datos en gráficas de intensidad frente a tiempo, facilitando la interpretación y el análisis de los resultados obtenidos.

2.3. Detección e integración de los eventos

Diseñar algoritmos precisos para la detección y la integración eficiente de los eventos relevantes en los datos citométricos, mejorando la capacidad de identificar y caracterizar adecuadamente los eventos de interés.

2.4. Cuantificación

Desarrollar un sistema de cuantificación precisa mediante métodos de patrón interno, garantizando una evaluación exacta de la concentración de los analitos en cada evento detectado.

3. Material y métodos

3.1. Instrumentación

Las medidas experimentales para la obtención de los datos utilizados para el desarrollo se llevaron a cabo en un CyTOF XT (Standard Biotools). Este equipo, recientemente instalado en la Universidad de Oviedo, cuenta con un sistema automático de introducción de muestras con detección de obstrucciones. Este sistema mantiene las muestras en frío y las resuspende justo antes del análisis. Además, a cada muestra agrega una cantidad de una suspensión de micropartículas cargadas con seis metales (itrio (^{89}Y), indio (^{115}In), cerio (^{140}Ce), terbio (^{159}Tb), lutecio (^{175}Lu) y bismuto (^{209}Bi)) para normalización de los datos.

3.2. Formato de los datos crudos

Los archivos originales utilizan un formato de tabla *Tab-Separated Values* (TSV), donde cada fila está delimitada por un salto de línea y cada columna por una tabulación.

3.3. Marcado de anticuerpos con metales

El marcaje de los anticuerpos se llevó a cabo utilizando el kit de marcaje MaxPar X8 de Standard Biotools. Este procedimiento se basa en la unión de un polímero orgánico al anticuerpo a través de uno o varios grupos tiol en la región constante del anticuerpo que se liberan mediante una reducción parcial del mismo en condiciones suaves. El polímero es capaz de alojar en su estructura mediante complejos de tipo quelato un número de átomos de elementos lantánidos, que en la bibliografía varía entre 20 y 40 (Zhang et al., 2020), de manera que cada molécula de anticuerpo se marca con varias decenas de átomos del metal. Para evitar ocupar innecesariamente canales innecesarios en la medida del CyTOF, se utiliza un isótopo enriquecido del lantánido.

Brevemente, el procedimiento experimental del marcaje consiste, en primer lugar, en una incubación del polímero con el lantánido isotópicamente enriquecido para que este se introduzca en los huecos de coordinación. Mientras tanto, el anticuerpo se somete a la reducción parcial con tris(2-carboxietil)fosfina durante 30 minutos a 37^{o}C para exponer los grupos tiol. Una vez terminada la reducción y la carga del polímero, se incuba el anticuerpo parcialmente reducido con el polímero cargado y se realizan varios lavados utilizando filtros de corte de 50 kDa para eliminar el exceso de polímero no unido.

Una vez finalizado el marcaje, el anticuerpo se cuantifica mediante espectrofotometría de abosrción UV en un equipo NanoDrop (Thermo Fisher).

3.4. Preparación de células

Las células de interés se prepararon a partir de cultivos celulares o tejidos primarios siguiendo las instrucciones del fabricante (Fluidigm, 2021). La tinción de viabilidad se realizó antes de la fijación. Las células se resuspendieron en Maxpar Cell Staining Buffer y se alicuotaron en tubos individuales de polipropileno de 5 mL a una concentración de 1–3 millones de células por 50 µL por muestra.

3.5. Tinción de células con anticuerpos de superficie

Se preparó un cóctel de anticuerpos 2X en Maxpar Cell Staining Buffer, siguiendo las instrucciones del fabricante (Fluidigm, 2021). Se añadieron 50 µL del cóctel de anticuerpos 2X a cada tubo, resultando en un volumen total de tinción de 100 µL. Se agitaron suavemente los tubos y se incubaron a temperatura ambiente durante 15 minutos. Se agitaron nuevamente las muestras y se continuó la incubación a temperatura ambiente durante otros 15 minutos. Después de la incubación, se lavaron las muestras dos veces con 2 mL de Maxpar Cell Staining Buffer, se centrifugaron a 300 g durante 5 minutos y se retiró el sobrenadante por aspiración.

3.6. Fijación de células

Se siguió el protocolo del fabricante (Fluidigm, 2021), para lo cual se preparó una solución fresca de formaldehído (FA) al 1.6 % a partir de un stock de formaldehído al 16 %. Se añadió 1 mL de la solución de FA al 1.6 % a cada tubo que contenía 1–3 millones de células. Se agitaron suavemente los tubos para mezclar bien y se incubaron las células a temperatura ambiente durante 10 minutos. Se centrifugaron las células a 800 x g durante 5 minutos, se aspiró el sobrenadante y se agitaron suavemente para resuspender las células en el volumen residual.

3.7. Tinción de células con *Cell-ID Intercalator*-Ir

Para poder detectar una señal correspondiente a cada célula, se utiliza un complejo de Ir capaz de intercalarse en el ADN de las células previamente permeabilizadas. Para ello se preparó una solución de intercalación, siguiendo las instrucciones del fabricante (Fluidigm, 2021), diluyendo Cell-ID Intercalator-Ir en Maxpar Fix and Perm Buffer a una concentración final de 125 nM. Se añadió 1 mL de la solución de intercalación a cada tubo que contenía 1–3 millones de células y se agitaron suavemente para asegurar una resuspensión completa. Se incubaron las muestras a temperatura ambiente durante 1 hora y se almacenaron a 2–8°C durante la noche.

Las muestras se pueden almacenar en la solución de intercalación a 2–8°C hasta por 48 horas antes de la adquisición.

3.8. Muestras

La muestra analizada es de un paciente anónimo con leucemia linfocítica crónica. Esta muestra es un resto no utilizado de otro proyecto de investigación y se ha recogido con el debido consentimiento informado del paciente y siguiendo los trámites necesarios del Comité de Ética de la Universidad de Oviedo.

La proteína *cluster of differentiation 20* (CD20) es un marcador de superficie de células B que interviene en el desarrollo y diferenciación de esta células y es un objetivo destacado para la terapia basada en anticuerpos de neoplasias malignas de células B, ya que se encuentra sobreexpresada en la mayoría de los pacientes con leucemia linfocítica crónica (Vlaming et al., 2021).

Para su análisis, las células fueron tratadas con anticuerpo anti-CD20, que previamente había sido marcado con el isótopo ^{175}Lu, como se ha indicado en el apartado 3.3. Además, las células fueron marcadas con el intercalador de Ir, como se indicó en el apartado 3.7.

3.9. Validación

La validación de los resultados obtenidos mediante el software desarrollado se llevó a cabo mediante comparación de estos resultados con los obtenidos mediante la técnica de referencia, SC-ICP-MS con un equipo de ICP-MS de triple cuadrupolo y utilizando los algoritmos estandarizados del grupo de investigación. Dado que en SC-ICP-MS los tiempos de integración son mucho más largos que en CyTOF (5 ms frente a 13 µs), los eventos están constituidos por un único punto de medida y el procesamiento de los datos se facilita. En este caso, para el filtrado de los eventos se considera que son eventos aquellos cuya intensidad supera la del fondo en tres veces su desviación estándar y se aplica un procedimiento iterativo mediante Microsoft Excel.

Para poder establecer esta comparación, se utilizaron dos muestras diferentes de un mismo paciente de leucemia linfocítica crónica. Una de ellas fue medida en CyTOF y la otra en SC-ICP-MS. Los datos de CyTOF fueron procesados con el software desarrollado, mientras que los de SC-ICP-MS se procesaron con el procedimiento mencionado. Finalmente, las poblaciones celulares obtenidas mediante las dos técnicas fueron comparadas en términos de la masa de ^{175}Lu.

3.10. Código

Todos los programas se desarrollaron en el lenguaje de programación Python 3 (versión del intérprete 3.11.8) (Van Rossum & Drake, 2009).

Se usó Numpy (versión 1.26.4) (C. R. Harris et al., 2020) y pandas (versión 1.5.3) (McKinney, 2010) para la carga y procesado de los datos.

Las interfaces gráficas funcionan utilizando la librería estándar Tkinter (versión 8.6) (Van Rossum, 2022).

Las gráficas fueron generadas usando Matplotlib (versión 3.8.3) (Hunter, 2007).

En adelante, los nombres de variables, palabras reservadas y otros términos utilizados dentro del código que son literales y no siguen necesariamente las normas gramaticales se escriben en una fuente monoespaciada para facilitar la lectura. E.g.: `foobar`.

3.11. Uso de *software* libre

Todas las herramientas desarrollados utilizan únicamente librerías de código libre, según la guía de la Free Software Foundation (F.S.F., 2023).

Todo el código escrito se comparte bajo la licencia GNU General Public License (véase https://www.gnu.org/licenses/), tal como fue publicada por la Free Software Foundation, tanto la versión 3 de la licencia como cualquier versión posterior.

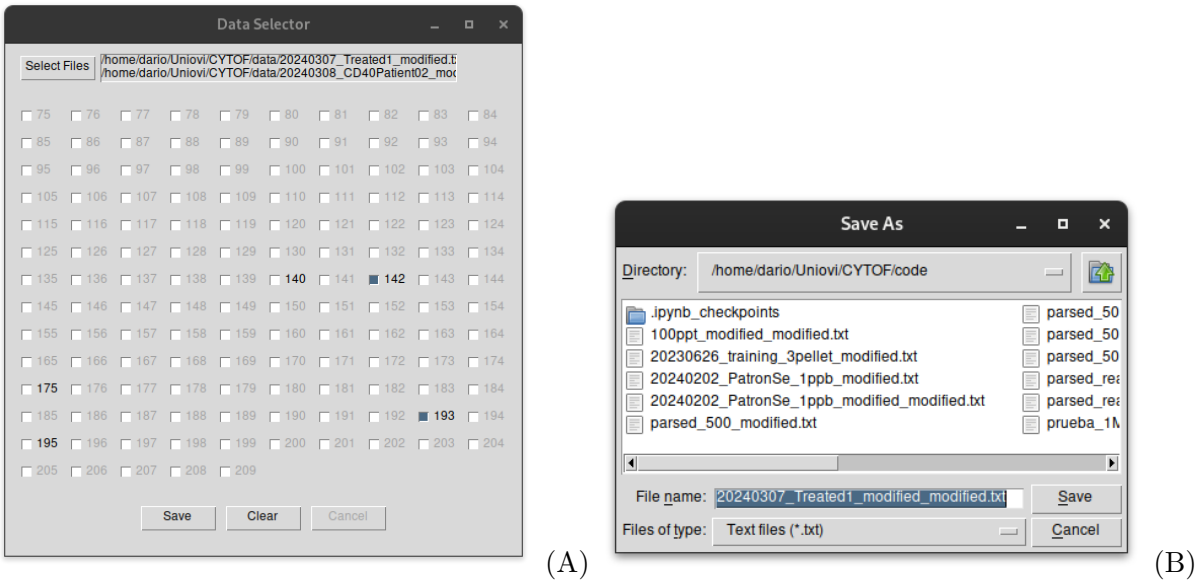
4. Resultados y discusión

4.1. Reducción del tamaño de los archivos

Se consideró necesario el desarrollo de un método para la selección de las masas de interés a partir de los datos crudos para facilitar la transferencia, almacenado y procesado de los archivos, debido al gran tamaño de los originales.

Para facilitar el manejo de los usuarios, el selector de masas tiene una aplicación de interfaz gráfica de usuario (GUI) escrita en Python utilizando la biblioteca estándar. El código fuente está disponible en el apéndice A y en `data_selector.py`.

La aplicación permite al usuario seleccionar uno o varios archivos de entrada y elegir qué masas de una serie predefinida (del 75 al 209) desea conservar en esos archivos, como se puede ver en la figura 4.A. Luego, la aplicación guarda una versión modificada de los archivos de entrada, conservando solo las masas seleccionadas por el usuario. La interfaz de guardadado se puede ver en la figura 4.B.



(A) (B)

Figura 4: Interfaz del selector de masas. (A) GUI princial, con unos archivos de ejemplo seleccionados. Solo aparecen resaltadas las masas disponibles en ambos archivos. Las masas 142 y 193 están marcadas para su guardado. (B) GUI generada por Tkinter para la escritura de los archivos. El nombre sugerido añade "`_modified`" al original.

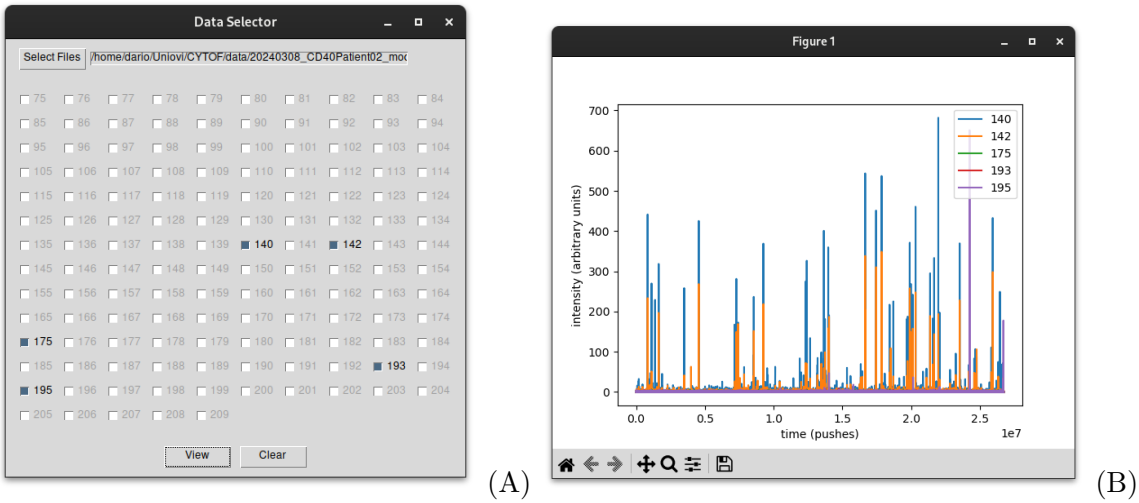
Internamente, el programa lee el archivo de entrada (`infile`) para obtener la primera línea, la cual contenga información sobre las masas disponibles en el archivo. Luego, extrae las masas disponibles de esa línea y las guarda en una lista. Después, calcula dinámicamente los índices de las columnas que corresponden a las masas especificadas por el usuario encontrando

los índices de las masas especificadas dentro de la lista de masas disponibles y abre el archivo de salida (`outfile`) para escribir datos en él. Escribe una línea de encabezado en el archivo de salida que enumera las masas seleccionadas. A continuación, lee el archivo de entrada en "trozos" (`chunks`) utilizando pandas. Esto se hace para manejar archivos grandes de manera eficiente. Los datos se leen por partes para no agotar la memoria. Para cada trozo leído, se seleccionan las columnas especificadas por los índices calculados anteriormente y las escribe en el archivo de salida. Paralelamente, se utiliza una operación de generador (`yield`) para calcular el progreso del proceso. Devuelve un valor en porcentaje calculado a partir del índice de la última línea procesada en relación con el número total de líneas en el archivo de entrada.

4.2. Visualización de los datos

Para la visualización de los datos extraídos se creó un programa para seleccionar datos específicos de un archivo, procesarlos y visualizarlos con un gráfico, mientras proporciona manejo de errores y retroalimentación clara al usuario. El código original está disponible en el apéndice B `data_viewer.py`.

Se desarrolló una GUI diseñada para seleccionar datos de un archivo (véase figura 5.A) y visualizarlos utilizando un gráfico (véase figura 5.B). Utiliza la biblioteca `tkinter` para los componentes de la GUI y `matplotlib` para graficar los datos.



(A) (B)

Figura 5: Interfaz del visor de datos. (A) GUI princial, con un archivo de ejemplo seleccionados. Solo aparecen resaltadas las masas disponibles, todas seleccionadas para ser graficadas. (B) GUI generada por `matplotlib` de los datos. En la parte inferior están las herramientas de navegación de la gráfica para desplazarse por ella, hacer *zoom*, navegar entre las vistas o guardar la gráfica en una archivo de imagen.

El programa comienza creando una ventana principal utilizando `tkinter` y configura varios marcos y widgets dentro de esta ventana. El primer marco incluye un botón para seleccionar un archivo y una etiqueta para mostrar la ruta del archivo seleccionado.

El programa proporciona botones para seleccionar números (masas) del 75 al 209. Estos botones están inicialmente deshabilitados. Cuando se selecciona un archivo, el programa lee la primera línea del archivo para determinar cuáles de estas masas están disponibles. Luego, habilita solo aquellos correspondientes a los números disponibles.

Cuando el usuario selecciona un archivo y algunos números (masas), puede hacer clic en el botón `View` para generar un gráfico. Al hacer esto se lee el archivo seleccionado para extraer las columnas de datos relevantes según los números seleccionados. Acto seguido se grafican los datos utilizando `matplotlib` en una subventana nueva, donde cada número seleccionado se grafica en el eje y contra un eje x común.

Adicionalmente, hay un botón `Clear` para restablecer todos los botones de verificación a su estado desmarcado.

Esta visualización no está disponible en el *software* propietario de CyTOF y permite llevar a cabo un análisis preliminar de los datos, comprobando su distribución en el tiempo y los perfiles de los eventos.

El gráfico generado representa unidades de intensidad en el eje vertical frente a "*pushes*.[en] el eje horizontal. Un *push* corresponde a un paquete de iones que es empujado hacia el analizador de TOF. Cada *push* tiene una duración de 13 µs, por lo que las unidades del eje horizontal son equivalentes a tiempo de análisis.

4.3. Calibrado instrumental

Para obtener una calibración que permitiera calcular la masa de un metal en una célula a partir de la intensidad de pico obtenida, se desarrolló una GUI para realizar calibraciones utilizando regresión de mínimos cuadrados ponderados. El código original está disponible en el apéndice C `data_calibrator.py`.

La aplicación aprovecha la biblioteca Tkinter para crear la GUI, NumPy para los cálculos numéricos y Matplotlib para trazar los resultados de la calibración. Las funcionalidades principales están encapsuladas dentro de la clase `CalibrationApp`, que gestiona la interfaz y el proceso de calibración.

La GUI consta de una ventana principal (véase figura 6) que permite al usuario agregar filas dinámicamente para seleccionar archivos de datos e ingresar concentraciones. Cada fila

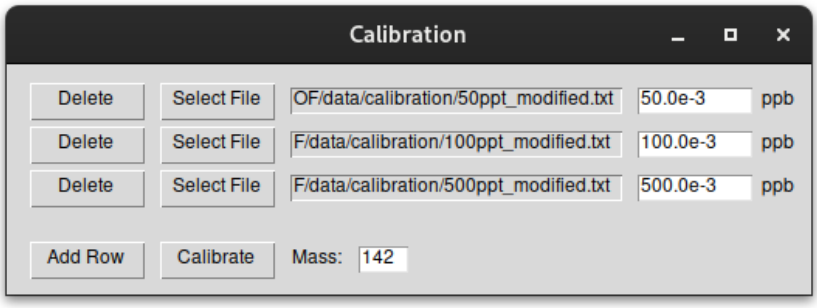


Figura 6: Interfaz de la aplicación de calibración. GUI princial, con varios archivos seleccionados. Al lado de cada archivo están introducidas las concentraciones y en la parte inferior la masa a analizar.

corresponde a un punto del calibrado y contiene un botón para la selección de archivos, una etiqueta que muestra la ruta del archivo elegido y un widget de entrada para ingresar la concentración correspondiente. Los usuarios pueden agregar y eliminar filas según sea necesario. Además, un panel de control en la parte inferior de la ventana proporciona botones para agregar nuevas filas e iniciar el proceso de calibración, junto con un widget de entrada para especificar la masa utilizada en la calibración.

Como ya se ha comentado, la calibración se lleva a cabo mediante la medida de patrones iónicos de concentraciones conocidas del elemento. Dado que estos patrones llegan al CyTOF de manera continua con el tiempo, generan una señal constante, afectada por únicamente por el ruido instrumental, por lo que, a partir de los archivos de datos obtenidos para estas medidas, se debe calcular la intensidad media para cada patrón. Para ello, el proceso de calibración comienza con la selección de archivos y la extracción de datos. Los usuarios seleccionan archivos de datos que contienen mediciones de señal, y la aplicación los lee, extrayendo las señales relevantes para la masa especificada y calculando la media y la desviación estándar. La concentración en ppb es la introducida por el usuario en la caja de texto a la derecha de cada fila.

Una vez pulsado el botón de calibrar, el programa computa la regresión por mínimos cuadrados ponderados de la concentración frente a la señal, utilizando como pesos el inverso del cuadrado de las desviaciones estándar.

Los resultados de la calibración, incluida la línea de regresión y los puntos de calibración con barras de error, se visualizan utilizando Matplotlib. El gráfico muestra la relación entre la concentración y la intensidad de la señal, anotada con la ecuación de regresión y el valor de r^2.

4.4. Detección e integración de eventos

La señal analítica sin procesar consiste en una serie de datos de intensidad frente al tiempo (ver figura 2), en la que se incluye tanto los eventos generados por cada célula, como el fondo iónico del elemento en disolución. Por lo tanto, de esta serie de datos deben extraerse e integrarse aquellos que pertenecen a cada célula.

Para la detección de los eventos se utilizaron los datos crudos con ruido de fondo y un algoritmo de procesamiento de señales de la librería `scikit`. Para ello se definió inicialmente la amplitud de los picos utilizando una masa "de selección", para la cual los eventos tienen que tener una duración de entre 10 y 150 puntos, obteniendo una lista de posiciones en las que el evento comienza y termina. Posteriormente, se seleccionan como válidos aquellos que cumplan ciertas condiciones adicionales, como el poseer más de una cantidad determinada de algunos elementos (que la intensidad para ciertos isótopos supere un cierto valor límite). Este método presenta la ventaja de poder ser reutilizado para casos en los que se empleen los datos "preprocesados" por el *software* del CyTOF, ya que haciendo cambios mínimos en los parámetros de detección y filtrado, se pueden incluir los eventos con una menor altura de pico o dar a entender al algoritmo que todo número positivo puede formar parte de un evento al no haber fondo.

La integración de los picos se hace para cierta masa "de interés". El código selecciona cada "región de pico" en la matriz original de los datos, hace el sumatorio para la masa y devuelve una lista con las integrales correspondientes a cada evento.

Una vez integradas las intensidades de los picos, estas se interpolan en el calibrado obtenido en el apartado 4.3, de manera que se obtiene la masa de metal en cada evento celular mediante la ecuación 2:

$$m = \frac{\sum(I - X_0)}{X_1} \cdot \nu \cdot dt \cdot TE \tag{2}$$

donde m será la masa de metal en la célula, I cada una de las intensidades detectadas para el evento, X_0 y X_1 la ordenada en el origen y la pendiente calculadas en la regresión, ν el flujo, dt el tiempo de integración y TE la eficiencia de transporte.

Debido al estado preliminar de este *software* y a que se consideran distintas implementaciones para realizar la tarea, así como la posibilidad de definir manualmente distintas condiciones de detección y filtrado que difícilmente se pueden acomodar a una interfaz gráfica, el código se adaptó a un *notebook* de Jupyter que permite interpretar y modificar el código de forma dinámica utilizando `IPython`. Una versión de *script* de Python equivalente está disponible en el apéndice D. El *notebook* original se puede encontrar en `data_detector.ipynb`.

4.5. Manejo de errores

Los programas incluyen comprobaciones de errores para asegurar una operación sin problemas: muestra mensajes de error si no se selecciona un archivo o números al intentar generar un gráfico o leer datos y maneja errores de lectura de archivos mostrando mensajes de error adecuados.

En el caso de la calibración, se verifica la presencia de una masa seleccionada, asegura que se proporcionen al menos tres puntos de datos para la calibración y verifica que las concentraciones ingresadas puedan convertirse a valores numéricos válidos. Se muestran mensajes de error apropiados en caso de entradas no válidas o errores de lectura de archivos.

4.6. Aplicación a un conjunto de datos experimentales

Se analizaron los datos obtenidos de células de paciente marcadas con ^{193}Ir y ^{175}Lu. El complejo de iridio que se intercala en el ADN se utilizó para filtrar todas las células, mientras que el lutecio con el que se marcó el anticuerpo anti-CD20 se utilizó para identificar aquellas que sobreexpresan CD20.

En primer lugar, se realizó el filtrado del archivo original para que incluya solamente las masas de interés utilizando `data_selector.py`. El conjunto original de datos pesaba 9.2 GB y el archivo modificado 443 MB.

A continuación, se realizó la visualización de los datos con `data_viewer.py`, como se muestra en la figura 7, donde se presenta una vista general de los datos y un zoom que destaca dos evento con las masas correspondientes. Se puede observar como ambos contienen ^{193}Ir, pero solo uno de ellos presenta además ^{175}Lu. Esta visualización, que no está disponible en el software propietario del CyTOF, permite hacer una comprobación rápida de la presencia de eventos y su calidad así como la presencia de varios isótopos en un solo evento.

Posteriormente, se construyó la recta de calibrado, utilizando `data_calibrator.py`, a partir del análisis de tres patrones elementales de concentraciones conocidas de lutecio entre 0.5 y 5 ppb, que puede observarse en la figura 8.

Mediante `data_detector.ipynb`, se llevó a cabo el filtrado de los eventos utilizando la masa ^{193}Ir, que debe estar presente en todas las células debido al intercalador de iridio utilizado en el marcaje. A partir de los eventos encontrados de esta forma, se obtuvo el área integrada para la masa ^{175}Lu, correspondiente a la marca utilizada en el anticuerpo anti-CD20. Estas integrales se interpolaron aplicando los coeficientes de la regresión en la ecuación 2 para obtener una lista de las masas de ^{175}Lu por célula, utilizando para definir los eventos la masa de ^{193}Ir a

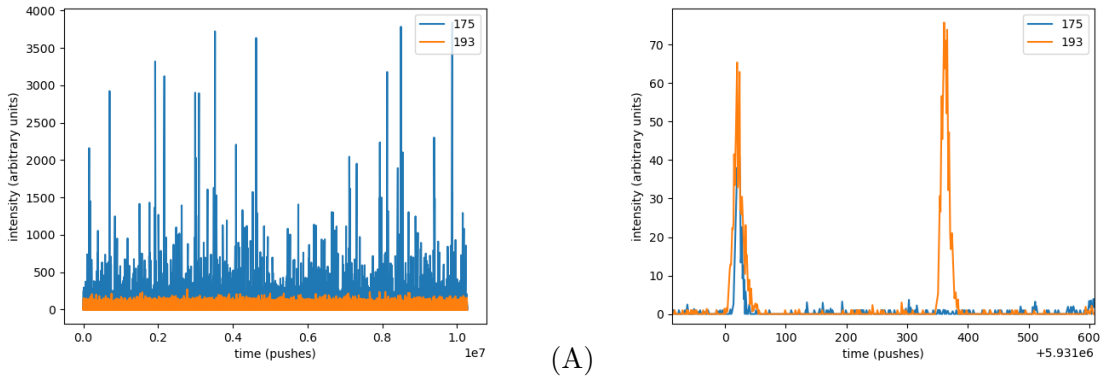


Figura 7: Gráficas generadas a partir de `data_viewer.py`. (A) Vista general de los datos. (B) Zoom en el que se pueden apreciar dos eventos de ^{193}Ir, uno con ^{175}Lu (a la izquierda) y otro sin él (derecha).

modo de selección.

Un esquema general de todo el proceso se puede observar en la figura 9.

Este conjunto de datos generado mediante el código se compararó con el obtenido mediante el método que se describió en la sección 3.9 por SC-ICP-MS, para lo cual se realizó un *boxplot* que permite comparar la dos distribución, tal como se muestra en la figura 10.

Pese a la diferencia en la dispersión de los datos, las medianas obtenidas y su rango intercuartílico (IQR) están en el mismo orden de magnitud (SC-ICP-MS: 9 fg (IQR: 3 - 26 fg), CyTOF: 12 fg (IQR: 10 - 17 fg)). Esto sugiere que el programa está tratando adecuadamente los datos obtenidos mediante el citómetro de masas.

En los datos obtenidos mediante SC-ICP-MS, se observa una mayor dispersión de los eventos, posiblemente debido a la incapacidad de descartar eventos generados por la llegada simultánea de dos o más células.

Estos eventos múltiples tienden a incrementar el tamaño de la nube iónica, lo que también prolonga la duración del evento. Como se mencionó en el apartado 4.4, en el proceso de integración de los datos de CyTOF se han utilizado solo los eventos cuya duración está entre 10 y 150 puntos, descartando aquellos por debajo y por encima de este rango, eliminando por tanto la mayoría de los picos correspondientes a múltiples células. Sin embargo, en el caso de SC-ICP-MS, no es posible eliminar estos eventos, ya que los tiempos de integración superiores hacen que solo haya un punto por evento, con lo que algunos de los puntos corresponden a múltiples células en lugar de a una sola.

Para comprobar la posibilidad de registrar eventos múltiples, así como de detectar distintas poblaciones celulares se hizo un *scatter plot* de los datos de la masa "de selección" y la "de

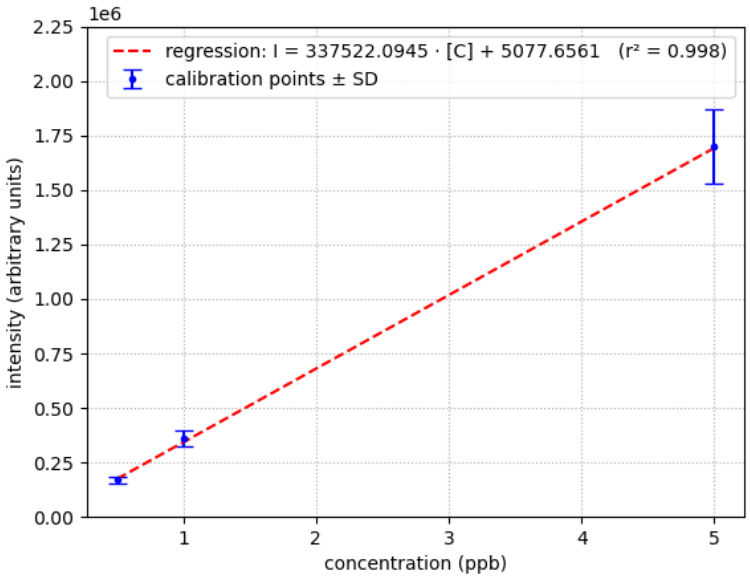


Figura 8: Puntos de calibración y regresión por mínimos cuadrados ponderados de masas frente a intensidad de ^{175}Lu.

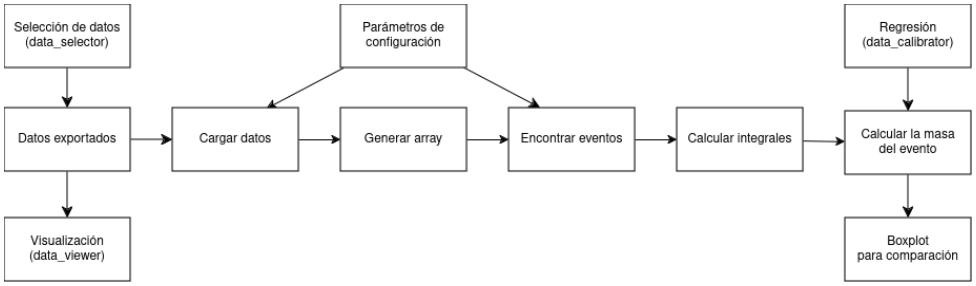


Figura 9: Diagrama de bloques del procesado de los datos y su interacción con otros programas. La fila central es el detector de eventos (`data_detector.py`). Cuando se utiliza otro programa se menciona su nombre entre paréntesis.

interés". En la figura 11 se distingue una población de células con una muy baja concentración de lutecio (población "A" en la figura) y una con una mayor concentración del susodicho metal (población "B" en la figura). La población "A", por lo tanto, se puede considerar negativa en la expresión de CD20, mientras que la población "B" puede considerarse positiva en este marcador. Esto es una muestra que viene de un paciente, por lo que no solo hay linfocitos que expresan CD20, sino que puede haber otros tipos celulares que no expresen este marcador o lo hagan en menor cantidad. Aquellas intensamente marcadas son las que consideraríamos CD20+.

La presencia de estas dos poblaciones diferentes era ya previsible desde la visualización de los datos, como se advertía en la figura 7.B. El pico de la derecha correspondería a la población "A" y el pico de la izquierda a la población "B".

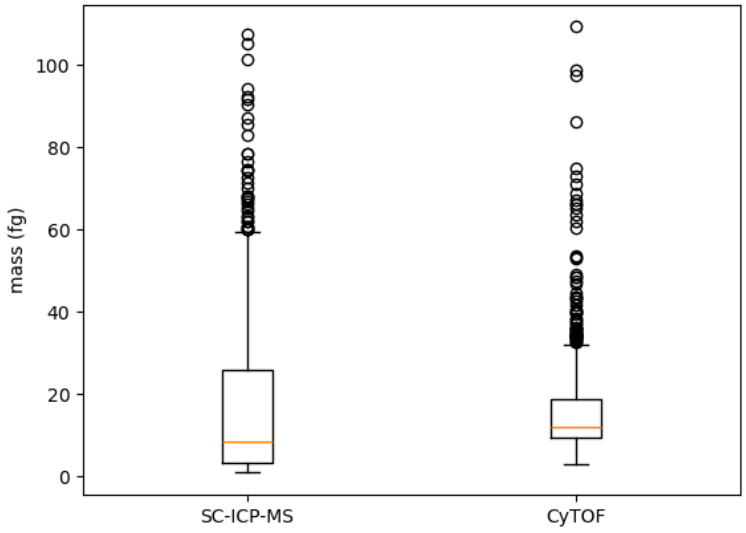


Figura 10: "*Boxplot*" comparando la distribución de las masas de ^{175}Lu por célula obtenidos mediante SC-ICP-MS y CyTOF.

Se advierte la presencia de un tercer conjunto de evento , marcados "C" en la figura, que tienen una alta intensidad en el marcador empleado para la definición del evento celular, ^{193}Ir. Este tercer grupo puede corresponder tanto a eventos que incluyan más de una célula como a un tercer tipo celular de mayor tamaño. Esta posibilidad no se descarta debibo a la complejidad de una muestra real de un paciente. La otra opción es que este grupo corresponda a eventos múltiples en CyTOF. Se comprobó por un filtrado de los datos con `numpy` que estos eventos solo representan un 6 % de los eventos totales.

La proporción de posibles eventos múltiples en CyTOF es muy baja con respecto a la que se puede esperar en SC-ICP-MS, lo cual vendría a reforzar la idea de que las diferencias de distribución en el *boxplot* son fruto de la presencia de datos adicionales de alta masa en el SC-ICP-MS.

4.7. Fortalezas

La innovación metodológica es destacable en este trabajo, ya que propone un enfoque que ofrece nuevas perspectivas en las técnicas de análisis celular.

La investigación está bien fundamentada en conceptos teóricos sólidos y utiliza técnicas avanzadas de espectrometría de masas para asegurar la validez de los resultados obtenidos.

La relevancia biomédica del trabajo es notable, ya que la capacidad de analizar células in-

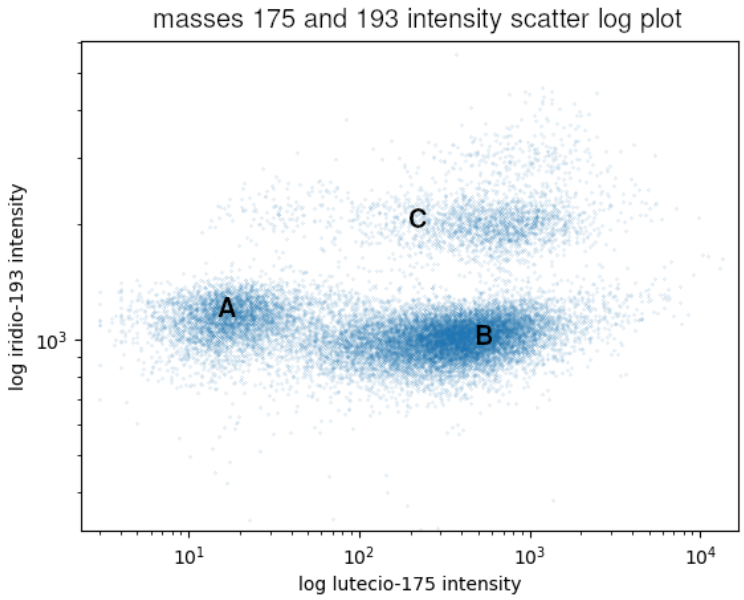


Figura 11: "*Scatter plot*" en el que se distinguen una población marcada con poco lutecio e iridio (A), una intensamente marcada con lutecio con baja concentración de iridio (B) y una intensamente marcada con lutecio con una alta concentración de iridio (C).

dividuales con una resolución tan detallada tiene implicaciones significativas para el diagnóstico, incluyendo la identificación de biomarcadores y el avance en terapias personalizadas.

La integración de la espectrometría de masas de plasma acoplado inductivamente en la citometría ofrece ventajas sobre aquella de flujo con marcadores fluorescentes convencionales al permitir la detección simultánea de múltiples elementos en una única célula y con ello un análisis más detallado.

El uso de *software* libre permite a cualquier investigador que desee proseguir con el estudio analizar de forma transparente el trabajo realizado y continuarlo sin necesidad de reimplementar ninguna parte o encontrarse con limitaciones en el uso de aquellas ya desarrolladas.

4.8. Debilidades

El tamaño de los trozos de archivo empleados en el procesamiento no debería estar incrustrado en el código, sino que debería calcularse en función del *hardware* (principalmente la memoria RAM), así como de los recursos disponibles en ese momento. Se consideró que este refinamiento excede el objetivo de este trabajo.

La validación de los resultados aún no posee una suficiente robustez estadística. La reciente iinstalación del equipo en la Universidad de Oviedo y el aún limitado número de muestras al que

se ha tenido acceso han limitado la cantidad de medidas experimentales que se han podido utilizar en este Trabajo. Se ha de realizar un estudio en el que se midan las células del mismo paciente, marcadas a la vez, así como idear un método que permita descartar los eventos múltiples en SC-ICP-MS.

La implementación de la técnica propuesto utilizando CyTOF puede ser compleja, al requerir un equipamiento especializado y costoso y un conocimiento avanzado, dificultando su adopción generalizada en los laboratorios interesados en el análisis celular de este tipo.

5. Conclusiones

El programa desarrollado en este Trabajo permite a los investigadores, en primer lugar, tener acceso a estos datos crudos que, de otra forma, quedarían ocultos en los archivos internos del *software* propietario. Por otro lado, permite tener un control total de las operaciones de tratamiento de los datos, que no siempre se detallan en los programas dedicados de la compañía. Además, da acceso a toda la información de las medidas, incluyendo no solo la intensidad integrada de cada evento, sino también a las duraciones, perfiles e información sobre el fondo causado por especies iónicas, lo que aumenta el nivel de informacion analítica al que se puede acceder y por tanto a la información biomédica asociada.

6. Bibliografía

Álvarez-Fernández García, R. (2020, octubre). *Análisis de pequeñas estructuras mediante ICP-MS: desarrollo metodológico y aplicaciones* [Tesis doctoral]. Consultado el 27 de abril de 2024, desde https://digibuo.uniovi.es/dspace/handle/10651/57940

Bandura, D. R., Baranov, V. I., Ornatsky, O. I., Antonov, A., Kinach, R., Lou, X., Pavlov, S., Vorobiev, S., Dick, J. E., & Tanner, S. D. (2009). Mass Cytometry: Technique for Real Time Single Cell Multitarget Immunoassay Based on Inductively Coupled Plasma Time-of-Flight Mass Spectrometry. *Analytical Chemistry*, *81*(16), 6813-6822. https://doi.org/10.1021/ac901049w

Douvris, C., Vaughan, T., Bussan, D., Bartzas, G., & Thomas, R. (2023). How ICP-OES Changed the Face of Trace Element Analysis: Review of the Global Application Landscape. *Science of The Total Environment*, *905*, 167242. https://doi.org/10.1016/j.scitotenv.2023.167242

Fluidigm. (2021). Maxpar Cell Surface Staining with Fresh Fix. https://www.standardbio.com/asset/332

F.S.F. (2023). Various Licenses and Comments about Them - GNU Project - Free Software Foundation. Consultado el 10 de junio de 2024, desde https://www.gnu.org/licenses/license-list.html%5C#SoftwareLicenses

F.S.F. (2024). Proprietary Software Is Often Malware. Consultado el 16 de junio de 2024, desde https://www.gnu.org/proprietary/

Fulwyler, M. J. (1965). Electronic Separation of Biological Cells by Volume. *Science*, *150*(3698), 910-911. https://doi.org/10.1126/science.150.3698.910

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357-362. https://doi.org/10.1038/s41586-020-2649-2

Harris, D. C. (2010). *Quantitative Chemical Analysis*. W. H. Freeman. https://books.google.es/books?id=kIgLJ1De_jwC

Hou, X., Amais, R. S., Jones, B. T., & Donati, G. L. (2021). Inductively Coupled Plasma Optical Emission Spectrometry. En *Encyclopedia of Analytical Chemistry* (pp. 1-29). John Wiley & Sons, Ltd. https://doi.org/10.1002/9780470027318.a5110.pub4

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, *9*(3), 90-95. https://doi.org/10.1109/MCSE.2007.55

Kimball, A. K., Oko, L. M., Bullock, B. L., Nemenoff, R. A., van Dyk, L. F., & Clambey, E. T. (2018). A Beginner's Guide to Analyzing and Visualizing Mass Cytometry Data. *The Journal of Immunology*, *200*(1), 3-22. https://doi.org/10.4049/jimmunol.1701494

Liu, Y., Zhao, H., Fu, B., Jiang, S., Wang, J., & Wan, Y. (2022). Mapping Cell Phenomics with Multiparametric Flow Cytometry Assays. *Phenomics*, *2*(4), 272-281. https://doi.org/10.1007/s43657-021-00031-0

Maecker, H. T., McCoy, J. P., & Nussenblatt, R. (2012). Standardizing immunophenotyping for the Human Immunology Project. *Nature Reviews Immunology*, *12*(3), 191-200. https://doi.org/10.1038/nri3158

McKinney, W. (2010). Data Structures for Statistical Computing in Python. En S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 56-61). https://doi.org/10.25080/Majora-92bf1922-00a

McKinnon, K. M. (2018). Flow Cytometry: An Overview. *Current Protocols in Immunology*, *120*(1), 5.1.1-5.1.11. https://doi.org/10.1002/cpim.40

Morin, A., Urban, J., Adams, P. D., Foster, I., Sali, A., Baker, D., & Sliz, P. (2012). Shining Light into Black Boxes. *Science*, *336*(6078), 159-160. https://doi.org/10.1126/science.1218263

Newell, E. W., & Cheng, Y. (2016). Mass cytometry: Blessed with the curse of dimensionality. *Nature Immunology*, *17*(8), 890-895. https://doi.org/10.1038/ni.3485

Parra, E. R. (2018). Novel Platforms of Multiplexed Immunofluorescence for Study of Paraffin Tumor Tissues. *Journal of Cancer Treatment and Diagnosis*, *2*(1). https://doi.org/10.29245/2578-2967/2018/1.1122

Pereira, J. S. F., Álvarez-Fernández García, R., Corte-Rodríguez, M., Manteca, A., Bettmer, J., LeBlanc, K. L., Mester, Z., & Montes-Bayón, M. (2023). Towards Single Cell ICP-MS Normalized Quantitative Experiments Using Certified Selenized Yeast. *Talanta*, *252*, 123786. https://doi.org/10.1016/j.talanta.2022.123786

Rikkert, L. G., Beekman, P., Caro, J., Coumans, F. a. W., Enciso-Martinez, A., Jenster, G., Le Gac, S., Lee, W., van Leeuwen, T. G., Loozen, G. B., Nanou, A., Nieuwland, R., Offerhaus, H. L., Otto, C., Pegtel, D. M., Piontek, M. C., van der Pol, E., de Rond, L., Roos, W. H., ... Terstappen, L. W. M. M. (2020). Cancer-ID: Toward Identification of Cancer by Tumor-Derived Extracellular Vesicles in Blood. *Frontiers in Oncology*, *10*. https://doi.org/10.3389/fonc.2020.00608

Schmit, T., Klomp, M., & Khan, M. N. (2021). An Overview of Flow Cytometry: Its Principles and Applications in Allergic Disease Research. En K. Nagamoto-Combs (Ed.), *Animal Models of Allergic Disease: Methods and Protocols* (pp. 169-182). Springer US. https://doi.org/10.1007/978-1-0716-1001-5_13

Shapiro, H. M. (2004). The evolution of cytometers. *Cytometry Part A*, *58A*(1), 13-20. https://doi.org/10.1002/cyto.a.10111

Spidlen, J., Moore, W., Parks, D., Goldberg, M., Bray, C., Bierre, P., Gorombey, P., Hyun, B., Hubbard, M., Lange, S., Lefebvre, R., Leif, R., Novo, D., Ostruszka, L., Treister, A., Wood, J., Murphy, R. F., Roederer, M., Sudar, D., ... Brinkman, R. R. (2010). Data File Standard for Flow Cytometry, version FCS 3.1. *Cytometry Part A*, *77A*(1), 97-100. https://doi.org/10.1002/cyto.a.20825

Stallman, R. (2002). *Free Software, Free Society: Selected Essays of Richard M. Stallman*. GNU Press. https://www.gnu.org/philosophy/fsfs/rms-essays.pdf

Stallman, R. (2013). Free Software Is Even More Important Now. Consultado el 16 de junio de 2024, desde https://www.gnu.org/philosophy/free-software-even-more-important.html

Stallman, R. (2021). What Is Free Software? Consultado el 28 de junio de 2024, desde https://www.gnu.org/philosophy/free-sw.html

Tanner, S. D., Bandura, D. R., Ornatsky, O., Baranov, V. I., Nitz, M., & Winnik, M. A. (2008). Flow cytometer with mass spectrometer detection for massively multiplexed single-cell biomarker assay. *Pure and Applied Chemistry*, *80*(12), 2627-2641. https://doi.org/10.1351/pac200880122627

Theiner, S., Rodriguez, M. C., & Traub, H. (2022). 13 Novel Applications of Lanthanoids as Analytical or Diagnostic Tools in the Life Sciences by ICP-MS Based Techniques. En A. Golloch (Ed.), *Handbook of Rare Earth Elements* (pp. 399-444). De Gruyter. https://doi.org/doi:10.1515/9783110696455-013

Van Rossum, G. (2022). Graphical User Interfaces with Tk. En *The Python Library Reference, Release 3.11.8*. Python Software Foundation. https://docs.python.org/3.11/library/tk.html

Van Rossum, G., & Drake, F. L. (2009). *Python 3 Reference Manual*. CreateSpace.

Vlaming, M., Bilemjian, V., Freile, J. Á., Lourens, H. J., van Rooij, N., Huls, G., van Meerten, T., de Bruyn, M., & Bremer, E. (2021). CD20 positive CD8 T cells are a unique and transcriptionally-distinct subset of T cells with distinct transmigration properties. *Scientific Reports*, *11*(1), 20499. https://doi.org/10.1038/s41598-021-00007-0

Zhang, Y., Zabinyakov, N., Majonis, D., Bouzekri, A., Ornatsky, O., Baranov, V., & Winnik, M. A. (2020). Tantalum Oxide Nanoparticle-Based Mass Tag for Mass Cytometry. *Analytical Chemistry*, *92*(8), 5741-5749. https://doi.org/10.1021/acs.analchem.9b04970

Apéndices

En las siguiente secciones se incluyen los programas referidos a lo largo del documento. Los metadatos incluidos en el código sobre la licencia, la información del módulo y el autor han sido eliminados por concisión en todos los archivos. Todo el código se puede encontrar en el repositorio (https://github.com/UO282759/UO282759_TFG) y en el enlace a cada archivo en el apéndice correspondiente.

En el caso del detector de eventos del apéndice D, el archivo original es un `notebook` de Jupyter y para poder mostrar el código en este documento, se incluye una versión *script* equivalente del código con elipsis ("`...`") en las partes que se deberían modificar interactivamente en el programa.

Se informa de que Darío Bagüés Castro, quien escribe el presente Trabajo de Fin de Grado, es el creador del código y a quien corresponde atribución por derechos de autor.

A. Selector de datos

data_selector.py

```python
import mmap
import threading
import tkinter as tk
from tkinter import ttk, filedialog, messagebox

import pandas as pd


CHUNKSIZE = 1000000


def line_count(infile):
    """
    Count the number of lines in a file.

    Parameters:
    infile (str): The path to the input file.

    Returns:
    int: The number of lines in the file.
    """
    lines = 0
    with open(infile, "r+") as f:
        buf = mmap.mmap(f.fileno(), 0)
        while buf.readline():
            lines += 1
    return lines

def data_selector(masses, infile, outfile):
    """
    Select specific columns from a file based on given masses and save to a new file.

    Parameters:
    masses (list of int): The masses to select.
    infile (str): The path to the input file.
    outfile (str): The path to the output file.

    Yields:
    float: The progress percentage of the operation.
    """
    with open(infile) as infile_r:
        first_line = infile_r.readline().strip()
        available_masses = [int(x) for x in first_line.split()[2:]]
    columns = [0] + [available_masses.index(int(i)) for i in masses]
    lines = line_count(infile)

    with open(outfile, "w") as outfile:
        outfile.write("Push number\t" + "\t".join(str(m) for m in masses) + "\n")
        for chunk in pd.read_csv(infile, delimiter="\t", chunksize=CHUNKSIZE):
            chunk.iloc[:, columns].to_csv(outfile, index=False, header=False, sep="\t")
            yield 100 - (1 - chunk.index[-1] / lines) * 100

def save_selection():
    """
    Save the selected numbers to the output file(s).
    """
    if path_var.get() == "":
        messagebox.showerror("Error", "No file selected.")
        return
```

```
    selected_numbers = [number for number, var in number_vars.items() if var.get()]
    if selected_numbers:
        save_path = []
        for path in path_var.get().split("\n"):
            save_path.append(
                filedialog.asksaveasfilename(
                    initialdir=path,
                    initialfile="".join(path.split("/")[-1].split(".")[:-1])
                    + "_modified.txt",
                    defaultextension=".txt",
                    filetypes=[("Text files", "*.txt"), ("All files", "*.*")],
                )
            )
        if len(save_path) == len(path_var.get().split("\n")):
            save_button.config(state=tk.DISABLED)
            clear_button.config(state=tk.DISABLED)
            cancel_button.config(state=tk.NORMAL)
            global thread_running
            thread_running = True
            threading.Thread(
                target=long_running_function,
                args=(selected_numbers, path_var.get(), save_path),
            ).start()
        else:
            messagebox.showerror(
                "Error", "You need to select a save file for every input file."
            )
    else:
        messagebox.showerror("Error", "No numbers selected.")

def long_running_function(selected_numbers, paths, save_path):
    """
    Execute the data selection process for each file in a separate thread.

    Parameters:
    selected_numbers (list of int): The selected masses.
    paths (str): The input file paths.
    save_path (list of str): The output file paths.
    """
    paths = paths.split("\n")
    for n, path in enumerate(paths):
        if not thread_running:
            break
        progress_var.set(f"File: {n+1}/{len(paths)}\tGetting file size...")
        for progress in data_selector(selected_numbers, path, save_path[n]):
            if not thread_running:
                break
            progress_var.set(f"File: {n+1}/{len(paths)}\tProgress: {progress:.2f} %")
    if thread_running:
        progress_var.set("Modified files saved.")
    else:
        progress_var.set("Operation canceled by user.")

    save_button.config(state=tk.NORMAL)
    clear_button.config(state=tk.NORMAL)
    cancel_button.config(state=tk.DISABLED)

def clear_selection():
    """
    Clear all selections.
    """
    for var in number_vars.values():
        var.set(False)
```

```
125 def cancel_thread():
    """
127     Cancel the currently running thread.
    """
129     global thread_running
    thread_running = False
131
    def select_file():
133     """
    Open a file dialog to select input files and enable buttons based on available masses.
135     """
    file_paths = filedialog.askopenfilenames()
137     if file_paths:
        available_masses = range(75, 210)
139         file_path_label.config(text="")
        for file_path in file_paths:
141             try:
                with open(file_path, "r") as f:
143                     first_line = f.readline()
                    new_masses = [int(x) for x in first_line.split()[2:]]
145                     available_masses = [i for i in available_masses if i in new_masses]
            except Exception as e:
147                 messagebox.showerror(
                    "Error", f"An error occurred while reading the file:\n{e}"
149                 )
                disable_buttons()
151             else:
                enable_buttons(available_masses)
153                 path_var.set("\n".join(path for path in file_paths))
                file_path_label.config(text=path_var.get())
155     else:
        messagebox.showerror("Error", "No files selected.")
157
    def enable_buttons(available_masses):
159     """
    Enable the checkbuttons based on available masses.
161
    Parameters:
163     available_masses (list of int): The masses that are available in the selected file.
    """
165     for num in range(75, 210):
        if num in available_masses:
167             number_vars[num].set(False)
            number_buttons[num].state(["!disabled"])
169         else:
            number_vars[num].set(False)
171             number_buttons[num].state(["disabled"])

173 def disable_buttons():
    """
175     Disable all checkbuttons.
    """
177     for num in range(75, 210):
        number_vars[num].set(False)
179         number_buttons[num].state(["disabled"])

181 # Create main window
    root = tk.Tk()
183 root.title("Data Selector")

185 # Create a frame for the file selector
    file_frame = tk.Frame(root)
187 file_frame.pack(padx=10, pady=10, anchor="w")
```

```
# Create a Label to display the file path
path_var = tk.StringVar()
file_path_label = ttk.Label(
    file_frame,
    textvariable=path_var,
    width=50,
    anchor="w",
    relief="sunken",
    justify="left",
)
file_path_label.grid(row=0, column=1)

# Create file selector button
file_button = ttk.Button(file_frame, text="Select Files", command=select_file)
file_button.grid(row=0, column=0, padx=5)

# Create a frame to hold the Checkbuttons
frame = tk.Frame(root)
frame.pack(padx=10, pady=10)

# Create checkable buttons for numbers 75 to 209
number_vars = {}
number_buttons = {}
for num in range(75, 210):
    var = tk.BooleanVar()
    chk = ttk.Checkbutton(frame, text=str(num), variable=var, state="disabled")
    chk.grid(row=(num - 75) // 10, column=(num - 75) % 10, padx=5, pady=5, sticky="w")
    number_vars[num] = var
    number_buttons[num] = chk

# Create save, clear, and cancel buttons
button_frame = tk.Frame(root)
button_frame.pack(pady=10)

save_button = ttk.Button(button_frame, text="Save", command=save_selection)
save_button.pack(side=tk.LEFT, padx=5)

clear_button = ttk.Button(button_frame, text="Clear", command=clear_selection)
clear_button.pack(side=tk.LEFT, padx=5)

# Create a Cancel button for the thread
cancel_button = ttk.Button(
    button_frame, text="Cancel", command=cancel_thread, state="disabled"
)
cancel_button.pack(side=tk.LEFT, padx=5)

# Create a progress label
progress_var = tk.StringVar()
progress_label = ttk.Label(root, textvariable=progress_var)
progress_label.pack()

# Initialize thread flag
thread_running = False

# Start GUI event loop
root.mainloop()
```

B. Visor de los datos

data_viewer.py

```python
import tkinter as tk
from tkinter import ttk, filedialog, messagebox

import numpy as np
import matplotlib.pyplot as plt
import matplotlib.style as mplstyle


def generate_plot():
    """
    Generate a plot of the selected mass data from the file.

    This function reads the file specified in the path_var variable, extracts
    the data corresponding to the selected mass numbers, and plots it using
    matplotlib. If no file is selected or no mass numbers are selected, it shows
    an error message.
    """
    selected_numbers = [number for number, var in number_vars.items() if var.get()]
    if path_var.get() == "":
        messagebox.showerror("Error", "No file selected.")
        return

    if not selected_numbers:
        messagebox.showerror("Error", "No numbers selected.")
        return

    with open(path_var.get(), encoding="utf-8") as f:
        available = [int(m) for m in f.readline()[:-1].split("\t")[1:]]
        masses = dict(zip(available, list(range(1, len(available) + 1))))

    try:
        ar = np.loadtxt(
            path_var.get(),
            delimiter="\t",
            skiprows=1,
            usecols=[masses[m] for m in selected_numbers],
        )
    except OSError as e:
        messagebox.showerror("Error", f"An error occurred while reading the file:\n{e}")
        return

    plt.plot(range(1, ar.shape[0] + 1), ar, label=selected_numbers)
    plt.legend(loc=1)
    plt.xlabel("time (pushes)")
    plt.ylabel("intensity (arbitrary units)")

    # Show the plot in a regular pyplot window
    plt.show()


def clear_selection():
    """
    Clear all mass number selections.

    This function resets all mass number checkbuttons to an unchecked state.
    """
    for n in number_vars.values():
        n.set(False)

```

```
def select_file():
    """
    Open a file dialog to select a file and enable relevant checkbuttons.

    This function opens a file dialog for the user to select a file. It then reads
    the available mass numbers from the file and enables the corresponding checkbuttons.
    If an error occurs while reading the file, it shows an error message and disables
    all checkbuttons.
    """
    file_path = filedialog.askopenfilename()
    if file_path:
        available_masses = range(75, 210)
        file_path_label.config(text="")
        try:
            with open(file_path, "r", encoding="utf-8") as f:
                first_line = f.readline()
                new_masses = [int(x) for x in first_line.split()[2:]]
                available_masses = [i for i in available_masses if i in new_masses]
        except OSError as e:
            messagebox.showerror(
                "Error", f"An error occurred while reading the file:\n{e}"
            )
            disable_buttons()
        else:
            enable_buttons(available_masses)
            # Display each selected file path in the label
            path_var.set(file_path)
            file_path_label.config(text=path_var.get())
    else:
        messagebox.showerror("Error", "No file selected.")


def enable_buttons(available_masses):
    """
    Enable checkbuttons for the available mass numbers.

    This function enables the checkbuttons for the mass numbers that are available
    in the selected file and disables the others.

    Args:
        available_masses (list): List of available mass numbers.
    """
    for n in range(75, 210):
        if n in available_masses:
            number_vars[n].set(False)  # Uncheck by default
            number_buttons[n].state(["!disabled"])
        else:
            number_vars[n].set(False)
            number_buttons[n].state(["disabled"])


def disable_buttons():
    """
    Disable all mass number checkbuttons.

    This function disables all mass number checkbuttons and sets their state to unchecked.
    """
    for n in range(75, 210):
        number_vars[n].set(False)
        number_buttons[n].state(["disabled"])


def on_closing():
```

```
    """
    Handle the window closing event.

    This function quits the Tkinter main loop and destroys the root window when the
    user attempts to close the application window.
    """
    root.quit()
    root.destroy()


mplstyle.use("fast")

# Create main window
root = tk.Tk()
root.title("Data Selector")
root.protocol("WM_DELETE_WINDOW", on_closing)

# Create a frame for the file selector
file_frame = tk.Frame(root)
file_frame.pack(padx=10, pady=10, anchor="w")

# Create a label to display the file path
path_var = tk.StringVar()
file_path_label = ttk.Label(
    file_frame,
    textvariable=path_var,
    width=50,
    anchor="w",
    relief="sunken",
    justify="left",
)
file_path_label.grid(row=0, column=1)

# Create file selector button
file_button = ttk.Button(file_frame, text="Select Files", command=select_file)
file_button.grid(row=0, column=0, padx=5)

# Create mass selection checkbuttons for numbers 75 to 209
frame = tk.Frame(root)
frame.pack(padx=10, pady=10)

# Create checkable buttons
number_vars = {}
number_buttons = {}
for num in range(75, 210):
    var = tk.BooleanVar()
    chk = ttk.Checkbutton(frame, text=str(num), variable=var, state="disabled")
    chk.grid(row=(num - 75) // 10, column=(num - 75) % 10, padx=5, pady=5, sticky="w")
    number_vars[num] = var
    number_buttons[num] = chk

# Create view and clear buttons
button_frame = tk.Frame(root)
button_frame.pack(pady=10)
save_button = ttk.Button(button_frame, text="View", command=generate_plot)
save_button.pack(side=tk.LEFT, padx=5)
clear_button = ttk.Button(button_frame, text="Clear", command=clear_selection)
clear_button.pack(side=tk.LEFT, padx=5)

# Start GUI event loop
root.mainloop()
```

C. Calibración de los datos

data_calibrator.py

```python
import tkinter as tk
from tkinter import ttk, filedialog, messagebox

import numpy as np
from matplotlib import pyplot as plt


def m(x, w):
    """
    Calculate the weighted mean.

    Parameters:
    x (array-like): Values.
    w (array-like): Weights.

    Returns:
    float: Weighted mean of x.
    """
    return np.sum(x * w) / np.sum(w)


def cov(x, y, w):
    """
    Calculate the weighted covariance.

    Parameters:
    x (array-like): Values of the first variable.
    y (array-like): Values of the second variable.
    w (array-like): Weights.

    Returns:
    float: Weighted covariance of x and y.
    """
    return np.sum(w * (x - m(x, w)) * (y - m(y, w))) / np.sum(w)


def corr(x, y, w):
    """
    Calculate the weighted correlation.

    Parameters:
    x (array-like): Values of the first variable.
    y (array-like): Values of the second variable.
    w (array-like): Weights.

    Returns:
    float: Weighted correlation of x and y.
    """
    return cov(x, y, w) / np.sqrt(cov(x, x, w) * cov(y, y, w))


def wls(x, y, w):
    """
    Perform weighted least squares regression.

    Parameters:
    x (array-like): Independent variable values.
    y (array-like): Dependent variable values.
    w (array-like): Weights.
```

```
    Returns:
    tuple: (slope, intercept, r_squared) of the regression line.
    """
    c = corr(x, y, w)
    slope = c * np.std(y) / np.std(x)
    intercept = m(y, w) - m(x, w) * slope
    return slope, intercept, c**2


class CalibrationApp:
    """Application for performing calibration using weighted least squares regression."""

    def __init__(self, root):
        """
        Initialize the CalibrationApp.

        Parameters:
        root (tk.Tk): Root window of the Tkinter application.
        """
        self.root = root
        self.root.title("Calibration")

        self.rows = []
        self.create_widgets()

    def create_widgets(self):
        """Create and layout the widgets for the application."""
        # Container frame for rows
        self.frame = ttk.Frame(self.root)
        self.frame.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

        # Create initial rows
        for _ in range(3):
            self.add_row()

        # Bottom row with "Add Row" and "Calibrate" buttons
        self.bottom_frame = ttk.Frame(self.root)
        self.bottom_frame.pack(padx=10, pady=10, fill=tk.X)

        self.add_button = ttk.Button(
            self.bottom_frame, text="Add Row", command=self.add_row
        )
        self.add_button.pack(side=tk.LEFT, padx=5)

        self.calibrate_button = ttk.Button(
            self.bottom_frame, text="Calibrate", command=self.calibrate
        )
        self.calibrate_button.pack(side=tk.LEFT, padx=5)

        self.mass_label = ttk.Label(self.bottom_frame)
        self.mass_label.configure(text="Mass:")
        self.mass_label.pack(side=tk.LEFT, padx=5)

        validate_mass = (self.frame.register(str.isdigit), "%P")
        self.mass_entry = ttk.Entry(
            self.bottom_frame, width=4, validate="key", validatecommand=validate_mass
        )
        self.mass_entry.insert(0, "")
        self.mass_entry.pack(side=tk.LEFT, padx=5)

    def add_row(self):
        """Add a new row to the application for selecting files and entering concentrations."""
        row_frame = ttk.Frame(self.frame)
```

```
        row_frame.pack(fill=tk.X, pady=2)

        delete_button = ttk.Button(
            row_frame, text="Delete", command=lambda rf=row_frame: self.delete_row(rf)
        )
        delete_button.pack(side=tk.LEFT, padx=5)

        file_button = ttk.Button(
            row_frame, text="Select File", command=lambda: self.select_file(row_frame)
        )
        file_button.pack(side=tk.LEFT, padx=5)

        file_label = ttk.Entry(row_frame, state="readonly", width=30)
        file_label.pack(side=tk.LEFT, padx=5)

        validate_concentration = (
            self.frame.register(self.validate_calibration_number),
            "%P",
        )

        concentration_entry = ttk.Entry(
            row_frame, width=10, validate="key", validatecommand=validate_concentration
        )
        concentration_entry.insert(0, "0.0e00")
        concentration_entry.pack(side=tk.LEFT, padx=5)

        mass_label = ttk.Label(row_frame)
        mass_label.configure(text="ppb")
        mass_label.pack(side=tk.LEFT)

        self.rows.append((row_frame, file_label, concentration_entry))

    def delete_row(self, row_frame):
        """
        Delete a specified row from the application.

        Parameters:
        row_frame (ttk.Frame): Frame of the row to be deleted.
        """
        for widget in row_frame.winfo_children():
            widget.destroy()
        row_frame.destroy()
        self.rows = [row for row in self.rows if row[0] != row_frame]

    def select_file(self, row_frame):
        """
        Open a file dialog to select a file and update the corresponding row's file label.

        Parameters:
        row_frame (ttk.Frame): Frame of the row where the file was selected.
        """
        file_path = filedialog.askopenfilename()
        if file_path:
            for row in self.rows:
                if row[0] == row_frame:
                    row[1].config(state=tk.NORMAL)
                    row[1].delete(0, tk.END)
                    row[1].insert(0, file_path)
                    row[1].config(state="readonly")

    def calibrate(self):
        """Perform calibration using the selected files and entered concentrations."""
        x = []
        y = []
```

```
        s = []
        if not (self.mass_entry.get()):
            messagebox.showerror("Error", "No mass selected for the calibration.")
            return
        mass = int(self.mass_entry.get())
        if len(self.rows) < 3:
            messagebox.showerror(
                "Error", "At least three points are needed for the calibration."
            )
            return
        for _, file_label, concentration_entry in self.rows:
            try:
                file_path = file_label.get()
                with open(file_path, "r", encoding="utf-8") as f:
                    first_line = f.readline()
                    new_masses = [int(x) for x in first_line.split()[2:]]
                    if mass not in new_masses:
                        messagebox.showerror(
                            "Error",
                            f"Mass {mass} not in available masses {new_masses} in file {file_path}.",
                        )
                        return
                    else:
                        signal = np.loadtxt(
                            file_path,
                            delimiter="\t",
                            skiprows=1,
                            usecols=[new_masses.index(mass) + 1],
                        )
            except OSError as e:
                messagebox.showerror(
                    "Error", f"An error occurred while reading the file:\n{e}"
                )
                return
            x.append(float(concentration_entry.get()))
            y.append(np.mean(signal))
            s.append(np.std(signal))

        x = np.array(x)
        y = np.array(y)
        s = np.array(s)
        w = 1 / s**2

        slope, intercept, r2 = wls(x, y, w)
        xp = [x.min(), x.max()]
        pred = np.array([intercept + x.min() * slope, intercept + x.max() * slope])

        plt.figure()

        # Plot the results
        plt.errorbar(
            x,
            y,
            yerr=s,
            linestyle="none",
            marker=".",
            color="b",
            capsize=5,
            label="calibration points \pm SD",
        )
        plt.plot(
            xp,
            pred,
            "r—",
```

```
            label=f"regression: I = {slope:.4f} * [C] + {intercept:.4f}   (r^2 = {r2:.3f})",
        )
        plt.grid(linestyle=":")
        plt.ylabel("intensity (arbitrary units)")
        plt.xlabel("concentration (ppb)")
        plt.legend()
        plt.show()

    def validate_calibration_number(self, mass):
        """
        Validate if the input string can be converted to a float.

        Parameters:
        mass (str): Input string to validate.

        Returns:
        bool: True if valid float, False otherwise.
        """
        try:
            float(mass)
            return True
        except ValueError:
            self.frame.bell()
            return False


if __name__ == "__main__":
    root = tk.Tk()
    app = CalibrationApp(root)
    root.mainloop()
```

D. Detector de eventos

Versión *script* de data_detector.ipynb

```python
# ### IMPORTS

import sys

import numpy as np
import scipy.signal
from matplotlib import pyplot as plt

# ### CONFIGURATION

DATADIR = ...  # set to the directory containing data
DATASET = ...  # set to the filename of the data
LOAD_MASSES = (142, 175, 193, 195)
MIN_EVENT_LENGTH = 10
MAX_EVENT_LENGTH = 150
MASS_FOR_SELECTION = 193
MASS_OF_INTEREST = 175

# ### MAIN

# #### Generate numpy array

with open(DATADIR / DATASET) as f:
    available = [int(m) for m in f.readline()[:-1].split("\t")[1:]]
    masses = dict(zip(available, list(range(1, len(available) + 1))))

for mass in LOAD_MASSES:
    assert (
        mass in masses
    ), f"Mass {mass} not in the available masses:{', '.join(m for m in masses)}"

ar = np.loadtxt(
    DATADIR / DATASET,
    delimiter="\t",
    skiprows=1,
    usecols=[masses[m] for m in LOAD_MASSES],
)

print(
    f"Loadad array with {ar.shape[1]} columns and {ar.shape[0]} rows, with a size of {sys.getsizeof(ar)
    } bytes."
)

mass_dict = dict(zip(LOAD_MASSES, range(len(LOAD_MASSES))))

# #### Get all peaks

events_by_mass = {}
for i in range(ar.shape[1]):
    peaks = scipy.signal.find_peaks(
        ar[:, i], width=(MIN_EVENT_LENGTH, MAX_EVENT_LENGTH)
    )[0]
    if not peaks.size:
        print("Could,'t detect any event for mass", LOAD_MASSES[i])
    peak_widths = scipy.signal.peak_widths(ar[:, i], peaks, rel_height=1.0)
    left = peak_widths[2]
    right = peak_widths[3]
    events_by_mass[LOAD_MASSES[i]] = list(zip(left.astype(int), right.astype(int)))

```

```
# #### Get the integral of every event
sum_I = np.array(
    [
        i
        for i in [
            np.sum(ar[side[0] : side[1], mass_dict[MASS_OF_INTEREST]], axis=0)
            for side in events_by_mass[MASS_FOR_SELECTION]
        ]
    ]
)

#

DT = 13e-6  # 13 us == 13e-6 s
FLOW = 0.5  # 30 uL/min == 0.5 uL/s
TE = 0.1  # 0.1 = 10 %
# 1 ppb == 1 ng/mL == 1 fg/uL

X0 = ...  # insert value from calibrate
X1 = ...  # insert value from calibrate
event_mass_interest = (sum_I - X0) / X1 * DT * FLOW / TE

# ### VALIDATION

icp = ...  # load the icp data as an array-like

plt.figure()
plt.boxplot((icp, event_mass_interest[event_mass_interest < 120]))
plt.gca().set_xticklabels(["SC-ICP-MS", "CyTOF"])
plt.gca().set_ylabel("mass (fg)")
plt.show()
```