# UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA



Profesor Patrocinante:

**Miguel E. Figueroa T.**


Informe de Memoria de Título
para optar al título de:

**Ingeniero Civil Electrónico**

# Implementación de algoritmo para el análisis de datos en instrumentación DOAS


Concepción, Abril de 2010

Tomás Ángel Glaría López

UNIVERSIDAD DE CONCEPCIÓN
Facultad de Ingeniería
Departamento de Ingeniería Eléctrica

Profesor Patrocinante:
Miguel E. Figueroa T.

# Implementación de algoritmo para el análisis de datos en instrumentación DOAS

Tomás Ángel Glaría López

Informe de Memoria de Título
para optar al Título de

Ingeniero Civil Electrónico

Abril de 2010

# Resumen

La creciente degradación de la calidad del aire ha llevado a normar las concentraciones de contaminantes atmosféricos que pueden ser causales de enfermedades en las personas. De los diversos métodos existentes para la medición de partículas en el aire, el desarrollo de instrumentación de Espectroscopía de Absorción Óptica Diferencial (Differential Optical Absorption Spectroscopy: DOAS) ha cobrado gran vigencia siendo un método no invasivo de medición y permitiendo analizar concentraciones de diversos contaminantes en tiempo real.

Esta Memoria de Título presenta un software para el control de instrumentación DOAS de tipo activo que implementa un algoritmo para el análisis de espectros obtenidos con esta instrumentación. Las capacidades de la aplicación incluyen una interfaz gráfica, almacenamiento y carga de espectros medidos e historiales de concentraciones diarias, y la opción de automatizar la captura, almacenamiento y análisis de espectro. La aplicación además facilita al usuario la obtención y análisis de datos, permite la configuración y utilización de la instrumentación DOAS, y despliega y almacena datos históricos de concentración.

A quienes estuvieron ahí sin siquiera saberlo

# Agradecimientos

A aquellos que me apoyaron, animaron y/o me guiaron a lo largo del desarrollo de este documento.

# Tabla de Contenidos

# LISTA DE TABLAS

# Lista de Figuras

# Nomenclatura

| | |
|---|---|
| $\lambda$ | : longitud de onda. |
| $\sigma(\lambda)$ | : sección eficaz de absorción |
| $\sigma_{j0}(\lambda)$ | : componente de $\sigma(\lambda)$ que varía lentamente al variar $\lambda$ |
| $\sigma'_0(\lambda)$ | : componente de $\sigma(\lambda)$ que varía rápidamente al variar $\lambda$ |
| $\varepsilon_R$ | : coeficiente de extinción debido a la dispersión de Rayleigh |
| $\varepsilon_M$ | : coeficiente de extinción debido a la dispersión de Mie |
| $c$ | : concentración del contaminante |
| L | : distancia recorrida por el haz de luz en instrumentación DOAS |
| $A(\lambda)$ | : capacidad de transmisión de instrumentos ópticos y fenómenos de turbulencia |
| $I(\lambda)$ | : Intensidad de luz medida por la instrumentación DOAS |
| $I_0$ | : Intensidad de luz de la fuente |
| $I'_0(\lambda)$ | : Intensidad de la componente de $I(\lambda)$ que varía lentamente al variar $\lambda$ |
| D' | : densidad óptica espectral |

# Abreviaciones

### *Mayúsculas*

| | |
|---|---|
| DSP | : Digital Signal Processor |
| DOAS | : Differential Optical Absorption Spectroscopy |
| FPGA | : Field Programmable Gate Array |
| HMI | : Human Machine Interface |
| OMS | : Organización Mundial de la Salud |
| EPA | : Agencia de Protección Ambiental |
| U.A. | : Unidad Arbitraria |

# 1 Introducción

## 1.1 Introducción General

El desarrollo tecnológico del hombre ha degradado progresivamente la calidad del medio ambiente. Esto se ha convertido en una preocupación constante para el ser humano puesto que atenta contra la calidad de vida.

Este hecho ha llevado a normar las cantidades presentes en la atmósfera de ciertas partículas que pueden ser perjudiciales para la salud y que son generados por diferentes fuentes. A modo de ejemplo, se puede considerar el caso del dióxido de nitrógeno ($NO_2$) para el cual la Organización Mundial de la Salud (OMS) reporta que la exposición a este contaminante puede producir efectos agudos y crónicos en la salud de las personas, especialmente en aquellos más sensibles como los asmáticos. Similarmente, la Agencia de Protección Ambiental de los Estados Unidos (EPA) menciona también que este contaminante puede producir irritación pulmonar y disminuir la resistencia a las enfermedades respiratorias. En Chile, la norma primaria de calidad del aire para dióxido de nitrógeno en aire establece como concentración anual máxima 100 [$\mu g/m^3N$] y como concentración máxima en una hora 400 [$\mu g/m^3N$]. Otros contaminantes atmosféricos de interés en Chile son el ozono, cuya norma primaria de calidad del aire es de 120 [$\mu g/m^3N$] como concentración de ocho horas; el monóxido de carbono, cuya norma primaria de calidad del aire es de 10 [$\mu g/m^3N$] como concentración de ocho horas y de 30 [$\mu g/m^3N$] como concentración de una hora; el dióxido de azufre, cuya norma primaria de calidad del aire es de 80 [$\mu g/m^3N$] como concentración anual y de 250 [$\mu g/m^3N$] como concentración de veinticuatro horas.

Las concentraciones de estos contaminantes pueden ser medidas con instrumentación basada en la quimioluminiscencia, que es emisión de luz producto de una reacción química. Este tipo de instrumentación, que genéricamente consiste en una cámara de reacción y un tubo multiplicador, utiliza mediciones cuantitativas de la emisión óptica de la especie química excitada para determinar su concentración. Para la medición del $NO_2$, por ejemplo, se utiliza la reacción química que combina óxido nítrico, NO, con ozono, $O_3$, cuyo producto es oxígeno, $O_2$, y dióxido de nitrógeno en estado excitado [12]. En el paso de un estado de mayor energía a uno de menor energía, el $NO_2$ libera una cantidad luz proporcional a la cantidad de óxido de nitrógeno al inicio de la reacción.

$$NO + O_3 \quad \rightarrow \quad NO_2 + O_2 + luz$$

Para poder medir la cantidad de $NO_2$ presente, primero debe pasarse por un conversor para tener una

cantidad de NO proporcional a la de $NO_2$ presente inicialmente, para luego utilizar la reacción química y relacionar la cantidad de luz emitida con la de NO y, en consecuencia, con la de $NO_2$. Si en la muestra inicial, existe NO en conjunto con $NO_2$, es necesario restar sustraer la concentración del primero al valor de concentración obtenido por la reacción para obtener la cantidad de $NO_2$ de la muestra inicial. Este tipo de instrumentación tiene como desventaja el requerir una muestra para poder realizar el análisis y así obtener la concentración del contaminante, además, es específico para un tipo de contaminante, se requiere una reacción química distinta el análisis de diferentes partículas por lo que se requiere tener almacenados los reactivos necesarios para cada tipo de reacción química.

Otro método de medición de contaminantes consiste en la toma de una muestra para su posterior análisis químico. Éste tiene como desventajas que el proceso de análisis puede tomar demasiado tiempo como para detectar y corregir algún alza del nivel de contaminantes permitido y que se requiere una muestra para cada tipo de contaminante a analizar.

Un tercer método de análisis, es la utilización de fotometría para la medición de concentración de contaminantes. En este tipo de instrumentación, se tiene una recámara con una fuente de luz en un lado y un fotómetro del otro por el cual se fluye una muestra del gas a analizar. El fotómetro mide a intensidad de luz generada en ausencia del gas de muestra para así obtener una intensidad de referencia, y mide intensidad de luz recibida en presencia del gas de muestra. Luego, basándose en la ley de Lambert-Beer se calcula la concentración del contaminante en la muestra tomada. Este tipo de instrumento debe medir constantemente la fuente de referencia para obtener los valores de concentración y debe tomar una muestra del gas a analizar.

Una alternativa que ha cobrado gran vigencia es el uso de instrumentación DOAS (Differential Optical Absorption Spectroscopy) que permite la obtención de la concentración de diferentes contaminantes en tiempo real. Esta instrumentación es un método no invasivo para la medición de contaminantes, desde una fuente se envía un haz de luz, el cual atraviesa una columna abierta de aire, donde las débiles concentraciones entre sus constituyentes gaseosos forman trazas de gases que pueden absorber parte de esta luz. Un receptor mide la cantidad de luz recibida y envía esta información a un computador donde se cuantifica la absorción. En función de la luz absorbida en los diferentes rangos del espectro, se calcula la concentración de los diferentes contaminantes en esa ventana de absorción.

Este tipo de instrumentación tiene como ventajas el ser capaz de entregar valores de concentración de diferentes compuestos en tiempo real, permite la obtención de concentración de

diferentes partículas a partir de una misma medición y es un método no invasivo, no requiere tomar una muestra del gas a analizar. Además, tiene la ventaja de no requerir, en ciertas condiciones, calibración.

## 1.2 Trabajos Previos

La base teórica tras un instrumento DOAS se encuentra documentado en el libro *"Differential Optical Absorption Spectroscopy"* de Ulrich Platt y Jochen Stutz. [1]. En él se mencionan diferentes métodos para el ajuste lineal (ajuste por mínimos cuadrados lineal) y no lineal (minimización por gradiente, Gauss-Newton, Lavenberg-Marquardt) de los datos a la función que relaciona la concentración de los contaminantes con la intensidad de luz sensada. Para el análisis, se realiza una iteración de ajustes lineales y no lineales de modo de ajustar los datos a los parámetros de la función que describe el proceso, y a la curva de referencia.

Existen numerosos trabajos que muestran diversas aplicaciones con instrumentación DOAS. En un suburbio de Nanjing, China, se utilizó instrumentación DOAS activo bi-estático para la medición de contaminantes atmosféricos. Dos telescopios newtonianos son utilizados en esta configuración, uno como emisor, que enfoca la luz proveniente de una lámpara de xenón, y otro como receptor, que recibe la luz y la envía a un espectrómetro por medio de una fibra óptica [5]. La instrumentación DOAS es capaz de utilizar instrumentación existente para el análisis de contaminantes, como es una lámpara de precaución para el tráfico aéreo, lo cual permite realizar mediciones DOAS sin la necesidad de preocuparse por la instalación de fuentes de luz [6]. En la antártica se utilizaron dos estaciones de medición basadas en instrumentación DOAS pasiva para comprobar la concentración de ozono estratosférico en el período posterior al invierno [4].

Si bien la lámpara de xenón es de común utilización en la instrumentación DOAS, tiene como desventaja su alto consumo eléctrico y baja vida útil. A raíz de esto se ha estudiado la utilización de LED para la fuente de luz. Éstos tienen como ventaja una mayor vida útil y un menor consumo energético, sin embargo poseen una banda espectral bastante angosta por lo que se debe escoger una fuente de luz LED correspondiente al rango de operación en el cual se desea medir y utilizar más de una lámpara de este tipo para medir contaminantes en diferentes rangos del espectro [2]. En efecto, una luz LED azul que opera en el rango alrededor de los 465[nm] es utilizada para mediciones en tiempo real de dióxido de nitrógeno [3].

## 1.3 Hipótesis de Trabajo

El cálculo de las concentraciones se realiza mediante una comparación entre los datos medidos en terreno y un patrón medido en laboratorio. Es posible de realizar el análisis de las mediciones de espectro mediante ajuste polinomial de primer orden, para la parte lineal, y sistemas de ajustes no lineales para el ajustes de curvas medidas con la curva patrón, con esta comparación y los valores de sección eficaz de absorción de los contaminantes, es posible calcular la concentración de éstos en la atmósfera.

## 1.4 Objetivos

### 1.4.1 Objetivo General

Implementar un algoritmo para el análisis de las mediciones en la instrumentación DOAS.

### 1.4.2 Objetivos Específicos

- Implementación del algoritmo en un sistema embebido autónomo.
- Utilización de este sistema para mediciones en terreno.
- Análisis de resultados.

## 1.5 Alcances y Limitaciones

Se implementará el software para el análisis de mediciones hechas por el sistema óptico DOAS. Éste desplegará las concentraciones de los diferentes contaminantes a medir a en pantalla y los almacenará en un archivo. La obtención de los datos y la calibración de la instrumentación están fuera de los alcances de este trabajo.

# 2 Medición DOAS mediante la ley de Lambert-Beer

## 2.1 Introducción

La instrumentación DOAS es un método no invasivo para la medición de contaminantes atmosféricos. Éste tiene tres partes principales, el emisor, el receptor y el equipo de análisis. El emisor envía un haz de luz al receptor, que atraviesa diferentes moléculas. Diferentes moléculas absorben luz en diferentes rangos del espectro a lo largo del camino recorrido por la luz. El receptor recibe el haz de luz y envía los datos al equipo de análisis que mide la intensidad de luz en diferentes rangos del espectro y relaciona esta intensidad con un valor de concentración para cada contaminante.

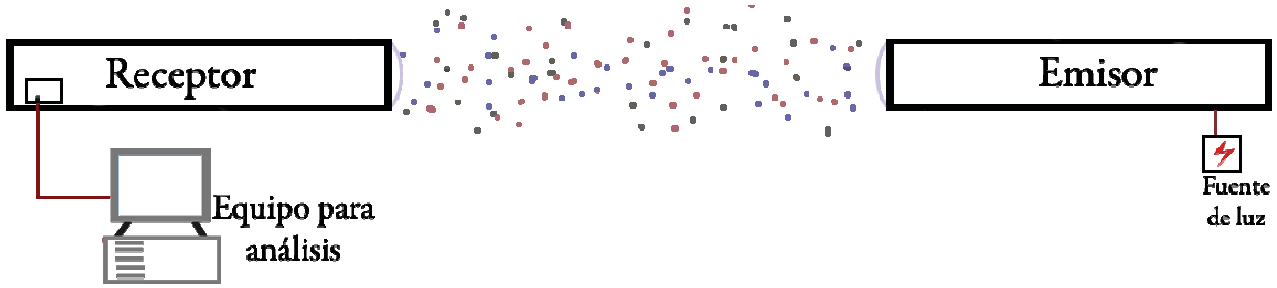


*Figura 2.1: Esquema de un instrumento DOAS*

El cálculo de la concentración de cada contaminante se basa en la ley de Lambert-Beer que relaciona la intensidad de luz que es absorbida por una molécula con la concentración de ella en el espacio.

## 2.2 Ley de Lambert-Beer

La ley de Lambert-Beer relaciona la cantidad de luz absorbida por una molécula con la longitud de camino óptico recorrida por la luz y con las propiedades ópticas de las moléculas absorbentes. En la atmósfera, las diferentes partículas presentes en el aire son las que absorben parte de la luz, y se rigen por la fórmula 2.1:

$$I(\lambda) = I_0 \cdot e^{-L\cdot(\sum(\sigma_j(\lambda)\cdot c_j)+\varepsilon_R+\varepsilon_M)} \cdot A(\lambda) \qquad (2.1)$$

Donde, L es la distancia recorrida por el haz de luz, $c_j$ es la concentración de la partícula, $\sigma(\lambda)$ es la sección eficaz de absorción, que corresponde a la capacidad de una molécula de absorber un fotón de una determinada longitud de onda; $\varepsilon_R$ corresponde al coeficiente de extinción debido a la dispersión de Rayleigh; $\varepsilon_M$, al coeficiente de extinción debido a la dispersión de Mie y $A(\lambda)$

representa la capacidad de transmisión de los instrumentos ópticos y los fenómenos de turbulencia.

## 2.3 Densidad óptica diferencial

La sección eficaz de absorción se puede separar en una parte que varía rápidamente en función de la longitud de onda de la luz, y otra que varía lentamente en función de ésta.

$$\sigma_j(\lambda) = \sigma_{j0}(\lambda) + \sigma'_j(\lambda) \tag{2.2}$$

Siendo $\sigma_{j0}(\lambda)$ la parte que varía lentamente, en función de $\lambda$, y $\sigma'_j(\lambda)$, la parte que varía rápidamente.

Considerando los fenómenos de dispersión de Rayleigh y de Mie, y aquellos determinados por $A(\lambda)$, dentro de un pequeño rango de valores de $\lambda$, la ecuación 2.1 se puede escribir como:

$$I(\lambda) = I'_0(\lambda) \cdot e^{-L \cdot \sum (\sigma'_j(\lambda) \cdot c_j)} \tag{2.3}$$

La cual sólo se encuentra expresada en función de la parte que varía rápidamente con respecto a la longitud de onda, $\lambda$.

$I(\lambda)$, es medida por el instrumento y mediante un ajuste polinomial, se puede calcular $I'_0(\lambda)$. Luego, despejando la sumatoria de la ecuación:

$$D' = \ln\left(\frac{I'_0(\lambda)}{I(\lambda)}\right) = L \cdot \sum \sigma_j \cdot c_j \tag{2.4}$$

Donde D' se conoce como densidad óptica diferencial.

## 2.4 Ajuste por mínimos cuadrados

En la ecuación 2.4, el largo de la sección atravesada por la luz y los valores de sección eficaz de absorción, $\sigma_j$, son conocidos para cada una de las partículas a medir. Por lo tanto sólo resta calcular los valores de concentración de las moléculas, lo que, además, se puede simplificar considerablemente, pues las zonas de absorción de los diferentes contaminantes atmosféricos se producen principalmente en diferentes rangos de longitud de onda, aunque existen zonas comunes de absorción.

En efecto, la figura 2.4.1, muestra los diferentes rangos de absorción para diferentes partículas en la atmósfera.

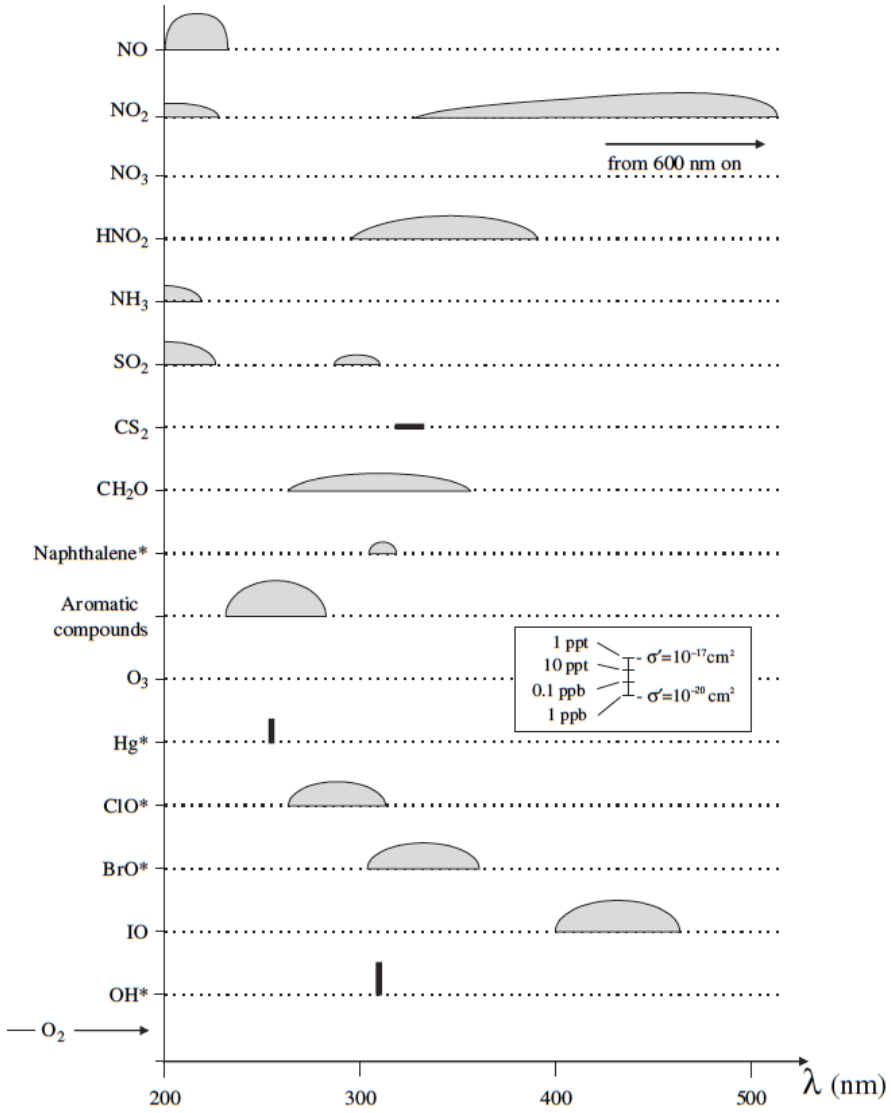


*Figura 2.4.1: Rangos de absorción de diferentes partículas atmosféricas [5]*

Para obtener el valor de concentración de partículas, se debe ajustar por mínimos cuadrados la densidad óptica diferencial a la curva de la sección eficaz de absorción de la partícula a analizar dentro del rango de absorción correspondiente, lo cual es posible dado que las secciones eficaces de absorción, $\sigma_j$, de las diferentes partículas son linealmente independientes. Secciones eficaces de absorción para diferentes compuestos, se pueden observar en la figura 2.4.2.

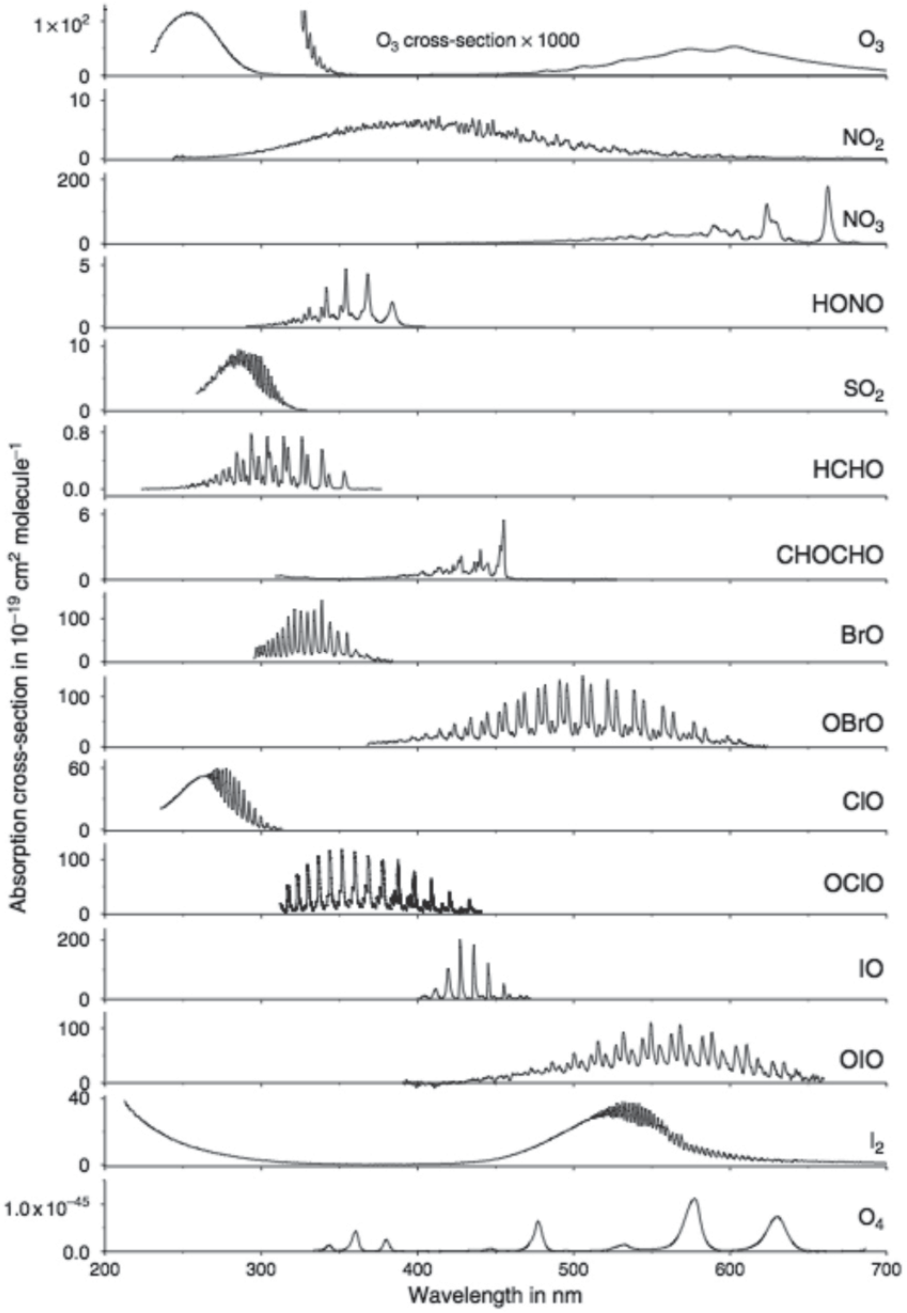


*Figura 2.4.2: Sección eficaz de absorción para diferentes compuestos en función de la longitud de onda [5]*

El ajuste polinomial se realiza mediante ajuste de mínimos cuadrados. Se busca la ecuación que describe la parte lenta del proceso de absorción descrito por la ecuación (2.1). Separando ambas la parte de rápida variación con respecto a λ y la de lenta variación:

$$I(\lambda) = I_0 \cdot e^{-L\cdot(\sum(\sigma_j 0(\lambda)\cdot c_j)+\varepsilon_R+\varepsilon_M)} \cdot A(\lambda) \cdot e^{-L\cdot(\sum(\sigma'_j(\lambda)\cdot c_j)} \tag{2.5}$$

La parte de variación lenta en función de λ, en un pequeño rango de λ, es la que se aproxima por ajuste polinomial.

Linealizando la ecuación (2.5):

$$\ln(I(\lambda)) = \ln(I_0 \cdot e^{-L\cdot(\sum(\sigma_j(\lambda)\cdot c_j)+\varepsilon_R+\varepsilon_M)} \cdot A(\lambda)) \tag{2.6}$$

$$\ln(I(\lambda)) = -L\cdot(\sum(\sigma_j(\lambda)\cdot c_j)+\varepsilon_R+\varepsilon_M)+\ln(I_0(\lambda)) \tag{2.7}$$

La ecuación (2.7) se desea ajustar a:

$$y = mx + b \tag{2.8}$$

Luego, esto corresponde a resolver por cuadrados mínimos el conjunto de ecuaciones

$$[\lambda_i \; 1]\cdot\begin{bmatrix} m \\ b \end{bmatrix} = [\log(I(\lambda_i))] \tag{2.9}$$

La solución de este sistema, entrega los valores de *m* y *b* de modo que se tiene el valor de $I'_0$ de la ecuación (2.3) dado por:

$$I'_0(\lambda) = e^{m\lambda+b} \tag{2.10}$$

Reemplazando este valor en la ecuación (2.4) se obtiene el valor de la densidad óptica diferencial. Ahora para obtener el valor de la concentración, se debe realizar otro ajuste de mínimos cuadrados entre la densidad óptica diferencial y la sección eficaz de absorción del contaminante.

$$[D \; 1]\cdot\begin{bmatrix} k_1 \\ k_2 \end{bmatrix} = \sigma(\lambda) \tag{2.11}$$

Debido a posibles diferencias de alineamiento de la longitud de onda entre la sección eficaz de referencia y el espectro medido, es necesario realizar un alineamiento de longitud de onda. Este proceso consiste en buscar el desplazamiento que minimiza la diferencia entre el espectro medido y el de referencia tras el cual se obtiene un factor de desplazamiento. Luego se repiten el cálculo de la densidad óptica diferencial (ecuación 2.11) considerando este corrimiento en el espectro de referencia y el alineamiento hasta que se estabilicen los valores.

Finalmente, se puede calcular el valor de la concentración del contaminante buscado como

$$c = \frac{1}{L\cdot k_1} \tag{2.12}$$

# 3 Instrumentación DOAS

## 3.1 Descripción de Hardware

En este trabajo se utilizó instrumentación DOAS del tipo activo, bi-estático, es decir, usando una fuente de luz artificial como emisor.

Para el emisor se utiliza una lámpara de arco de Xenón, de uso automotriz, de 35[W] como fuente de luz. Ésta va acoplada a un telescopio reflector tipo Newtoniano, con un espejo principal de 203[mm] de diámetro, una distancia focal de 1000[mm] y una razón focal F/D de 4,9.

El receptor está compuesto por un telescopio reflector tipo Newtoniano, con un espejo principal de 150[mm] de diámetro, una distancia focal de 450[mm] y una razón focal F/D de 3. La luz es llevada desde el telescopio a un espectrómetro mediante una fibra óptica multimodo de 400[µm] de diámetro y 2[m] de largo. El espectrómetro usado es un Ocean Optics HR4000 sensibilizado para el rango espectral de 197,56[nm] a 659,58[nm].

El equipo de análisis corresponde a un computador portátil, marca Acer modelo Aspire 3620, corriendo Windows XP SP2 y conectado directamente con el espectrómetro por conexión usb.

En la figura 3.1 se muestra un esquema de la configuración a utilizar.

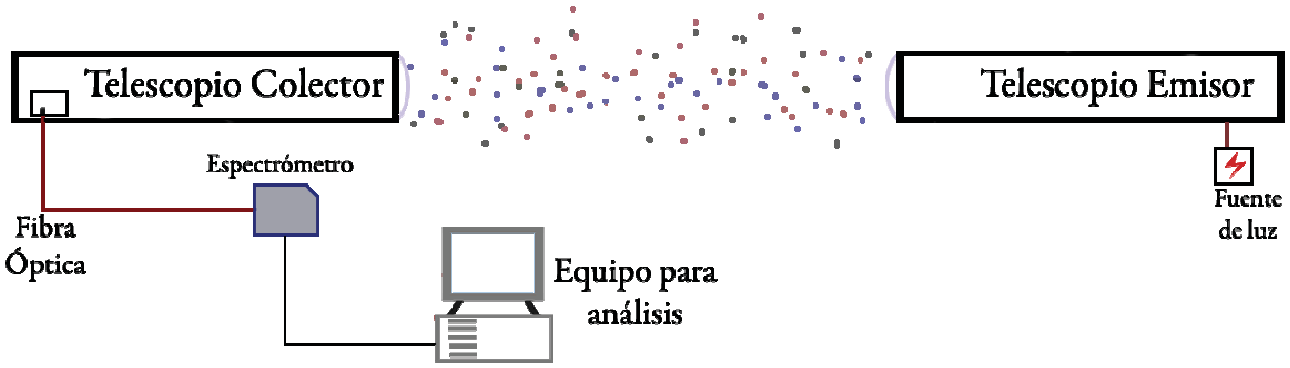


*Figura 3.1: Esquema de Instrumentación DOAS Activo bi-estático implementado*

| | | |
|---|---|---|
| **Fuente de Luz** | Lámpara de Xenón de 35[W], uso automotriz | |
| **Telescopio Emisor** | Telescopio tipo Newtoniano | |
| | Diámetro de Espejo principal | 203[mm] |
| | Distancia Focal | 1.000[mm] |
| | Razón Focal (F/D) | 4,9 |
| **Telescopio Colector** | Telescopio tipo Newtoniano | |
| | Diámetro de Espejo principal | 150[mm] |
| | Distancia Focal | 450[mm] |
| | Razón Focal (F/D) | 3 |
| **Fibra Óptica** | Fibra Multimodo | |
| | Diámetro | 400[µm] |
| | Apertura Numérica | 0,22 |
| | Largo | 2 [m] |
| | Material | Silica Fusionada |
| | Transmisión | 200-1100 [nm] |
| | Recubrimiento | Acero Inoxidable |
| **Espectrómetro** | Marca | Ocean Optics |
| | Modelo | HR4000 |
| | Rango espectral | 197,56 [nm] – 659,58 [nm] |
| | Diseño | Symmetrical Czerny-Turner |
| | Distancia Focal | F=101,6[mm] |
| | Grating | 600 [groves/mm] @ 400[µm] |
| | Resolución | 0,14 [nm/píxel] |
| | Número de Pixels | 3648 |
| | Tipo de Detector | Toshiba TCD1304AP linear CCD Array |
| | Tamaño de Pixel | 8 [µm] x 200 [µm] |
| **Equipo de Análisis** | Computador Portátil | |
| | Marca | Acer |
| | Modelo | Aspire 2630 |
| | Procesador | Intel Celaron 1,5[GHz] |
| | Memoria RAM | 1,5 GB DDR2 |
| | Disco duro | 500 [GB] |
| | Sistema Operativo | Windows XP SP2 Professional |

*Tabla 3.1: Descripción de los componentes de la instrumentación DOAS*

## 3.2 Software implementado

El software implementado trabajo actualmente con mediciones ya almacenadas o con mediciones realizadas con el espectrómetro, posee un menú desplegable para seleccionar el contaminante a analizar y para abrir el espectro con el cual realizar el análisis.

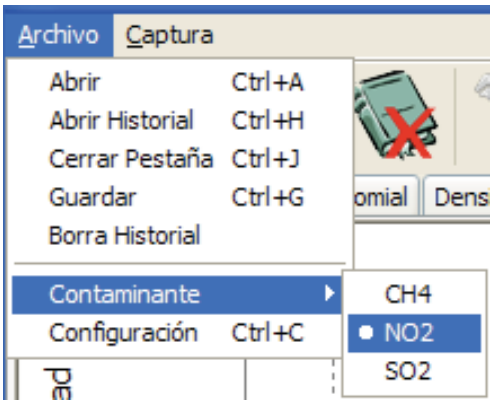


*Figura 3.2.1: Menú de selección de contaminante*

La selección del contaminante, configura automáticamente el rango de análisis y carga el valor de la sección eficaz de absorción desde un archivo que contiene la sección eficaz del contaminante correspondiente.

En la configuración se puede cambiar la unidad de despliegue de la concentración, el valor máximo de concentración antes que se active la alarma y la distancia a la que se efectuó la medición. Las unidades de despliegue disponibles a escoger son moléculas/cm$^2$, p.p.m., p.p.b., p.p.t. y µg/m$^3$N.

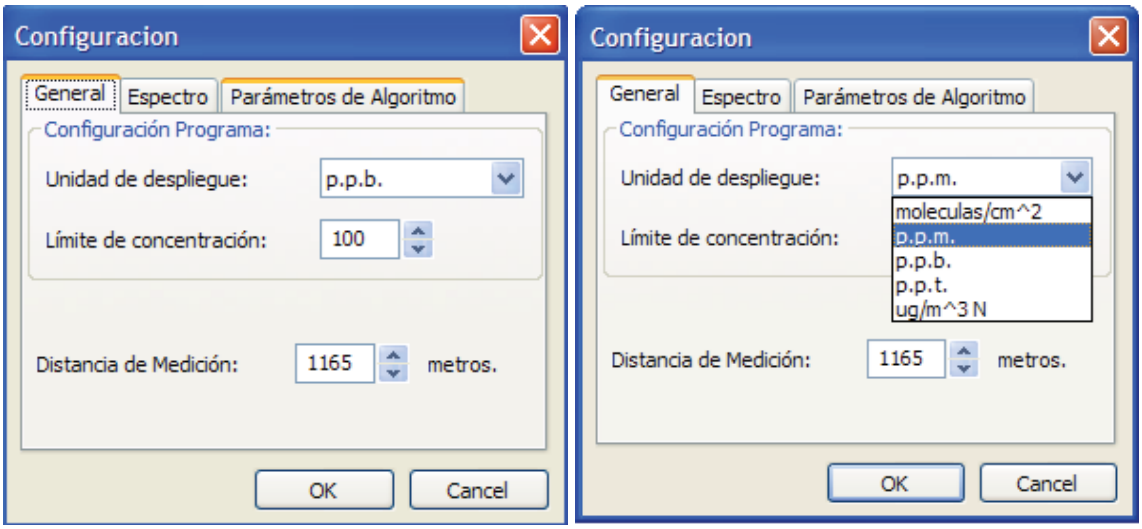


*Figura 3.2.2: Pantalla de configuración general*

El cambio de unidad de despliegue actualiza todos los gráficos de concentración, el historial del día y los gráficos de históricos abiertos.

En la configuración del espectrómetro se pueden configurar el tiempo de integración para la medición de espectro, activar o desactivar el temporizador para automatizar la toma de espectros y el tiempo de éste. En caso de no detectarse un espectrómetro compatible, esta ventana queda deshabilitada.

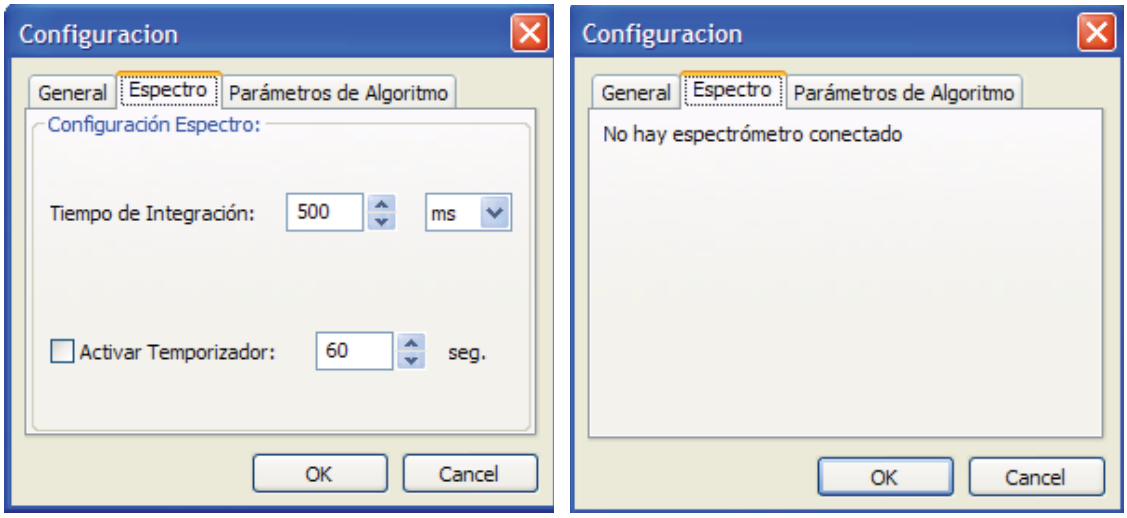


*Figura 3.2.3: Pantalla de configuración del espectrómetro*

En la configuración del algoritmo, se pueden configurar los órdenes de los polinomios a utilizar en el algoritmo para la eliminación de la parte del espectro que varía rápidamente en función de la longitud de onda y se puede modificar la región espectral en la que se realiza el análisis. Si se realiza un cambio en esta última opción o en el orden del polinomio de ajuste para la sección transversal, se vuelve a cargar el archivo de referencia correspondiente al contaminante a analizar para cargar los datos con los nuevos parámetros.


*Figura 3.2.4: Pantalla de configuración de parámetros del algoritmo*

Al escoger la opción de abrir un archivo de espectro guardado o la opción para cargar un histórico, aparece una ventana para escoger los archivos a cargar.

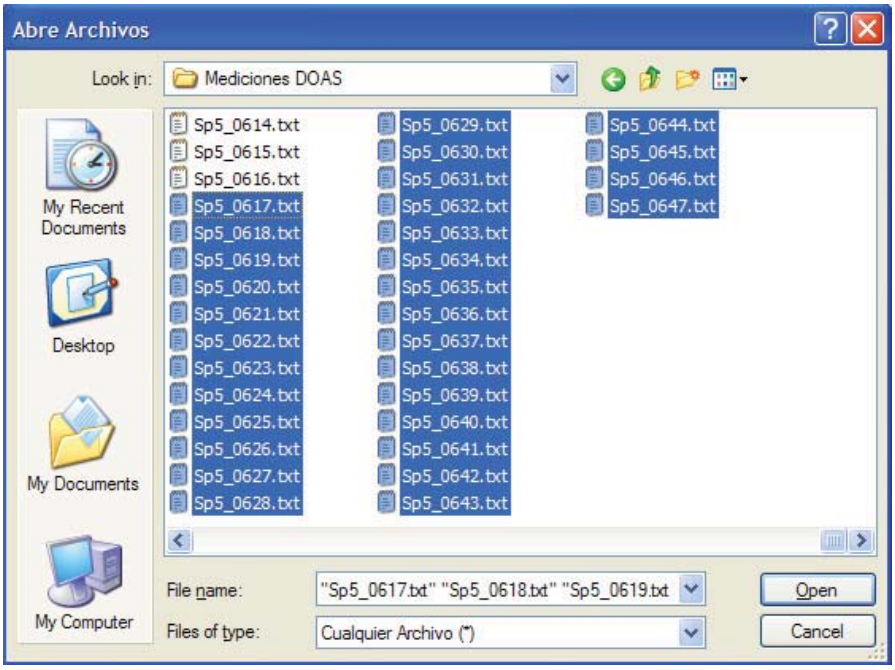


*Figura 3.2.5: Ventana de apertura de espectro guardado*

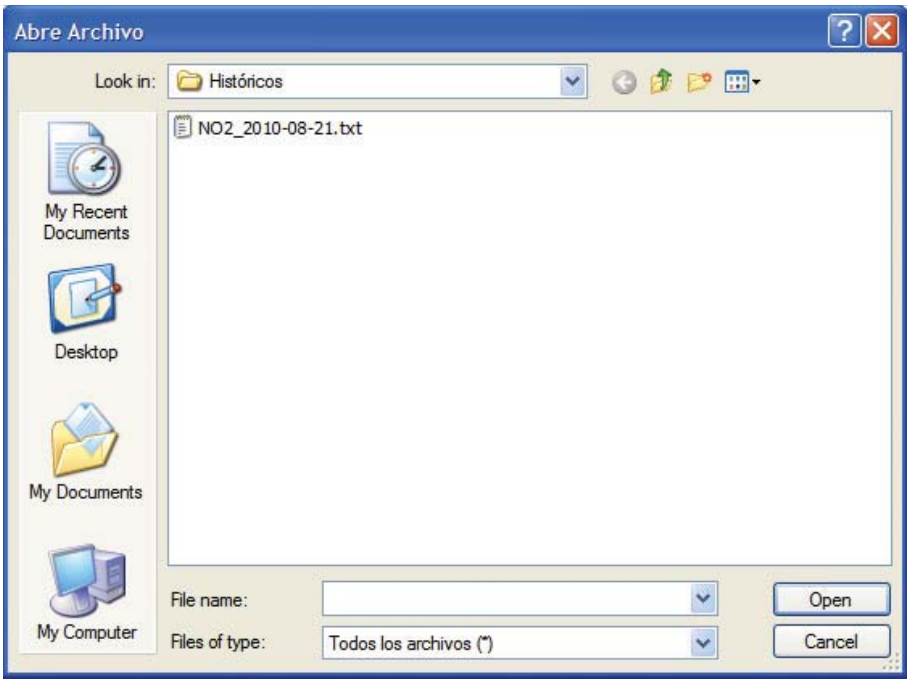


*Figura 3.2.6: Ventana de apertura de un archivo de históricos*

Al abrir un archivo de espectro, se cargan los valores almacenados en él y se calcula el valor de concentración con los parámetros especificados en la configuración para el contaminante seleccionado. Los resultados son graficados en la ventana principal 'Historia'. Para los valores que sobrepasen el límite especificado en la configuración, se despliega un mensaje en la parte inferior indicando el valor obtenido y la fecha en la cual se produjo esa concentración.

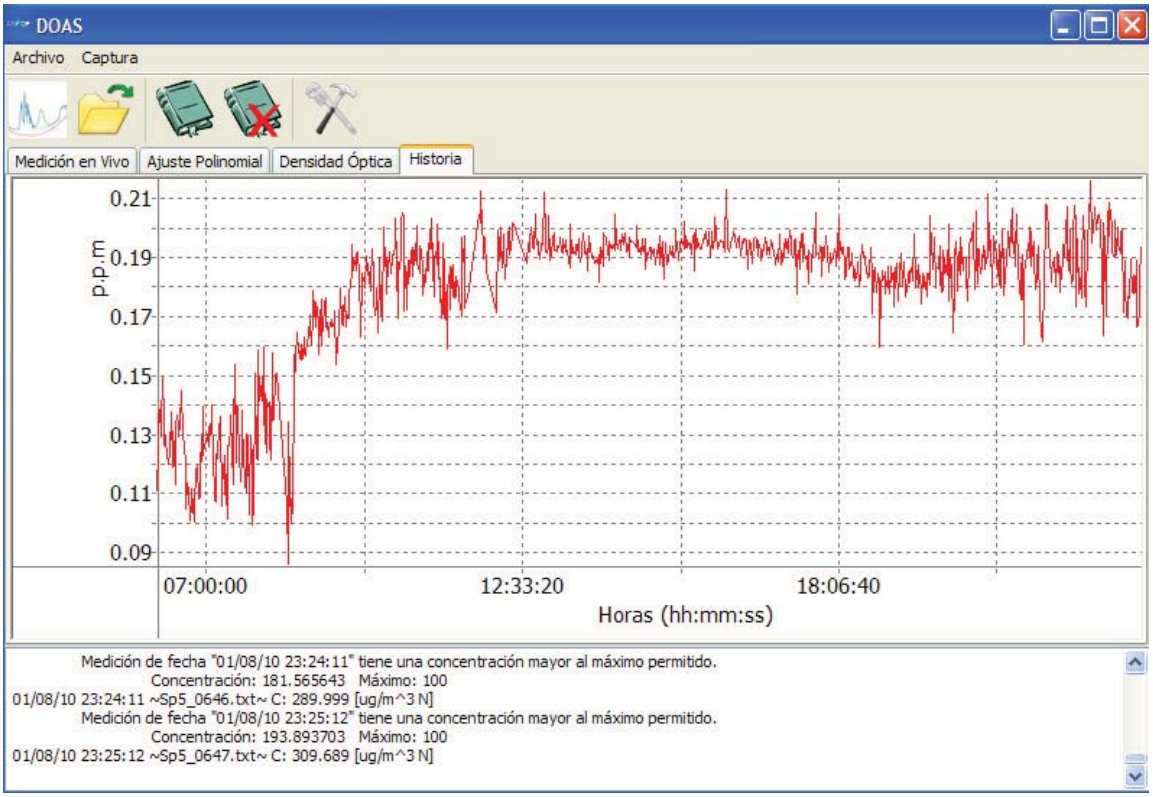


*Figura 3.2.7: Ventana principal, gráfica de los valores de concentración a lo largo de un día*

En el caso de cargar un histórico, se crea una nueva pestaña que despliega los valores guardados en el archivo. El nombre de la pestaña corresponde al nombre del archivo cargado y se muestra en las unidades que especificadas en la ventana de configuración.

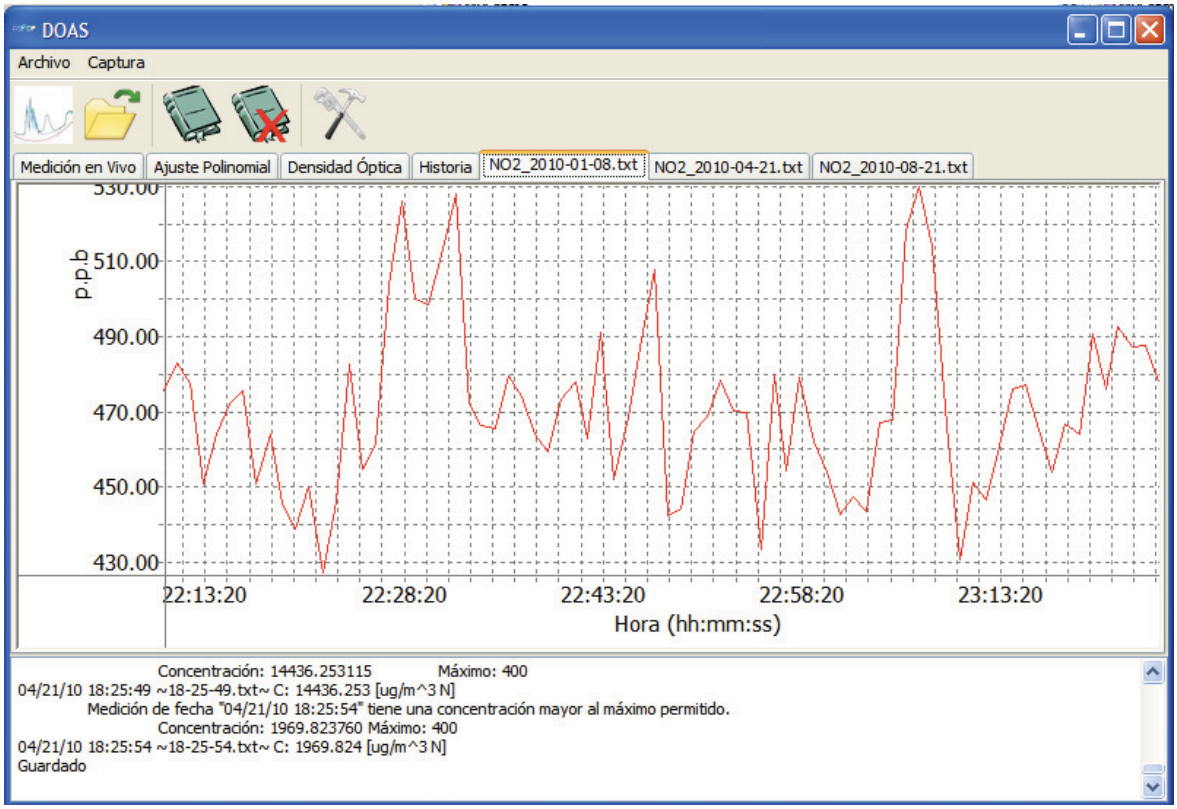


*Figura 3.2.8: Gráfico de históricos*

Una barra de botones permite el uso del programa sin utilizar, en su mayoría, los menús de la parte superior. Estos botones permiten capturar un espectro en caso de existir un espectrómetro conectado, abrir un archivo de espectro previamente almacenado, cargar archivos de históricos así como cerrar un histórico abierto y acceder a la ventana de configuración. La selección del contaminante a analizar sólo se puede escoger por medio del uso del menú.


*Figura 3.2.9: Panel de botones*

La pestaña “Medición en vivo” muestra el espectro que mide el espectrómetro, si es que se detecta alguno, en todo su rango de operación. Las pestañas ‘Ajuste polinomial’ y ‘Densidad Óptica’ grafican, respectivamente, la medición en el rango con su ajuste polinomial, y el espectro de

referencia con la densidad óptica del espectro analizado. Estos gráficos permiten la detección de errores en la selección de parámetros o en los procesos de medición.

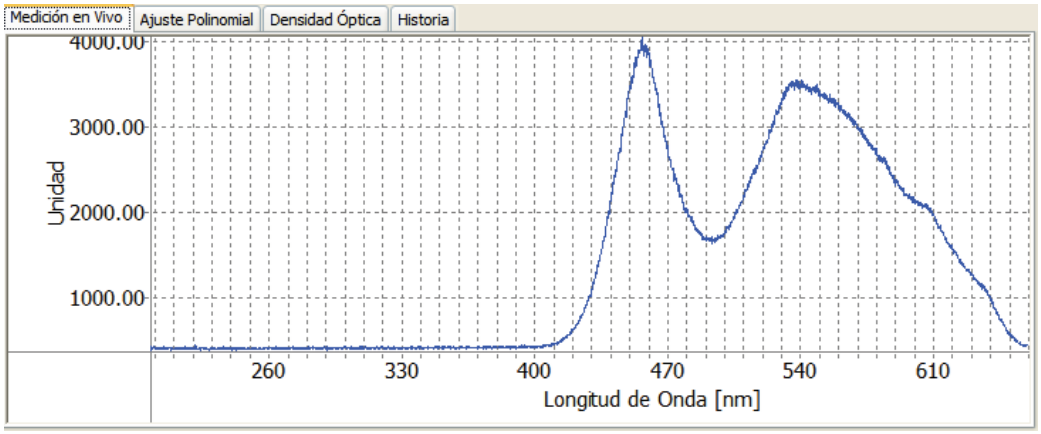


*Figura 3.2.10: Pestaña 'Medición en Vivo'*

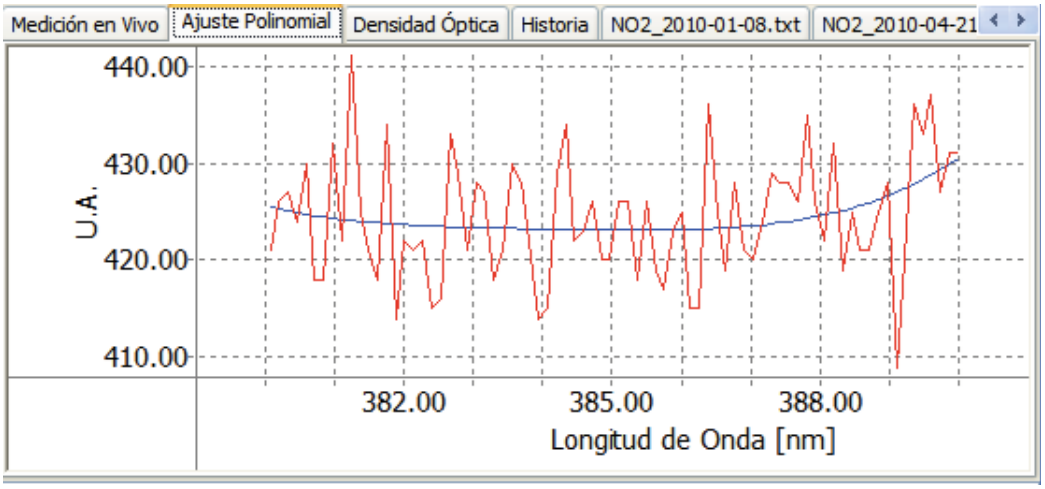


*Figura 3.2.11: Pestaña 'Ajuste Polinomial'*

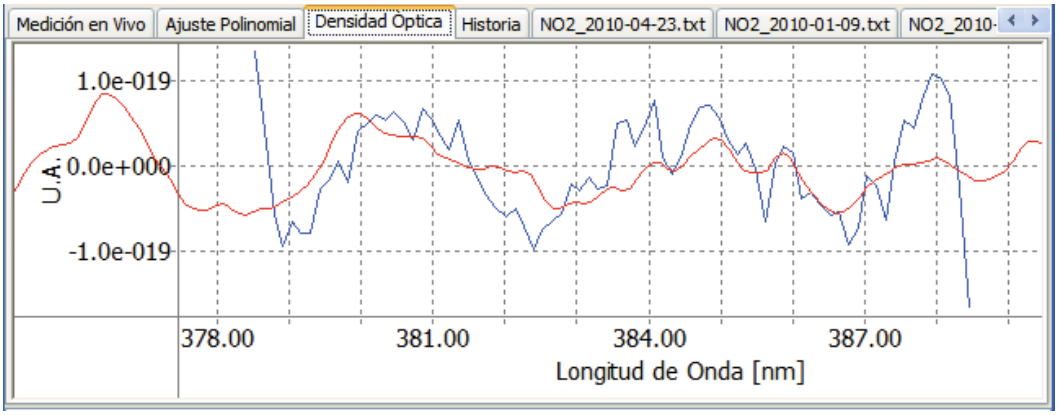


*Figura 3.2.12: Pestaña 'Densidad Óptica'*

# 4 Estructura del programa

## 4.1 Bloques funcionales

El programa de análisis de datos se descompone en diferentes bloques funcionales para las diferentes partes de la aplicación.

- **Configuración del Programa**: bloque de selección de parámetros del programa, tipo de contaminante a analizar, tiempo de integración del espectrómetro, distancia recorrida por la luz en la medición, límites de valores para alarmas.
- **Carga de sección eficaz:** carga la sección eficaz correspondiente al contaminante escogido para poder ser utilizado en el análisis de datos.
- **Comunicación con espectrómetro**: se encarga de la lectura de datos y configuración de parámetros del espectrómetro.
- **Análisis del espectro**: a partir de un espectro cargado, calcula el valor de concentración a partir del tipo de contaminante a medir, la sección eficaz a utilizar y la distancia recorrida por la luz en la medición.
- **Almacenamiento y carga de históricos**: almacenamiento de las concentraciones calculadas y los respectivos espectros medidos para posterior visualización. Carga de las concentraciones para ser desplegados en la interfaz gráfica.
- **Despliegue de gráficos**: muestra en pantalla los valores de las concentraciones obtenidos a partir de mediciones realizadas en diferentes instancias.
- **Informe de alarmas:** Almacenamiento de momento en que se sobrepase un valor límite configurado, desplegando el valor obtenido.

### 4.1.1 Comunicación entre bloques:

En la figura 1.1, se observa la comunicación de los diferentes módulos en el programa.

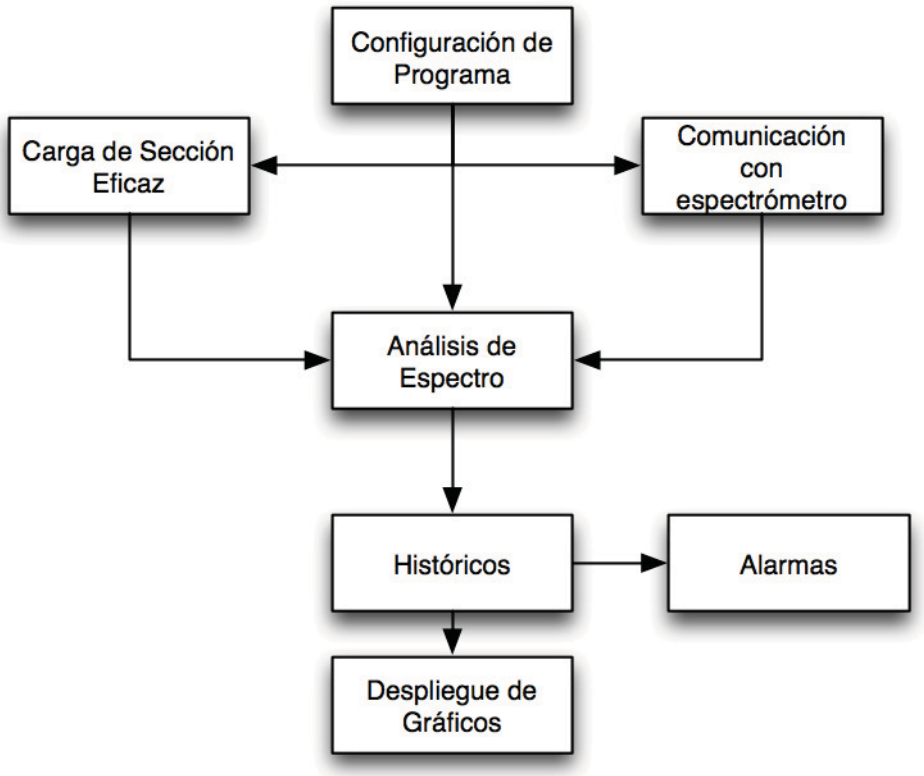


*Figura 4.1.1: Comunicación entre bloques funcionales*

### 4.1.2 Configuración del Programa:

Este módulo determina diferentes configuraciones del programa. La selección del contaminante a analizar se escoge desde el menú principal del programa, el resto de las configuraciones, desde la ventana de configuración (función 'Configuracion' en la clase 'MyFrame'). En esta ventana se configura la distancia recorrida por la luz durante la medición, el valor límite sobre el cual se alerta, la unidad de despliegue, el tiempo de integración del espectrómetro, el temporizador para automatizar mediciones y parámetros del algoritmo como el rango del espectro a analizar y el orden de los polinomios utilizados en el ajuste polinomial.

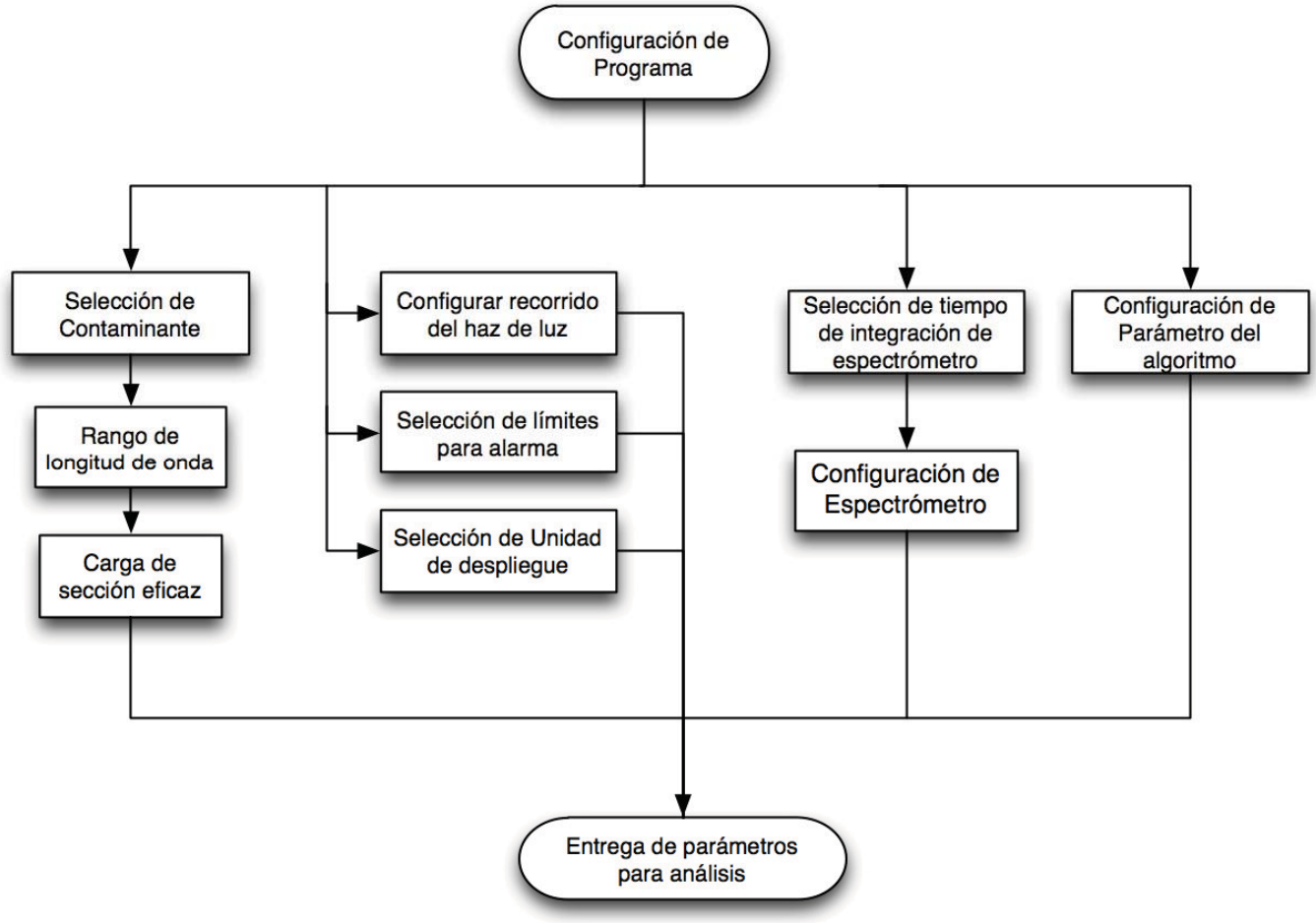


*Figura 4.1.2: Bloque de Configuración del Programa*

### 4.1.3 Carga de Sección Eficaz:

Carga el archivo correspondiente al contaminante a analizar, se cargan los valores dentro del rango de análisis y se obtienen los parámetros a utilizar en el análisis de las mediciones. (Función 'Carga_SeccEfic' en clase 'Myframe');

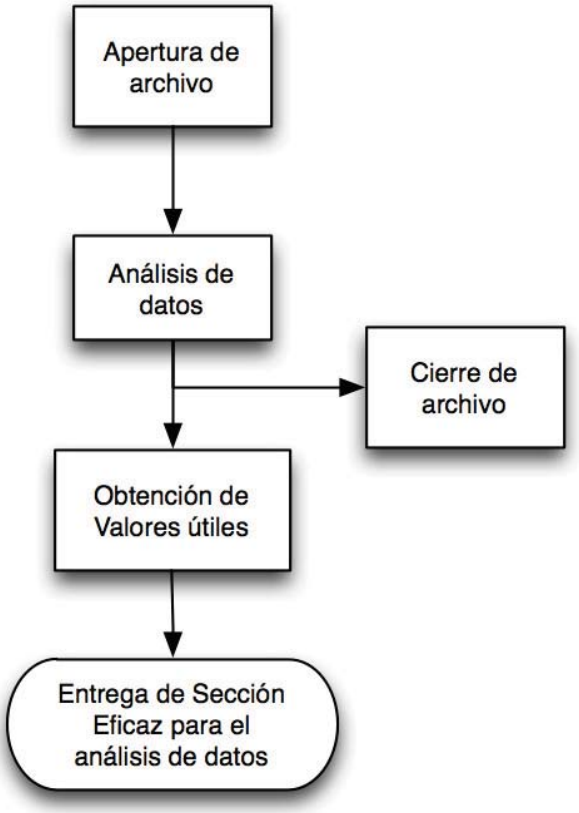


*Figura 4.1.3: Bloque de Carga de sección Eficaz*

### 4.1.4 Comunicación con espectrómetro:

En la comunicación con el espectrómetro se distinguen dos partes, la de configuración y la de medición de espectro.

En cuanto a configuración, se determina el tiempo de integración que utilizará el espectrómetro, esto es configurado en la pestaña de configuración del programa (función 'Configuracion' de la clase 'MyFrame').

` En cuanto a lectura de datos, a petición del programa (función 'Test_Espectro' y función 'Func_Temp_Espectro' en clase 'Myframe'), el espectrómetro mide los valores del espectro para el cual está diseñado los que son almacenados en el programa y utilizados en para el análisis de concentración.

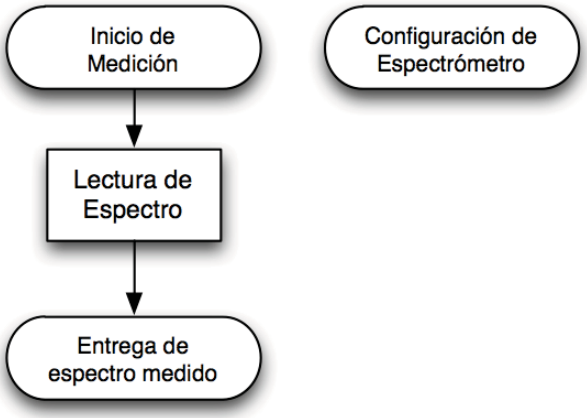


*Figura 4.1.4: Bloque de Comunicación con espectrómetro*

### 4.1.5 Análisis del espectro:

En este módulo se calculan los valores de concentración de los espectros medidos (función 'Calculo_LMS' de clase 'MyFrame'). Los parámetros necesarios para el análisis se obtienen del módulo de configuración y corresponden a la distancia recorrida por el haz de luz de las mediciones, los valores de la sección eficaz de absorción a diferentes longitudes de onda, el rango longitudes de onda en que se realiza el análisis y el orden del polinomio al cual se ajusta el espectro medido.

En el análisis se realiza el ajuste de la medición con la sección eficaz de absorción dentro del rango espectral correspondiente, obteniendo el valor de concentración del contaminante. Este resultado, el espectro medido y la hora en la que fue medido el espectro están disponibles para su almacenamiento.

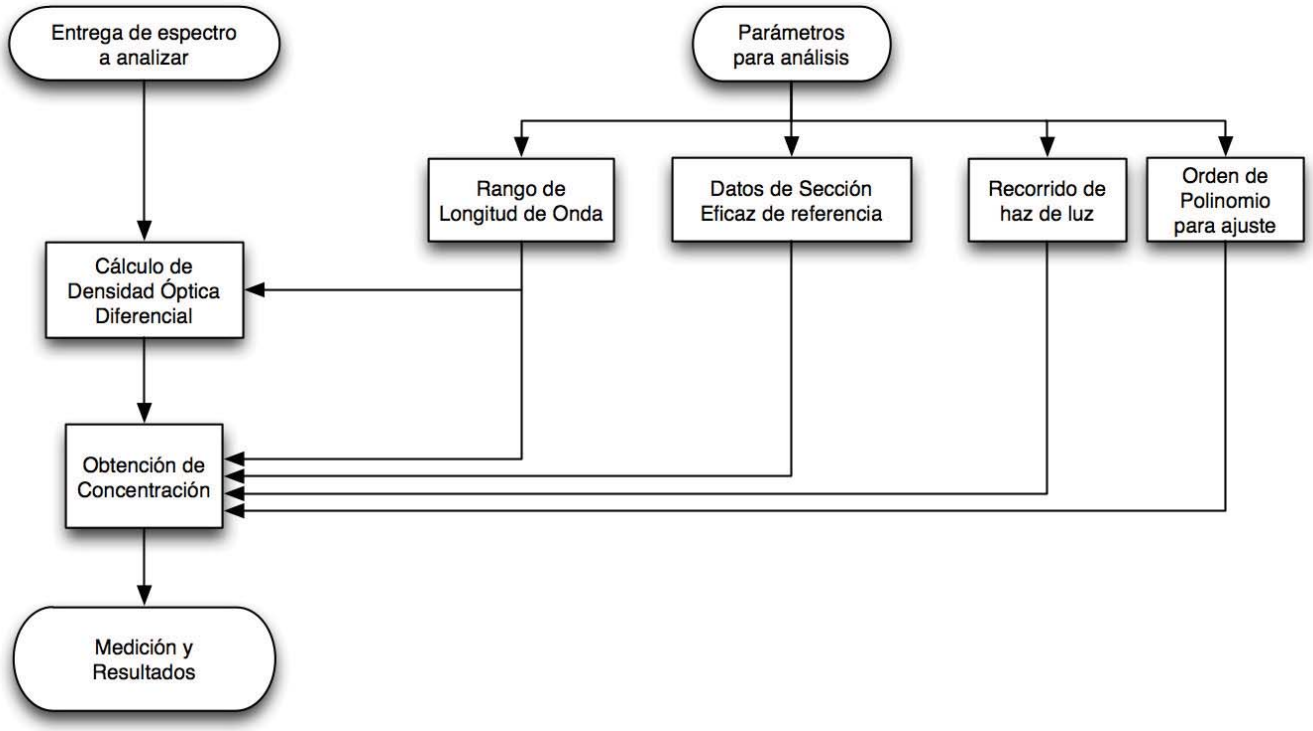


*Figura 4.1.5: Bloque de Análisis del Espectro*

### 4.1.6 Almacenamiento y carga de históricos:

En este módulo se realiza el almacenamiento y carga de los espectros medidos y las concentraciones calculadas junto con la hora respectiva de medición. Si la medición corresponde a la adquisición de datos por medio del espectrómetro, éstos son almacenados directamente en un archivo (función 'Guarda_medicion' en clase 'MyFrame'), para luego analizar los datos y calcular el valor de concentración. Si en cambio, la medición corresponde a la adquisición de datos por medio de la apertura de un archivo guardado con anterioridad (función 'Abrir' en clase 'MyFrame'), se procede directamente al análisis de los datos cargados. Durante el análisis de datos, se revisa si la concentración obtenida excede el valor límite fijado en cuyo aparece un mensaje de alerta en la ventana de alarmas junto con la hora de la medición y el valor obtenido.

Para cada espectro analizado, se obtiene un valor de concentración el cual se agrega al listado de concentraciones medidas durante el día (función 'Add_DATA' en clase 'clArchivo') y se actualizan los diferentes gráficos de la aplicación.

Al guardar un histórico (función 'Guarda' en clase 'clArchivo'), se almacenan los valores de concentración y la hora a la cual fueron obtenidos en un archivo para su posterior visualización.

Para el caso de apertura de un archivo histórico (función 'Abrir_Hist' en clase 'MyFrame'), se crea una nueva ventana para desplegar los datos y se cargan los valores de concentración del archivo.

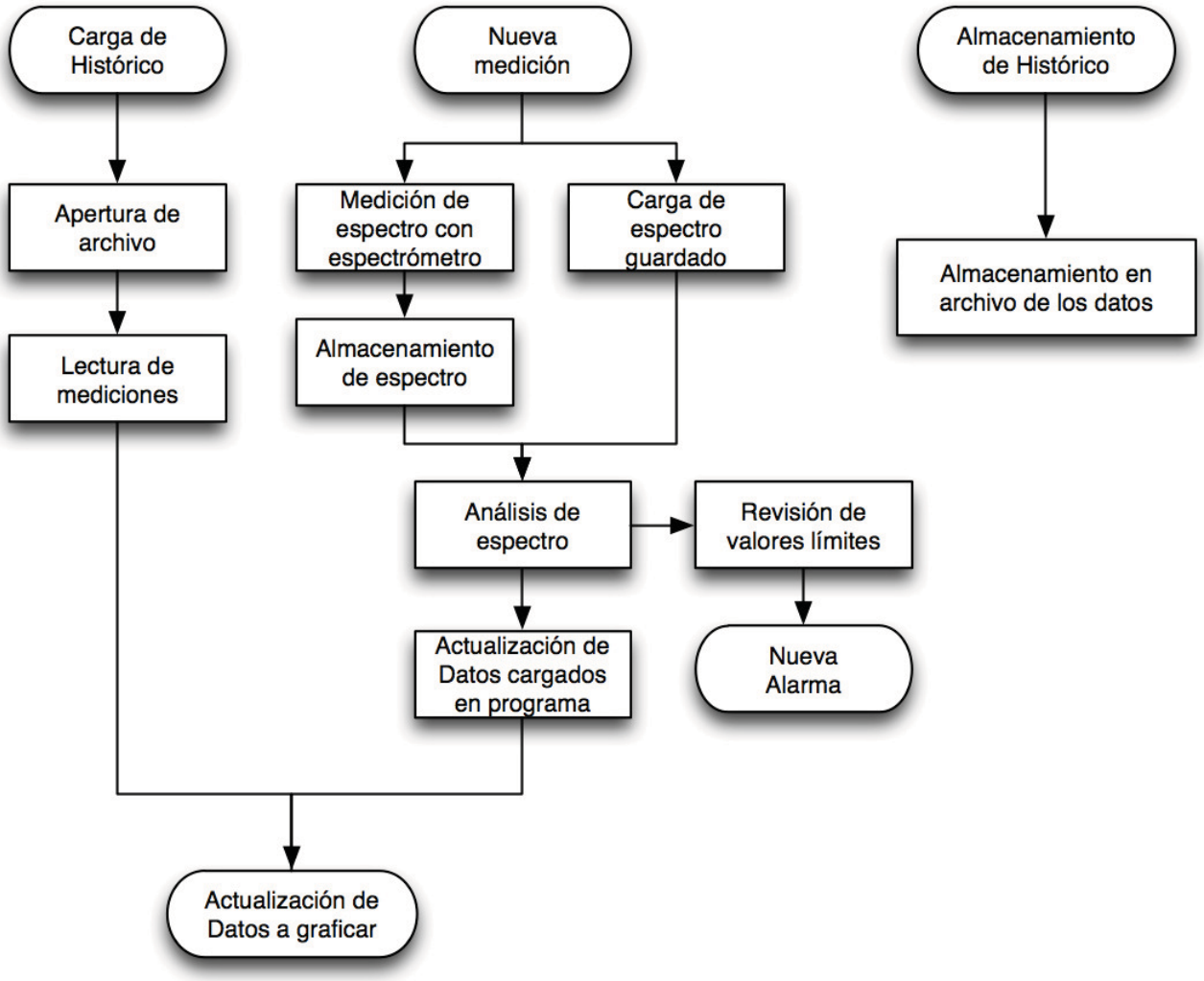


*Figura 4.1.6: Bloque de Almacenamiento y Carga de históricos*

### 4.1.7 Despliegue de gráficos:

Este módulo despliega las concentraciones almacenadas en el módulo de almacenamiento y carga de históricos, mostrando los valores de concentración calculados según la fecha y hora de medición del espectro analizado. Las curvas a graficar cargan los valores a graficar uno por uno (función 'Add_Curva' en clase 'mpCurva') o de un conjunto de datos ya cargados (función 'Set_Curva' en clase 'mpCurva'), tras los cuales se actualiza la ventana de los gráficos.

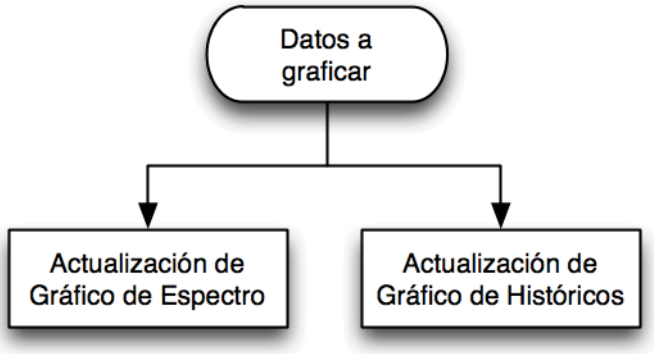


*Figura 4.1.7: Bloque de Despliegue de gráficos*

### 4.1.8 Informe de alarmas:

En caso que alguna concentración sobrepase un valor límite, se activa un mensaje de alerta, que despliega el valor de la concentración y la fecha en que ocurrió (variable 'alarma' de clase tipo 'wxTextCtrl'). Estos valores son mostrados en la interfaz gráfica del programa.

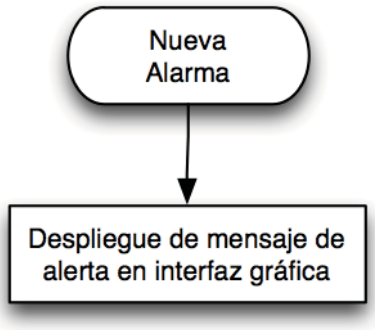


*Figura 4.1.8: Bloque de Informa de Alarmas*

## 4.2 Diagrama de Clases

El programa fue realizado utilizando la versión 2.8.10 de wxWidgets [13], una librería para crear aplicaciones mutiplataforma comunicándose con las API propias del sistema operativo para la creación de interfaz de usuario. Esta librería provee clases y funciones para el desarrollo de la interfaz gráfica del programa como los menus, uso de botones y diferentes ventanas de diálogos, así como manejo de temporizadores y control administración de eventos.

Para el gráfico de las curvas se utiliza wxMathPlot [14], en su versión 0.1.1, una librería que agrega funcionalidades de gráfico en 2D a wxWidgets. Las clases de las curvas a graficar son derivadas de clases de esta librería la cual también provee clases para las ventanas donde se grafican.

Otras librerías son utilizadas en el análisis de datos, la librería SPLINE [17] es utilizada para interpolar puntos de la sección eficaz de absorción y la librería GSL [15] que es utilizada para el ajuste polinomial en diferentes secciones del código y que es llamada a partir de la función 'polinomialfit' [16], la librería 'levmar' [18] permite utilizar el algoritmo de Levenberg-Marquardt para lograr el alineamiento de longitud de onda.

"MyFrame" es la clase base del programa, en ella se encuentran las funciones que realizan el análisis de datos, la configuración del programa y presenta la interfaz de usuario.

"mpCurva" posee la información de cada curva grafica en las distintas ventanas. Estas curvas son graficadas en las clases 'mpWindow'.

"clArchivo" es la clase las concentraciones de los espectros medidos y analizados así como la fecha y hora en que se realizó la medición.

"Wrapper" corresponde a la interfaz para la conexión del programa con el espectrómetro. Por medio de esta clase, se configura el espectrómetro y se obtienen las mediciones.

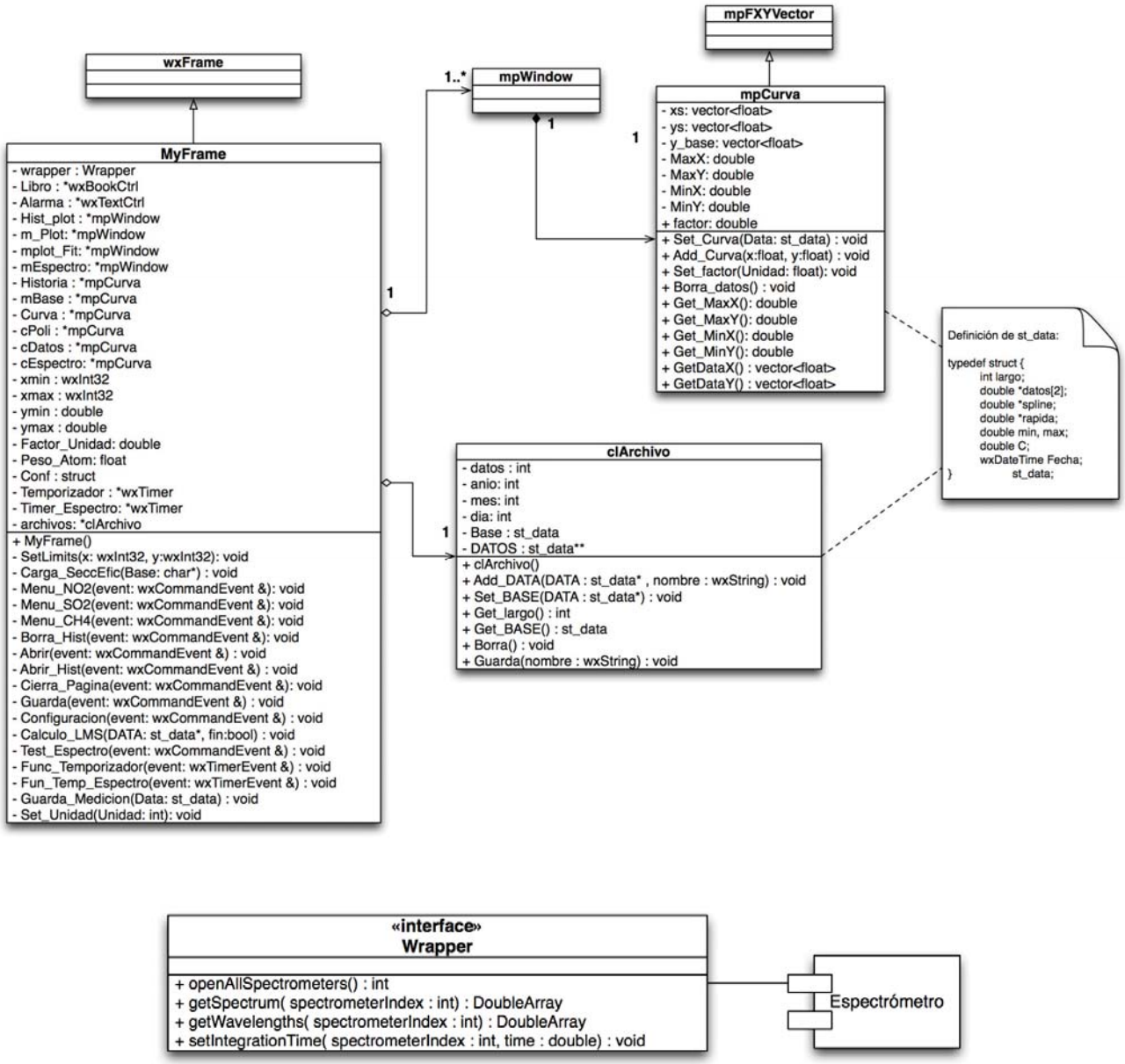


*Figura 4.2: Diagrama de Clases*

## 4.3 Diagrama de Casos de Uso

En el diagrama de casos de uso se observan las opciones de control que tiene un usuario de la aplicación sobre ella.

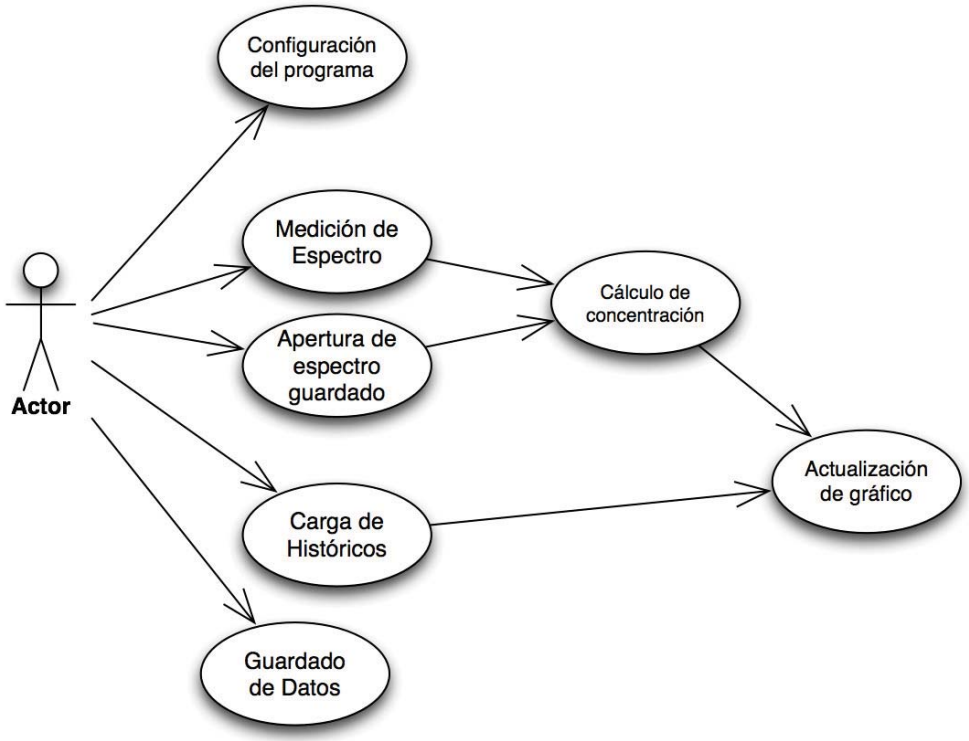


*Figura 4.3: Diagrama de Casos de uso*

# 5 Comparación de resultados

Para la verificación del programa, se comparan los resultados obtenidos por la aplicación creada con los resultados obtenidos mediante la aplicación del algoritmo en Matlab.

Los datos fueron obtenidos de mediciones realizadas en la Universidad de Concepción, el día viernes 8 de enero de 2010, entre las 6:05 hrs. y las 23:25 hrs. aproximadamente obteniendo una muestra por minuto. Se utilizó instrumentación DOAS activo bi-estático con un recorrido del haz de luz de 1.165 [m]. De este conjunto de datos, se utiliza una muestra de ella para la comparación de resultados obtenidos por ambos programas, algunos de los valores de concentración obtenidos en cada uno de los casos fueron tabulados en la tabla 5.1.

Los resultados obtenidos por ambas aplicaciones entregan valores muy cercanos, con errores de menos del 2% en la mayoría de ellos, por lo que se concluye que el algoritmo fue correctamente implementado. Cabe mencionar que el tiempo de ejecución del análisis de los datos en Matlab es considerablemente mayor que el tiempo de ejecución del análisis de los mismos datos en la aplicación creada. En Matlab, el calcular 100 concentraciones tarda alrededor de 11 minutos; en la aplicación, el calcular las 100 concentraciones correspondientes a las mismas mediciones que en el caso anterior, tarda entre 3 a 4 segundos, es decir, el programa es más de 100 veces más rápido.

Las curvas de concentración obtenidos para el conjunto de mediciones utilizadas en la comparación de resultados se observan en las figuras 5.1 y 5.2.

| ***Concentración obtenida de la aplicación creada [$\mu g/m^3$]*** | ***Concentración obtenida a partir de Matlab [$\mu g/m^3$]*** | ***Diferencia entre valores obtenidos [$\mu g/m^3$]*** | ***Diferencia Porcentual*** |
|---|---|---|---|
| 307,14 | 304,01 | 3,13 | 1,03% |
| 307,07 | 304,03 | 3,04 | 1,00% |
| 323,44 | 320,03 | 3,41 | 1,07% |
| 300,53 | 297,49 | 3,04 | 1,02% |
| 301,88 | 298,83 | 3,04 | 1,02% |
| 300,67 | 297,99 | 2,69 | 0,90% |
| 299,09 | 295,83 | 3,26 | 1,10% |
| 305,75 | 302,74 | 3,01 | 1,00% |
| 304,70 | 301,44 | 3,26 | 1,08% |
| 288,75 | 286,10 | 2,64 | 0,92% |
| 304,98 | 301,54 | 3,45 | 1,14% |
| 299,63 | 296,40 | 3,23 | 1,09% |
| 301,00 | 297,92 | 3,09 | 1,04% |
| 311,56 | 308,54 | 3,01 | 0,98% |
| 300,56 | 297,80 | 2,76 | 0,93% |
| 301,73 | 298,56 | 3,17 | 1,06% |
| 291,39 | 288,23 | 3,16 | 1,10% |

| | | | |
|---|---|---|---|
| 299,97 | 297,22 | 2,75 | 0,92% |
| 301,85 | 298,72 | 3,12 | 1,05% |
| 297,63 | 294,66 | 2,97 | 1,01% |
| 299,83 | 296,71 | 3,11 | 1,05% |
| 299,95 | 296,89 | 3,07 | 1,03% |
| 304,27 | 301,14 | 3,13 | 1,04% |
| 301,23 | 298,56 | 2,67 | 0,89% |
| 294,69 | 291,67 | 3,02 | 1,03% |
| 304,53 | 301,48 | 3,05 | 1,01% |
| 321,00 | 317,56 | 3,43 | 1,08% |
| 297,88 | 295,09 | 2,78 | 0,94% |
| 307,14 | 304,40 | 2,73 | 0,90% |
| 302,00 | 299,18 | 2,82 | 0,94% |
| 301,75 | 299,04 | 2,71 | 0,91% |
| 304,47 | 301,26 | 3,21 | 1,07% |
| 296,17 | 293,14 | 3,03 | 1,03% |
| 287,77 | 284,70 | 3,07 | 1,08% |
| 289,19 | 286,22 | 2,97 | 1,04% |
| 293,85 | 290,69 | 3,16 | 1,09% |
| 304,63 | 301,32 | 3,31 | 1,10% |
| 305,43 | 302,21 | 3,22 | 1,07% |
| 292,89 | 289,88 | 3,01 | 1,04% |
| 294,69 | 291,97 | 2,73 | 0,93% |
| 299,16 | 295,95 | 3,21 | 1,08% |
| 301,40 | 298,23 | 3,17 | 1,06% |
| 289,65 | 286,33 | 3,32 | 1,16% |
| 286,74 | 284,24 | 2,50 | 0,88% |
| 293,36 | 290,37 | 2,99 | 1,03% |
| 288,09 | 285,43 | 2,67 | 0,93% |
| 293,65 | 290,59 | 3,06 | 1,05% |
| 297,00 | 294,00 | 3,00 | 1,02% |
| 284,00 | 280,91 | 3,09 | 1,10% |
| 289,31 | 286,02 | 3,29 | 1,15% |
| 295,50 | 292,88 | 2,63 | 0,90% |
| 306,42 | 303,62 | 2,80 | 0,92% |
| 299,76 | 296,65 | 3,11 | 1,05% |
| 301,48 | 298,33 | 3,16 | 1,06% |
| 300,42 | 297,27 | 3,15 | 1,06% |
| 290,97 | 287,64 | 3,34 | 1,16% |
| 294,60 | 291,38 | 3,22 | 1,11% |
| 288,26 | 285,23 | 3,04 | 1,06% |
| 289,03 | 286,57 | 2,47 | 0,86% |
| 291,32 | 288,36 | 2,97 | 1,03% |
| 304,74 | 301,19 | 3,55 | 1,18% |
| 288,56 | 285,35 | 3,21 | 1,12% |
| 269,50 | 266,78 | 2,72 | 1,02% |
| 291,07 | 288,14 | 2,93 | 1,02% |
| 293,61 | 290,87 | 2,74 | 0,94% |
| 293,53 | 290,18 | 3,35 | 1,15% |
| 280,73 | 277,59 | 3,14 | 1,13% |
| 291,41 | 287,94 | 3,47 | 1,20% |
| 289,59 | 286,52 | 3,07 | 1,07% |

| | | | |
|---|---|---|---|
| 288,94 | 285,96 | 2,98 | 1,04% |
| 251,99 | 248,89 | 3,10 | 1,25% |
| 281,97 | 278,57 | 3,40 | 1,22% |
| 291,80 | 289,25 | 2,55 | 0,88% |
| 287,21 | 284,18 | 3,03 | 1,06% |
| 288,12 | 284,99 | 3,13 | 1,10% |
| 291,63 | 288,29 | 3,34 | 1,16% |
| 280,14 | 276,59 | 3,55 | 1,28% |
| 286,96 | 283,95 | 3,01 | 1,06% |
| 283,51 | 280,55 | 2,97 | 1,06% |
| 295,05 | 291,78 | 3,27 | 1,12% |

*Tabla 5.1: Comparación de resultados*

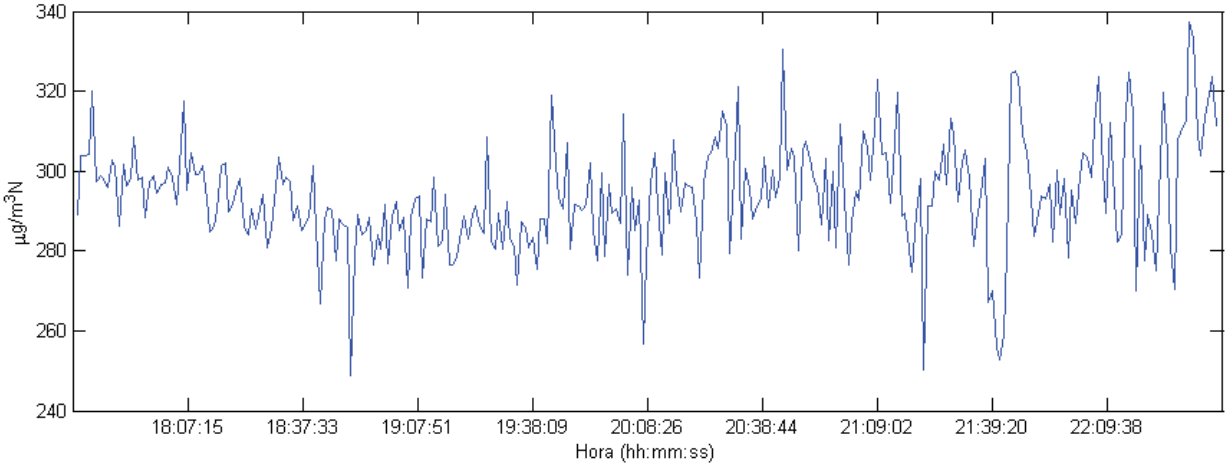


*Figura 5.1: Curva de concentración obtenida en Matlab*

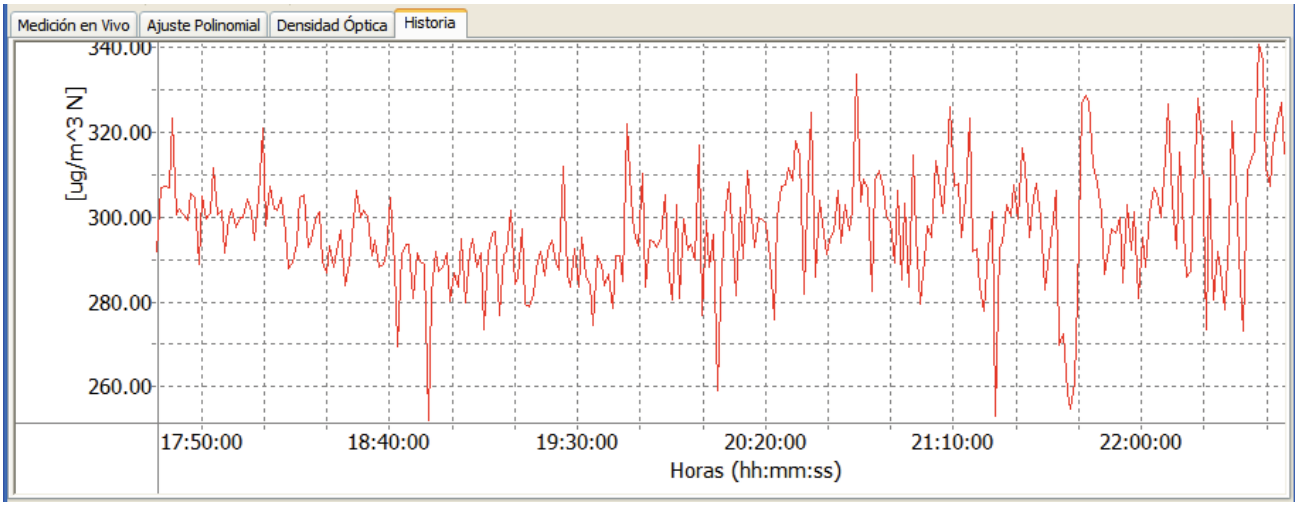


*Figura 5.2: Curva de concentración obtenida en la aplicación*

Si bien, la implementación del algoritmo es correcta, los valores obtenidos pueden no serlo. Primero que todo, aún no es posible validar los valores de concentración obtenidos al no poseer valores de concentración ya validadas con las cuales comparar, por lo que sólo se compara la implementación del algoritmo en la aplicación creada con el algoritmo realizado en Matlab. Por otro lado, los valores obtenidos no son realmente los esperados. Esto se debe a la lámpara utilizada en las mediciones la cual posee una modulación comparable con la sección transversal de absorción de $NO_2$ a medir dentro del rango escogido en el análisis. Este efecto de la lámpara esconde la absorción del contaminante por su baja concentración en la atmósfera, por lo que no se puede garantizar que los valores obtenidos reflejen valores reales de contaminante en el ambiente. De hecho, al revisar el espectro de la lámpara en el rango de longitud de onda utilizado en el análisis, la modulación de ella es bastante similar al espectro de referencia por lo que el ajuste entre una medición y el espectro de referencia de $NO_2$ puede no ser válido en esta ventana espectral.

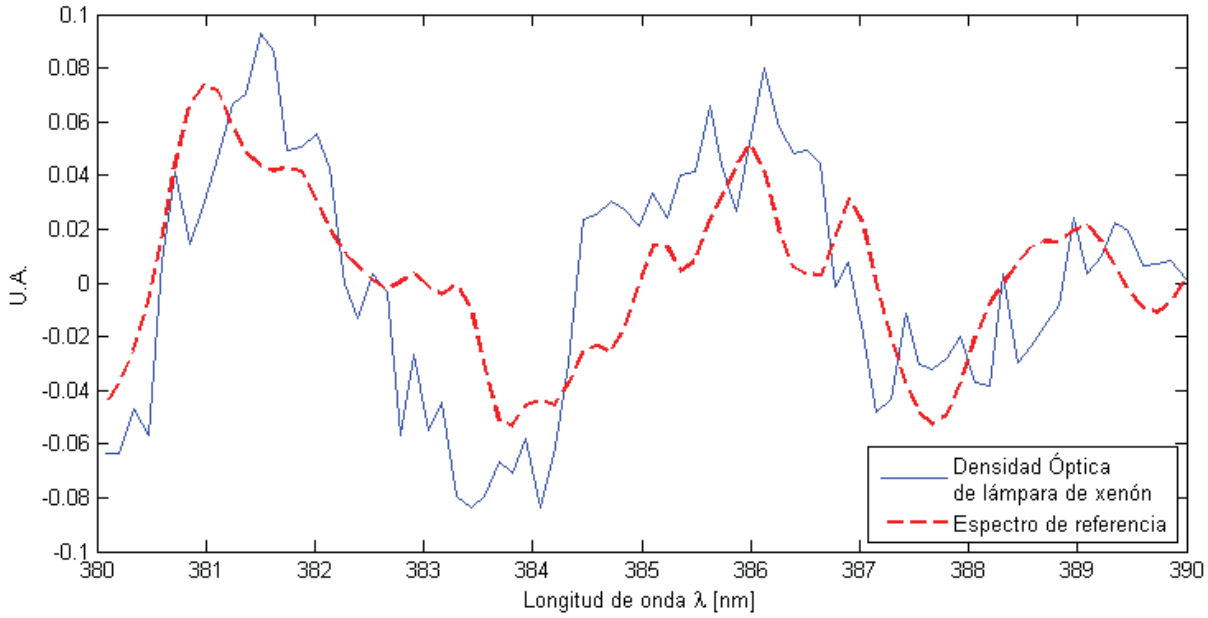


*Figura 5.3: Ajuste del espectro de la lámpara al espectro de referencia*

Una opción para superar este problema sería eliminar el efecto producido por la lámpara en cada medición; sin embargo, esto no es realizable por dos motivos: primero las variaciones rápidas de la intensidad con respecto a la longitud de onda para ambos espectros son del mismo orden y segundo, la intensidad de luz recibida de la lámpara varía con la temperatura ambiental produciendo

el efecto ‘mirage’, es decir, una pérdida del acoplamiento entre la luz final recibida por el telescopio y la fibra óptica del espectrómetro.

En el proceso del análisis DOAS se descarta aquellas interacciones de tipo radiación-materia, como los procesos de dispersión Rayleigh y Mie además de los efectos instrumentales, pues éstos varían lentamente en función de la longitud de onda. La lámpara de xenón, en cambio, posee variaciones rápidas con respecto a la longitud de onda, por lo que su efecto no se elimina del análisis. La figura 5.4 muestra el espectro de la lámpara de xenón utilizada en el rango 300[nm]-600[nm] mostrando un rizado en todo el espectro, por lo que tampoco es posible escoger otro rango para realizar el análisis de concentración del contaminante. La lámpara utilizada durante las mediciones no es la adecuada para este tipo de instrumentación, se requiere una cuyo espectro sea más suave variando lentamente en función de la longitud de onda.

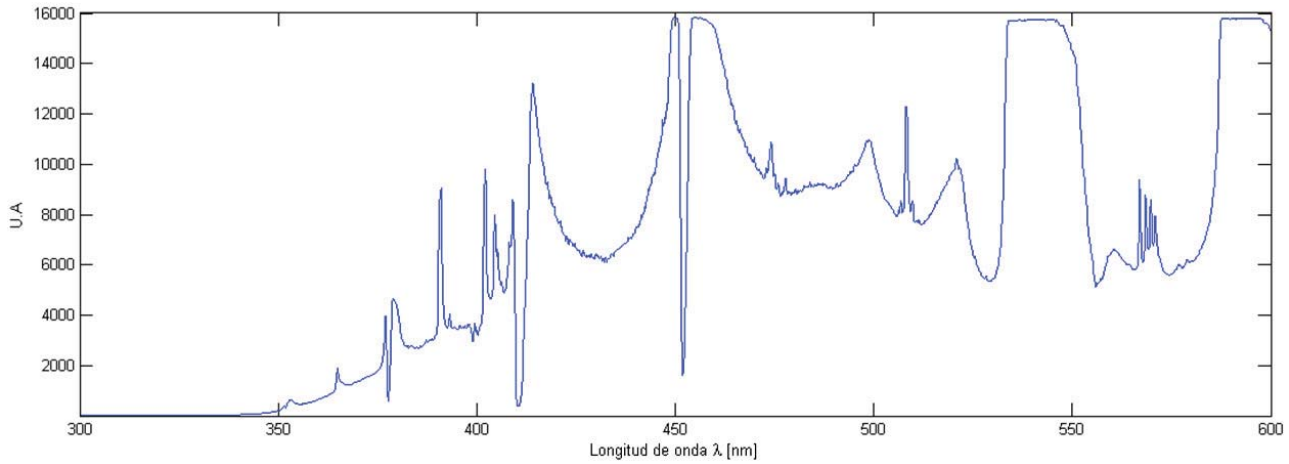


*Figura 5.4: Espectro de la lámpara de Xenón*

# 6 Trabajo futuro

## 6.1 Sumario

En este trabajo se presenta el algoritmo para el cálculo de concentraciones a partir de mediciones de espectro utilizando instrumentación DOAS activa. Se comparan los resultados de la implementación con los resultados de la aplicación del algoritmo en Matlab. En el presente documento no se verifica la calidad de las mediciones, sólo se realiza el análisis.

## 6.2 Conclusiones

La implementación del algoritmo para el análisis de datos de instrumentación DOAS en un programa propietario permite tener una única aplicación para el control de la instrumentación y el análisis de las mediciones, evitando el requerir más de un programa para la configuración del instrumento, adquisición de datos, almacenamiento de éstos, análisis y gráfico de los resultados. Además que es diseñado específicamente para la aplicación, buscando satisfacer los requerimientos específicos del usuario manteniendo un bajo requerimiento de poder de cálculo.

Por otro lado, la utilización de una aplicación diseñada para el análisis de datos permite un ahorro significativo en el tiempo de ejecución del análisis. El cálculo de concentración del contaminante a partir del conjunto de datos utilizados en la comparación de resultados tarda considerablemente menos, entre 100 y 200 veces menos, que el mismo cálculo en Matlab, siendo utilizable para cálculo de concentraciones de contaminantes en tiempo real.

Si bien la implementación del algoritmo en la aplicación es exitosa, obteniendo resultados similares al compararlos con los valores obtenidos del análisis de los mismos datos en Matlab, con errores bajo del 2%, los valores obtenidos no son validados por un instrumento patrón que indique los valores teóricos de concentración esperados. Por orto lado, la lámpara de xenón utilizada durante las mediciones no es la adecuada;  ella interfiere en el análisis de concentración, pues posee una modulación comparable con la de la sección transversal de absorción del $NO_2$ en el rango espectral del análisis. Una fuente de luz adecuada para este tipo de aplicación es aquella que posea un espectro más suave dentro de la ventana espectral en que se realice el análisis y no tenga una modulación comparable a la sección transversal de absorción del contaminante a analizar.

## 6.3 Trabajo Futuro

Si bien se logró el objetivo del trabajo, realizar una aplicación para calcular las concentraciones de contaminantes y que se comunicase con la instrumentación, los valores obtenidos no han sido validados con otro instrumento.

Como la lámpara no es la adecuada para este tipo de aplicación, a futuro se ha de repetir de mediciones y análisis con utilizando una fuente de luz que no produzca interferencia en los espectros medidos.

Es necesario realizar un proceso de prueba para la aplicación en terreno, destinándole un equipo y tiempo de operación para poder comprobar la estabilidad del programa.

Diversos métodos de análisis o filtros para el análisis con instrumentación DOAS pueden ser agregados en una futura iteración de la aplicación, como pueden ser el uso de la transformada wavelet, el análisis de componentes independientes [7] o la implementación de un filtro Kalman [8].

# *Bibliografía*


[1] "Differential Optical Absorption Spectroscopy" Ulrich Platt and Jochen Stutz. Springer, 2008.

[2] Christoph Kern; Sebastian Trick; Bernhard Rippel; Ulrich Platt, "Applicability of light-emitting diodes as light sources for active differential optical absorption spectroscopy measurements", *Optical Society of America*, vol.45, no.9, March 2006

[3] F. Xu; Z. Lv; X. Lou; Y. Zhang; Z. Zhang, "Nitrogen dioxide monitoring using a blue LED", *Optical Society of America*, vol.47, no.29, 2008.

[4] U. Frieß; K. Kreher; P. V. Johnston; Ulrich Platt, " Ground-Based DOAS Measurements of Stratospheric Trace Gases at Two Antarctic Stations during the 2002 Ozone Hole Period", *American Meteorological Society*, March 2005

[5] Wang Jing; Zhu Bin; Niu Shengjie, "Differential Optical Absorption Spectroscopy System for Suburb of Nanjing Atmospheric Pollution Monitoring," *Bioinformatics and Biomedical Engineering*, 2008. ICBBE 2008. The 2nd International Conference on , vol., no., pp.4070-4074, 16-18 May 2008.

[6] Yoshii, Y.; Kuze, H.; Takeuchi, N., "Differential optical absorption spectroscopy of atmospheric NO2 with a pulsed, white flashlight," *Lasers and Electro-Optics*, 2003. CLEO/Pacific Rim 2003. The 5th Pacific Rim Conference on , vol.2, no., pp. 682 vol.2-, 15-19 Dec. 2003.

[7] Sung-Hwan Han; Hyeon-Ho Kim; Dae-Sung Kim; Geun-Taek Ryu; Hun Choi; Dae-Young Lee; Hyeon-Deok Bae, "A New Method for Analysis of Differential Optical Absorption Spectra using Wavelet Transform and Independent Component Analysis," *Instrumentation and Measurement Technology Conference, 2005. IMTC 2005. Proceedings of the IEEE* , vol.2, no., pp. 1579-1584, 16-19 May 2005.

[8] Li SuWen; Liu WenQing; Xie PinHua; Zhang YuJun, "Application of Kalman Filtering and Wavelet Transform in DOAS," *Information Acquisition, 2006 IEEE International Conference on* , vol., no., pp.748-753, 20-23 Aug. 2006.

[9] Fabián A. Videla; Daniel C. Schinca; Jorge O. Tocho, "Alternative Method for Concentration Retrieval in Differential Optical Absorption Spectroscopy Atmospheric-Gas Pollutant Measurements," *Appl. Opt*. 42, 3653-3661, 2003

[10] "Optimization Engineering: Theory and Practice", Singeresu S. Rao, 3rd Edition, 1996.

[11] V. A. Buzanovskii; A. A. Bulaev, "Chemiluscent Gas Analizers", *Chemical and Petroleum Engineering*, vol.44, no.9-10, pp.514-518, Sept. 2008

[12] Chemiluminescent Measurement of NO/NOx in Gas Analysers, http://www.k2bw.com/chemiluminescence.htm

[13] wxWidgets, http://www.wxwidgets.org

[14] wxMathPlot, http://wxmathplot.sourceforge.net

[15] GNU Scientific Library, http://www.gnu.org/software/gsl/

[16] Rosetta Code, http://rosettacode.org/wiki/Polynomial_regression

[17] SPLINE, http://people.sc.fsu.edu/~burkardt/cpp_src/spline/spline.html

[18] levmar, Levenberg-Marquardt nonlinear least squares slgorithms in C/C++, http://www.ics.forth.gr/~lourakis/levmar/

# 7 Código Fuente

## A.1. Cabeceras

### A.1.1 clArchivo.h

```
#pragma once
#include "Frame.h"

class clArchivo{
	int datos,anio,mes,dia;                    // Variables para almacenar fecha del dia
	st_data BASE;
	st_data **DATOS;
public:
	clArchivo(){                               // Inicio de las variables de la clase
		datos                =       0;
		DATOS                =       NULL;
		BASE.datos[1]  =       NULL;
		BASE.datos[0]  =       NULL;
		BASE.rapida          =       NULL;
		BASE.spline          =       NULL;
	};
	~clArchivo();

	void Add_DATA(st_data *, wxString, mpCurva *);       // Agrega una medición
	void Set_BASE(st_data *);     // Carga los datos de la sección eficaz de referencia
	int Get_largo();              // Retorna el número de mediciones almacenadas
	st_data Get_BASE();           // Retorna los datos de la sección eficaz de referencia
	void Borra();                 // Borra los datos almacenados en la clase
	int Guarda(wxString);  // Guarda los datos a un archivo
};
```

### A.1.2 definicion.h

```
#pragma once
#define NO2_MIN 380           //      Inicio del rango de longitud de onda para NO2
#define NO2_MAX 390           //      Fin del rango de longitud de onda para NO2
#define SO2_MIN 295           //      idem para SO2
#define SO2_MAX 305
#define CH4_MIN 425           //      idem para CH4
#define CH4_MAX 447
#define NUM_AVOGADRO 6.02214179e23//  Numero de Avogadro
#define SO2_PA 64;            //      Peso atómico del SO2
#define NO2_PA 46;            //      Peso atómico del NO2
#define CH4_PA 10;            //      Peso atómico del NO2


#define TAM_ERROR 301  //      Largo del vector para el cálculo de corrimiento
#define MAX_DESP 1.5   //      Hasta cuánto corrimiento se revisa
#define PREC_DESP 100  //      Con una precisión de 0.01, son 100 puntos por unidad
                       //      Si se inicia en -1.5, con TAM_ERROR=301 se llega a +1.5
#define CH4_BASE "seNO2"      //      Carga del archivo de sección eficaz de referencia
#define NO2_BASE "seNO2"
#define SO2_BASE "seSO2"

#include <wx/datetime.h>

//Para el espectrómetro
#include "ArrayTypes.h"
#include "Wrapper.h"

typedef struct {
	int largo;
```

```
        double *datos[2];
        double *spline;
        double *rapida;
        double min, max;
        double C;
        wxDateTime Fecha;
}st_data;                       // Estructura para el almacenamiento de espectros

typedef struct {
        int Orden;
        double coef;
        double *Lambda;
        double *Espectro;
        st_data stBASE;
}       st_LevMar;

class clArchivo;
class MyFrame;

st_data *crea_st();
```

### A.1.3 Frame.h

```
if __PROJECT2FRM_h__
#define __PROJECT2FRM_h__

#ifdef __BORLANDC__
        #pragma hdrstop
#endif

#ifndef WX_PRECOMP
        #include <wx/wx.h>
        #include <wx/frame.h>
#else
        #include <wx/wxprec.h>
#endif

#include <wx/menu.h>

#include <wx/image.h>
#include <wx/listctrl.h>
#include <wx/sizer.h>
#include <wx/log.h>
#include <wx/intl.h>
#include <wx/math.h>
#include <wx/bookctrl.h>
#include <wx/file.h>
#include <wx/propdlg.h>
#include <wx/object.h>
#include <wx/spinctrl.h>
#include <wx/arrstr.h>
#include <wx/choice.h>
#include <wx/timer.h>
#include <wx/filefn.h>
#include <wx/textfile.h>

#include "spline.h"
#include "polifitgsl.h"
#include "definicion.h"
#include "PlotCurve.h"
#include "levmar.h"
#include "LevMar_func.h"

class MyFrame : public wxFrame
{
        private:
                DECLARE_EVENT_TABLE();

        public:
                MyFrame(wxWindow *parent,
```

```
                wxWindowID id,
                const wxString &title,
                const wxPoint& pos = wxDefaultPosition,
                const wxSize& size = wxDefaultSize,
                                long style = wxCAPTION | wxRESIZE_BORDER | wxSYSTEM_MENU |
wxMINIMIZE_BOX | wxMAXIMIZE_BOX | wxCLOSE_BOX);
                ~MyFrame();

        private:
                wxMenuBar *WxMenuBar1; // Crea la barra de herramientas

        private:
                enum                            // Asigna valores a diferentes eventos
                {
                        ID_MNU_ARCHIVO_1002 = 1002,
                        ID_MNU_ABRIR_1003 = 1003,
                        ID_MNU_SALIDA_1004 = 1004,
                        ID_MNU_AYUDA_1005 = 1005,
                        ID_MNU_CONTAMINANTE_1006 = 1006,
                        ID_MNU_NO2_1007 = 1007,
                        ID_MNU_CH4_1008 = 1008,
                        ID_MNU_SO2_1009 = 1009,
                        ID_MNU_BORRA_HIST = 1010,
                        ID_MNU_CAPTURA_1011 = 1011,
                        ID_MNU_GUARDA,
                        ID_MNU_ABRIR_HIST,
                        ID_MNU_CIERRA_PAG,
                        ID_MNU_CONFIG,
                        ID_BOOK,
                        ID_TIMER,
                        ID_TIMER_ESPECTRO,
                        ID_DUMMY_VALUE_
                };

        private:
                void OnClose(wxCloseEvent& event);
                void CreateGUIControls();                               // La barra de menu y botones
                mpWindow *m_plot,*Hist_plot, *m_plotFit, *mEspectro; // Ventanas de gráficos
                mpCurva *m_base, *Curva;                        // Curvas a graficar en las ventanas
                mpCurva *cPoli, *cDatos, *cEspectro;
                wxTextCtrl  *m_log;
                mpInfoLayer    *nfo, *nfo1;
                wxBookCtrl     *Libro;                          // Control de pestañas
                wxString strBASE,strREF;                        // Texto para guardado de datos

                void SetXlimits(wxInt32, wxInt32);              // Fija el rango de longitud de onda
                double getXmin() {return xmin;};      // Retorna valor de longitud de onda de inicio
                double getXmax() {return xmax;};      // Retorna valor de longitud de onda de término
                void Carga_SeccEfic(const wxChar *);  // Carga datos de sección eficaz de referencia
                void Abrir(wxCommandEvent&);          // Carga una medición previamente guardada
                void Abrir_Hist(wxCommandEvent&);     // Carga un archivo de históricos
                void Cierra_Pagina(wxCommandEvent&);  // Cierra una pestaña de históricos
                void Guarda(wxCommandEvent&);         // Guarda el histórico actual
                void Menu_CH4(wxCommandEvent&);       // Carga los datos de diferentes
                void Menu_NO2(wxCommandEvent&);       //      contaminantes a evaluar
                void Menu_SO2(wxCommandEvent&);
                void Borra_Hist(wxCommandEvent&);     // Borra historial del día actualmente cargado.
                void Configuracion(wxCommandEvent&);  // Abre la ventana de configuración
                void Calculo_LMS(st_data*, bool);     // Realiza el cálculo de concentración
                void Test_Espectro(wxCommandEvent&);  // Recibe una medición desde el espectrómetro
                void Func_Temporizador(wxTimerEvent &);// Temporizador: realiza una medición

                void Func_Temp_Espectro(wxTimerEvent &);// Temporizador: muestra el espectro en vivo
                void Guarda_medicion(st_data);        // Almacena en archivo medición de espectrómetro
                void Set_Unidad(int);                 // Fija la unidad de despliegue de los gráficos

                struct {
                        int Tiempo;
                        int Unidad;
                        int Distancia;
                        int Conc_Unidad;
                        int Max;
                        float Timer;
```

```
                        int Orden;
                        int Poli_Med;
                        int Poli_Ref;
                }       Conf;                   // Estructura de configuración
                Wrapper wrapper;                // Interfaz para controlar el espectrómetro
                wxInt32 xmin,xmax;              // Valores inicio y término rango de longitud de onda
                double ymin, ymax, Factor_Unidad;     // Valores máx/mín para gráficos y constante
para el cálculo de alarmas
                clArchivo *archivos;            // Clase para almacenamiento de historial diario
                mpCurva *Historia,**Curva_Historicos; // Curvas historial diario e históricos pasados
                mpWindow **Plot_Historicos;           // Ventanas para graficar históricos pasados
                int     numHistoricos, partida;       // Número de históricos abiertos
                                                      // variable para no escoger rango al cargar secc_eficaz
                wxTextCtrl  *Alarma;   // Cuadro de texto con informaciones alarmas
                float Peso_Atom;

                struct {
                        wxDateTime Ene;
                        wxDateTime Feb;
                        wxDateTime Mar;
                        wxDateTime Abr;
                        wxDateTime May;
                        wxDateTime Jun;
                        wxDateTime Jul;
                        wxDateTime Ago;
                        wxDateTime Sep;
                        wxDateTime Oct;
                        wxDateTime Nov;
                        wxDateTime Dic;
                }       Meses;                                  // Estructura con los meses del año

                wxTimer *Temporizador, *Timer_Espectro;             // Temporizadores
};

mpWindow *Crea_Pag(wxBookCtrl *, wxString, unsigned int);   // Función para crear pestaña
#endif
```

### A.1.4 MyApp.h

```
#ifndef __PROJECT2FRMApp_h__
#define __PROJECT2FRMApp_h__

#ifdef __BORLANDC__
        #pragma hdrstop
#endif

#ifndef WX_PRECOMP
        #include <wx/wx.h>
#else
        #include <wx/wxprec.h>
#endif

class MyApp : public wxApp
{
        public:
                bool OnInit();
                int OnExit();
};

#endif
```

### A.1.5 PlotCurve.h

```
#pragma once
#include "mathplot.h"
#include "definicion.h"
class mpCurva;

class mpCurva: public mpFXYVector            // mpFXYVector, de wxMathPlot
{
```

```
        std::vector<float> xs;              // Variable de longitud de onda
        std::vector<float> ys, y_base;      // Variable de concentración
    wxInt32 x_ini,x_fin;
    st_data DATA,BASE_SE;
    double *ypval, *yppval;                 // Variables para el cálculo de spline
        double MaxY,MinY,MaxX,MinX;         // Valores límites para graficar

public:
        double factor;                      // Factor para el cambio de unidad
        mpCurva(wxString      name    = wxEmptyString,
                        int                 flags   = mpALIGN_NE
                ):mpFXYVector(name,flags){ factor = 1; };
    void Set_Curva(st_data DataIn);                 // Carga los datos de la curva
        void Add_Curva(float x, float y);           // Agrega un par de datos a la curva
        void Set_factor(float Unidad);              // Cambia la unidad de despliegue
        void Borra_datos();                         // Vacía los datos
        double Get_MaxY(){ return MaxY;}            // Retorna los vlaores límites
        double Get_MinY(){ return MinY;}            // para graficar
        double Get_MinX(){ return MinX;}
        double Get_MaxX(){ return MaxX;}
        std::vector<float> GetDataX() {return xs;}; // Retorna los pares de valores
        std::vector<float> GetDataY() {return ys;}; // de la curva
};
```

## A.2. Fuentes:

### A.2.1 clArchivo.cpp

```
#include "clArchivo.h"
clArchivo::~clArchivo()
{
        Borra();
}

void clArchivo::Add_DATA(st_data *DATA, wxString nombre, mpCurva *Historia)
{
        int pre,post;
        if( datos !=0  &&                           // Si no está vacío
           (    dia!= DATA->Fecha.GetDay()    ||    // y la fecha es de un día
                mes!= DATA->Fecha.GetMonth()  ||    // distinto a las mediciones
                anio!= DATA->Fecha.GetYear()  )  )  // actuales
        {
                        Guarda(nombre);             // Guarda
                        Borra();                    // y borra
                        Historia->Borra_datos();
        }
        if( datos == 0 )                            // Si está vacío
        {
                anio = DATA->Fecha.GetYear();       // Carga la fecha actual
                mes     = DATA->Fecha.GetMonth();
                dia     = DATA->Fecha.GetDay();
        }
        int i;
        st_data **viejo,**nuevo;                    // Variables para guardas los datos
        viejo   =       DATOS;
        nuevo   =       (st_data **)malloc(sizeof(st_data*)*(datos+1));
        for(i=0 ; i<datos ; i++)
                nuevo[i]        =       DATOS[i];   // Copia lo datos anteriores
        nuevo[i]        =       DATA;               // Agrega el dato nuevo
        DATOS   =       nuevo;                      // Actualiza los punteros
        free(viejo);                                // Libera memoria
        datos++;                                    // Aumenta el contador de datos guardados
}
void clArchivo::Set_BASE(st_data *DATA)             // Fija los datos
{                                                   // de la sección eficas de referencia
        if(BASE.datos[0]!= NULL)      free(BASE.datos[0]);
        if(BASE.datos[1]!= NULL)      free(BASE.datos[1]);
        if(BASE.spline != NULL)       free(BASE.spline);
        if(BASE.rapida != NULL)       free(BASE.rapida);
```

```
        BASE    =       *DATA;
}
int clArchivo::Get_largo()
{       return datos; }
st_data clArchivo::Get_BASE()
{       return BASE; }
void clArchivo::Borra()                         // Borra los datos almacenados
{
        if (DATOS != NULL)
        {
                int i;
                for( i=0; i<datos ; i++)
                {
                        if(DATOS[i]->datos[0]!=NULL)  free(DATOS[i]->datos[0]);
                        if(DATOS[i]->datos[1]!=NULL)  free(DATOS[i]->datos[1]);
                        if(DATOS[i]->spline     !=NULL)       free(DATOS[i]->spline);
                        if(DATOS[i]->rapida  !=NULL)  free(DATOS[i]->rapida);
                        if(DATOS[i]                      !=NULL)       free(DATOS[i]);
                }
                free(DATOS);
                DATOS = NULL;
        }
        datos           = 0;                            // Ahora se tienen 0 datos guardados
};
int clArchivo::Guarda(wxString nombre)
{
        wxString Dir_old, Dir_nuevo;
        Dir_old         =       wxGetCwd();             // Carga el directorio de trabajo actual
        wxString texto;
        if ( !wxDirExists( "Históricos" ) )             // Crea carpeta 'Históricos', si no existe
                        wxMkdir( "Históricos" );
        if( wxDirExists("Históricos"))                  // Si existe
        {
                Dir_nuevo       =       Dir_old;
                Dir_nuevo.Append("\\");
                Dir_nuevo.Append("Históricos");
                wxSetWorkingDirectory(Dir_nuevo);       // Accede a la carpeta

                wxString temp,texto;
                temp.Printf("%04i-%02i-%02i.txt",anio,mes+1,dia);
                nombre.append(temp);                    // Agrega la fecha al nombre del archivo
                FILE *Archivo = fopen(nombre.c_str(),"w");   // Crea un archivo a guardar
                if( Archivo != NULL)
                {
                        texto.Printf("Historial de Concentraciones [g/cm^2]\n"
                                "++++++++++++++++++++++++++++++++++\n");
                        temp.Printf("Fecha de Medición: %02i/%02i/%4i\n",dia,mes+1,anio);
                        texto.Append(temp);
                        temp.Printf("Número de mediciones en el día: %i\n", datos);
                        texto.Append(temp);
                        temp.Printf(">>>>>Inicio de datos<<<<<\n");
                        texto.Append(temp);
                        int i;
                        for (i=0 ; i<datos ; i++)
                        {
                                temp.Printf("%i\t%f\n",       // Guarda los datos
                                                            DATOS[i]->Fecha.GetHour()*60*60 +
                                                            DATOS[i]->Fecha.GetMinute()*60 +
                                                            DATOS[i]->Fecha.GetSecond(),
                                                            DATOS[i]->C);
                                texto.Append(temp);
                        }
                        temp.Printf("0\t0\0");
                        texto.Append(temp);
                        fprintf(Archivo,texto.c_str());
                        fclose(Archivo);
                        wxSetWorkingDirectory(Dir_old);
                        return 0;
                }
                else
                {
                        wxSetWorkingDirectory(Dir_old);       //Si no se pudo acceder
                        return 1;                             //al directorio 'Históricos'
```

```
                }
        }
        else                            // Si no existe y no se creó la carpeta
                return 2;
}
```

### A.2.2 MyApp.cpp

```
#include "MyApp.h"
#include "Frame.h"
IMPLEMENT_APP(MyApp)
bool MyApp::OnInit()
{
    MyFrame* frame = new MyFrame(NULL, wxID_ANY, wxT("DOAS"),wxDefaultPosition, wxSize(800, 600));
    SetTopWindow(frame);
    frame->Show();
    return true;
}

int MyApp::OnExit()
{
        return 0;
}
```

### A.2.3 MyFrame.cpp

```
#include "Frame.h"
#include "clArchivo.h"

st_data *crea_st()
{
        st_data *st=(st_data *)malloc(sizeof(st_data));
        st->C           = 0;
        st->Fecha       = wxDateTime::Now();
        st->largo       = 0;
        st->max         = 0;
        st->min         = 0;
        st->datos[0]= NULL;
        st->datos[1]= NULL;
        st->spline      = NULL;
        st->rapida      = NULL;
        return st;
}
void Ajuste_Lambda2(double *p, double *x, int m, int n, void *data)
{                                               // Función a minimizar
        st_LevMar *LevMar_data = (st_LevMar *) data;
//      x       =       Datos esperados -> espectro medido
//      n       =       Número de "longitudes de onda" ; dim de x
//      p       =       Corrimiento de longitud de Onda
//      m       =       1; dimensión de p
        double *Lambda =        LevMar_data->Lambda;
        st_data BASE    =       LevMar_data->stBASE;
        double ypval, yppval;

        double Base;
        register int i;
        for(i=0; i<n; ++i)
        {
                Base    =       spline_cubic_val(BASE.largo,BASE.datos[0],
                                        Lambda[i]+p[0],               //Desplazamiento en eje X
                                        BASE.datos[1], BASE.spline, &ypval, &yppval );
                x[i]    =       Base*LevMar_data->coef - LevMar_data->Espectro[i];
        }
}
mpWindow *Crea_Pag(wxBookCtrl *Libro, wxString Titulo, unsigned int mode,wxString xlabel, wxString
ylabel)
{                                       // Crea pestaña en ventana principal
        wxPanel *panel = new wxPanel(Libro);                                // Crea página en 'libro'
        mpWindow *m_plot=new mpWindow( panel, wxID_ANY,      //Crea pantalla para dibujar
                                wxPoint(0,0), wxSize(500,500), wxSUNKEN_BORDER);
```

```
        wxBoxSizer *panelsizer = new wxBoxSizer( wxVERTICAL);//Ajusta pantalla a tamaño de la ventana
        panelsizer->Add( m_plot, 1, wxEXPAND );
        panel->SetSizer(panelsizer);
        Libro->AddPage(panel,Titulo,true,-1);
        wxString X,Y;
        X.Printf("                                                    ");
        X.Append(xlabel);
        Y.Printf("                    ");
        Y.Append(ylabel);
        wxFont graphFont(11, wxFONTFAMILY_DEFAULT, wxFONTSTYLE_NORMAL, wxFONTWEIGHT_NORMAL);
        mpScaleX* xaxis = new mpScaleX(X, mpALIGN_BOTTOM, true, mpX_NORMAL);
        mpScaleY* yaxis = new mpScaleY(Y, mpALIGN_LEFT, true);
        xaxis->SetFont(graphFont);
        yaxis->SetFont(graphFont);
        xaxis->SetLabelMode(mode);
        xaxis->SetDrawOutsideMargins(false);
        yaxis->SetDrawOutsideMargins(false);
        xaxis->SetTicks(false);
        yaxis->SetTicks(false);
        m_plot->SetMargins(0, 0, 50, 100);
        m_plot->AddLayer(     xaxis );
        m_plot->AddLayer(     yaxis );
        return m_plot;
}

BEGIN_EVENT_TABLE(MyFrame,wxFrame)                          // Define tabla de eventos
        EVT_CLOSE(MyFrame::OnClose)
        EVT_MENU(ID_MNU_ABRIR_1003, MyFrame::Abrir)
        EVT_MENU(ID_MNU_BORRA_HIST, MyFrame::Borra_Hist)
        EVT_MENU(ID_MNU_CH4_1008, MyFrame::Menu_CH4)
        EVT_MENU(ID_MNU_NO2_1007, MyFrame::Menu_NO2)
        EVT_MENU(ID_MNU_SO2_1009, MyFrame::Menu_SO2)
        EVT_MENU(ID_MNU_CAPTURA_1011, MyFrame::Test_Espectro)
        EVT_MENU(ID_MNU_GUARDA, MyFrame::Guarda)
        EVT_MENU(ID_MNU_ABRIR_HIST, MyFrame::Abrir_Hist)
        EVT_MENU(ID_MNU_CIERRA_PAG, MyFrame::Cierra_Pagina)
        EVT_MENU(ID_MNU_CONFIG, MyFrame::Configuracion)
        EVT_TIMER(ID_TIMER,MyFrame::Func_Temporizador)
        EVT_TIMER(ID_TIMER_ESPECTRO,MyFrame::Func_Temp_Espectro)
END_EVENT_TABLE()

MyFrame::MyFrame(wxWindow *parent, wxWindowID id, const wxString &title, const wxPoint &position,
                const wxSize& size, long style) : wxFrame(parent, id, title, position, size, style)
{
        archivos = new clArchivo();
        CreateGUIControls();
        strBASE.Printf(_T("NULL"));

        // Crea el libro
        Libro   =       new wxBookCtrl(this,ID_BOOK);
        wxBoxSizer *topsizer = new wxBoxSizer( wxVERTICAL);//Ajusta el libro al tamaño de la ventana
        topsizer->Add( Libro, 1, wxEXPAND );
        wxFont graphFont(11, wxFONTFAMILY_DEFAULT, wxFONTSTYLE_NORMAL, wxFONTWEIGHT_NORMAL);

        //      < -- Creación de Página de espectro en vivo          -- >
        mEspectro = Crea_Pag(Libro,_T("Medición en Vivo"),mpX_NORMAL,
                                    _T("Longitud de Onda"),_T("Unidad"));
        cEspectro       =       new mpCurva(wxString(_T("Curva")),mpALIGN_NE);
        wxPen Lapiz(*wxBLUE, 1, wxSOLID);
        cEspectro->SetPen(Lapiz);
        cEspectro->SetContinuity(true);
        mEspectro->AddLayer(cEspectro);
        mEspectro->EnableDoubleBuffer(true);
        mEspectro->SetMPScrollbars(true);

//      < -- Creación de Página de ajuste polinomial         -- >
        m_plotFit       =       Crea_Pag(Libro,_T("Ajuste Polinomial"), mpX_NORMAL,
                                        _T("Longitud de Onda"), _T("Unidad"));
        cPoli           =       new mpCurva(wxString(_T("Polinomio")),mpALIGN_NE);
        Lapiz.SetColour(*wxBLUE);
        cPoli->SetPen(Lapiz);
        cPoli->SetContinuity(true);
        cDatos          =       new mpCurva(wxString(_T("Datos")),mpALIGN_NE);
```

```
        Lapiz.SetColour(*wxRED);
        cDatos->SetPen(Lapiz);
        cDatos->SetContinuity(true);
        m_plotFit->AddLayer(cPoli);
        m_plotFit->AddLayer(cDatos);
        m_plotFit->EnableDoubleBuffer(true);
        m_plotFit->SetMPScrollbars(true);
//      < -- Fin de página de ajuste Polinomial             -- >

//      < -- Creación de Página de espectro actual          -- >
        m_plot = Crea_Pag(Libro,_T("Densidad Óptica"), mpX_NORMAL,
                          _T("Longitud de Onda"), _T("Unidad"));
        Curva   =       new mpCurva(wxString(_T("Curva")),mpALIGN_NE);
        Lapiz.SetColour(*wxBLUE);
        Curva->SetPen(Lapiz);
        Curva->SetContinuity(true);
        m_base  =       new mpCurva(wxString(_T("Curva Base")),mpALIGN_NE);
        Lapiz.SetColour(*wxRED);
        m_base->SetPen(Lapiz);
        m_base->SetContinuity(true);
//      < -- Fin de Página de espectro actual        -- >

        m_log = new wxTextCtrl( this, -1, wxT("This is the log window.\n"),wxDefaultPosition,
                          wxSize(0,0), wxTE_MULTILINE|wxTE_READONLY);
        wxLog *old_log = wxLog::SetActiveTarget( new wxLogTextCtrl( m_log ) );
        delete old_log;
        Alarma = new wxTextCtrl( this, -1, wxT("\t\tAlarmas.\n"),
                          wxDefaultPosition, wxSize(500,100), wxTE_MULTILINE|wxTE_READONLY);
        topsizer->Add(Alarma,0,wxEXPAND);

        st_data *DATA = (st_data *)malloc(sizeof(st_data));
        DATA->datos[0]      = (double *)malloc(sizeof(double)*2);
        DATA->datos[0][0]   = 0;
        DATA->datos[0][1]   = 1;
        DATA->datos[1]      = (double *)malloc(sizeof(double)*2);
        DATA->datos[1][0]   = 0;
        DATA->datos[1][1]   = 0;
        DATA->largo         = 2;
        DATA->spline        = spline_cubic_set(DATA->largo,DATA->datos[0],DATA->datos[1],0,0,0,0);
        Curva->Set_Curva(*DATA);
        m_base->Set_Curva(*DATA);
        cEspectro->Set_Curva(*DATA);
        mEspectro->Fit(200,700,0,15000);
        free(DATA);
        m_plot->AddLayer(Curva);
        m_plot->AddLayer(m_base);
        m_plot->EnableDoubleBuffer(true);
        m_plot->SetMPScrollbars(true);
        m_plot->Fit();

//      < -- Creación de Página de datos actuales           -- >
        Hist_plot = Crea_Pag(Libro,_T("Historia"), mpX_TIME, _T("Horas (hh:mm:ss)"), _T("Unidad"));
        Historia  = new mpCurva(wxString(_T("History")),mpALIGN_NE);
        Hist_plot->AddLayer(Historia);
        Hist_plot->EnableDoubleBuffer(true);
        Hist_plot->SetMPScrollbars(true);
        Hist_plot->Fit();                           //Ajusta el espacio a las curvas.
        Historia->Borra_datos();
        Historia->SetPen(Lapiz);
        Historia->SetContinuity(true);
//      < -- Fin de Página de Datos actuales         -- >

//      < -- Carga Valores de configuración          -- >
        wxFile Archivo;
        if(Archivo.Exists("DOAS.cnf"))
        {
                Archivo.Open("DOAS.cnf");
                Archivo.Read(&(Conf.Conc_Unidad),sizeof(int));
                Archivo.Read(&(Conf.Distancia),sizeof(int));
                Archivo.Read(&(Conf.Tiempo),sizeof(int));
                Archivo.Read(&(Conf.Unidad),sizeof(int));
                Archivo.Read(&(Conf.Max),sizeof(int));
                Archivo.Read(&(Conf.Timer),sizeof(float));
```

```
                Archivo.Read(&(Conf.Orden),sizeof(int));
                Archivo.Read(&(Conf.Poli_Med),sizeof(int));
                Archivo.Read(&(Conf.Poli_Ref),sizeof(int));
                Archivo.Close();
        }
        else{
                Conf.Conc_Unidad=       0;
                Conf.Distancia =        1;
                Conf.Tiempo             =       100;
                Conf.Unidad             =       1;
                Conf.Max                =       10;
                Conf.Timer              =       60;
                Conf.Orden              =       0;
                Conf.Poli_Med   =       5;
                Conf.Poli_Ref   =       2;
        }

//      <-- Crea los meses para leer fechas   -->
        wxDateTime now = wxDateTime::Now();
        now.SetMonth(now.Jan); Meses.Ene        =       now;
        now.SetMonth(now.Feb); Meses.Feb        =       now;
        now.SetMonth(now.Mar); Meses.Mar        =       now;
        now.SetMonth(now.Apr); Meses.Abr        =       now;
        now.SetMonth(now.Jun); Meses.Jun        =       now;
        now.SetMonth(now.Jul); Meses.Jul        =       now;
        now.SetMonth(now.Aug); Meses.Ago        =       now;
        now.SetMonth(now.Sep); Meses.Sep        =       now;
        now.SetMonth(now.Oct); Meses.Oct        =       now;
        now.SetMonth(now.Nov); Meses.Nov        =       now;
        now.SetMonth(now.Dec); Meses.Dic        =       now;
//      <--                                                             -->

//      <-- Carga por defecto el NO2 como gas a analizar     -->
        partida =       0;
        Menu_NO2(wxCommandEvent(0,0));

//      <-- Crea el temporizador      -->
        Temporizador    =       new wxTimer(this,ID_TIMER);
        Temporizador->Stop();

        Timer_Espectro =        new wxTimer(this,ID_TIMER_ESPECTRO);
        Timer_Espectro->Start(500);
        cEspectro->Set_factor(1);

//      <-- El sizer es la ventana principal  -->
        SetAutoLayout( TRUE );
        SetSizer( topsizer );

        if( !wxDirExists("Mediciones") )        //      Si no existe la carpeta mediciones,
                wxMkdir("Mediciones");          // la crea

        numHistoricos  = 0;                     // No hay históricos cargados
        Curva_Historicos= NULL;
        Plot_Historicos=NULL;
}

MyFrame::~MyFrame()
{
        wrapper.closeAllSpectrometers();
        wxFile Archivo;
        if(Archivo.Exists("DOAS.cnf"))
                Archivo.Open("DOAS.cnf", wxFile::write);     // 0 = read; 1 = write;
        else
                Archivo.Create("DOAS.cnf");
        if(Archivo.IsOpened())                  // Guarda los datos de configuración
        {
                Archivo.Write(&(Conf.Conc_Unidad),sizeof(int));
                Archivo.Write(&(Conf.Distancia),sizeof(int));
                Archivo.Write(&(Conf.Tiempo),sizeof(int));
                Archivo.Write(&(Conf.Unidad),sizeof(int));
                Archivo.Write(&(Conf.Max),sizeof(int));
                Archivo.Write(&(Conf.Timer),sizeof(float));
                Archivo.Write(&(Conf.Orden),sizeof(int));
```

```
                Archivo.Write(&(Conf.Poli_Med),sizeof(int));
                Archivo.Write(&(Conf.Poli_Ref),sizeof(int));
                Archivo.Close();
        }
}

void MyFrame::CreateGUIControls()
{
        WxMenuBar1 = new wxMenuBar();
        wxMenu *ID_MNU_ARCHIVO_1002_Mnu_Obj = new wxMenu(0);
        ID_MNU_ARCHIVO_1002_Mnu_Obj->Append(ID_MNU_ABRIR_1003, wxT("Abrir\tCtrl+A"),
                                            wxT("Abre un archivo"), wxITEM_NORMAL);
        ID_MNU_ARCHIVO_1002_Mnu_Obj->Append(ID_MNU_ABRIR_HIST, wxT("Abrir Historial\tCtrl+H"),
                                            wxT("Abre un archivo"), wxITEM_NORMAL);
        ID_MNU_ARCHIVO_1002_Mnu_Obj->Append(ID_MNU_CIERRA_PAG, wxT("Cerrar Pestaña\tCtrl+J"),
                                            wxT("Cierra pestaña"), wxITEM_NORMAL);
        ID_MNU_ARCHIVO_1002_Mnu_Obj->Append(ID_MNU_GUARDA, wxT("Guardar\tCtrl+G"),
                                            wxT("Guarda el día"), wxITEM_NORMAL);
        ID_MNU_ARCHIVO_1002_Mnu_Obj->Append(ID_MNU_BORRA_HIST, wxT("Borra Historial"),
                                wxT("Borra los datos de la pestaña \'Historia\'"), wxITEM_NORMAL);
        ID_MNU_ARCHIVO_1002_Mnu_Obj->AppendSeparator();

        wxMenu *ID_MNU_CONTAMINANTE_1006_Mnu_Obj = new wxMenu(0);
        ID_MNU_CONTAMINANTE_1006_Mnu_Obj->Append(ID_MNU_CH4_1008, wxT("CH4"), wxT(""), wxITEM_RADIO);
        ID_MNU_CONTAMINANTE_1006_Mnu_Obj->Check(ID_MNU_CH4_1008,false);
        ID_MNU_CONTAMINANTE_1006_Mnu_Obj->Append(ID_MNU_NO2_1007, wxT("NO2"), wxT(""), wxITEM_RADIO);
        ID_MNU_CONTAMINANTE_1006_Mnu_Obj->Check(ID_MNU_NO2_1007,true);
        ID_MNU_CONTAMINANTE_1006_Mnu_Obj->Append(ID_MNU_SO2_1009, wxT("SO2"), wxT(""), wxITEM_RADIO);
        ID_MNU_CONTAMINANTE_1006_Mnu_Obj->Check(ID_MNU_SO2_1009,false);
        ID_MNU_ARCHIVO_1002_Mnu_Obj->Append(ID_MNU_CONTAMINANTE_1006,
                                wxT("Contaminante"), ID_MNU_CONTAMINANTE_1006_Mnu_Obj);
        ID_MNU_ARCHIVO_1002_Mnu_Obj->Append(ID_MNU_CONFIG, wxT("Configuración\tCtrl+C"),
                                wxT("Abre la ventana de configuración"),wxITEM_NORMAL);
        WxMenuBar1->Append(ID_MNU_ARCHIVO_1002_Mnu_Obj, wxT("&Archivo"));
        wxMenu *ID_MNU_CAPTURA_1011_Mnu_Obj = new wxMenu(0);
        ID_MNU_CAPTURA_1011_Mnu_Obj->Append(ID_MNU_CAPTURA_1011,_T("Captura del &Espectro\tCtrl+E"),
                                                _T("Testeo de palabra"),wxITEM_NORMAL);
        WxMenuBar1->Append(ID_MNU_CAPTURA_1011_Mnu_Obj,wxT("&Captura"));
        SetMenuBar(WxMenuBar1);

        //      <<--    Toolbar         -->>
        CreateToolBar(wxNO_BORDER | wxTB_FLAT | wxTB_HORIZONTAL);
        wxToolBar *toolbar     =       GetToolBar();

        wxImage::AddHandler( new wxPNGHandler );
        toolbar->AddTool(ID_MNU_CAPTURA_1011,_T("Captura Espectro"), wxBitmap("iconos/espectro.png",
                                wxBITMAP_TYPE_PNG),_T("Captura el Espectro"));       //Captura espectro
        toolbar->AddTool(ID_MNU_ABRIR_1003,_T("Abrir"),wxBitmap("iconos/abrir.png",
                                wxBITMAP_TYPE_PNG), _T("Abrir un archivo"));         //Abrir Archivo
        toolbar->AddSeparator();
        toolbar->AddTool(ID_MNU_ABRIR_HIST, _T("Historico"),wxBitmap("iconos/historico.png",
                        wxBITMAP_TYPE_PNG), _T("Abre historial de concentraciones"));//Abrir Historial
        toolbar->AddTool(      ID_MNU_CIERRA_PAG, _T("Cerrar"),wxBitmap("iconos/hist_cerrar.png",
                        wxBITMAP_TYPE_PNG), _T("Cierra una ventana de históricos"));//Cierra Historial
        toolbar->AddSeparator();
        toolbar->AddTool(ID_MNU_CONFIG, _T("Configuración"),wxBitmap("iconos/config.png",
                        wxBITMAP_TYPE_PNG), _T("Abre la ventana de configuración"));//Configuración
        toolbar->Realize();

        SetTitle(wxT("DOAS"));
        wxIcon Icono("iconos/cefop.ico",wxBITMAP_TYPE_ICO);
        SetIcon(Icono);
        SetIcon(wxNullIcon);
        SetSize(8,8,801,551);
        Center();
}
void MyFrame::OnClose(wxCloseEvent& event)
{
        Destroy();
}
void MyFrame::SetXlimits(wxInt32 x, wxInt32 y)
{
        xmin=x; xmax=y;
```

```
        ymin= 1;        ymax=-1;
}
void MyFrame::Carga_SeccEfic(const wxChar *nombre)
{
        Set_Unidad(Conf.Conc_Unidad);
        wxString texto,text2;
        FILE *data;
        data = fopen(nombre,"r");
        if (data == NULL)
        {
               texto.Printf(
                   _T("Error abriendo el fichero BASE.\n")
                        _T("No se pudo abrir el archivo.")
                                  _T("\n%s"),nombre
                );

    wxMessageBox(texto, _T("Error de Apertura"), wxOK | wxICON_INFORMATION , this);
    return;
        }

        // <<--  Crea ventana de configuración de sección a utilizar        -->>
        wxPropertySheetDialog Config;
        Config.Create( this,wxID_ANY, _T("Rango Espectral"),wxDefaultPosition, wxSize(300,150),
                              wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER);
        Config.CreateButtons(wxOK);
        wxPanel* panel = new wxPanel(Config.GetBookCtrl());
        wxBoxSizer     *base_secc     =       new wxBoxSizer(wxVERTICAL);
        wxBoxSizer     *rango_secc    =       new wxBoxSizer(wxHORIZONTAL);
        rango_secc->Add(new wxStaticText(panel,wxID_ANY,_T("Desde: ")),0,wxALL|wxALIGN_CENTER,5);
        wxSpinCtrl     *rango_min     =       new wxSpinCtrl(panel,wxID_ANY,wxEmptyString,
                                                      wxDefaultPosition,wxSize(60,wxDefaultCoord));
        rango_min->SetRange(200,700);
        rango_min->SetValue(getXmin());
        rango_secc->Add(rango_min,0,wxALL|wxALIGN_CENTER_VERTICAL,5);

        rango_secc->Add(20,5,0,wxALL,0);

        rango_secc->Add(new wxStaticText(panel,wxID_ANY,_T("Hasta: ")),0,wxALL|wxALIGN_CENTER,5);
        wxSpinCtrl     *rango_max     =       new wxSpinCtrl(panel,wxID_ANY,wxEmptyString,
                                                      wxDefaultPosition,wxSize(60,wxDefaultCoord));
        rango_max->SetRange(200,700);
        rango_max->SetValue(getXmax());
        rango_secc->Add(rango_max,0,wxALL|wxALIGN_CENTER_VERTICAL,5);

        base_secc->Add(rango_secc,0,wxALL|wxALIGN_CENTER_VERTICAL,5);
        panel->SetSizer(rango_secc);
        Config.GetBookCtrl()->AddPage(panel,"Selección de rango espectral");
        if (partida != 0)
        {
            int validar=0;
            do{
                if (validar != 0)
                       wxMessageBox(_T("El valor de inicio debe ser mayor al de término."));
                if(Config.ShowModal() == wxID_OK)
                       SetXlimits(wxInt32(rango_min->GetValue()),wxInt32(rango_max->GetValue()));
                       validar++;
            }while(getXmin()>getXmax());             // Carga la ventana de configuración del rango
        }
        else
                partida++;

    st_data *DATA=(st_data *)malloc(sizeof(st_data));
        double valores[2][4000];
        int i=0,delta=15;
        do{
                fscanf(data,"%lf\t%lf", &valores[0][i], &valores[1][i]);// Carga valores de archivo
                if( (valores[0][i]<xmin-delta && valores[0][i]!=0)
                      ||  (i>0 && valores[0][i]==valores[0][i-1]));
                else if(valores[0][i]==0)
                {
                    i--;
                    break;
                }
```

```
                else
                  i++;
        }while( i==0 || (valores[0][i-1]>0 && valores[0][i-1]<xmax+delta));


        DATA->largo    =       i;
        DATA->datos[0] =       (double *)malloc(sizeof(double)*DATA->largo);
        DATA->datos[1] =       (double *)malloc(sizeof(double)*DATA->largo);
        DATA->rapida   =       NULL;
        DATA->spline   =       NULL;
        fclose(data);                                   //Cierra el archivo abierto.

        int NUM_COEF=Conf.Poli_Ref;                     // Realiza ajuste polinomial
        double *coef;
        coef    = (double *)malloc(sizeof(double)*NUM_COEF);
        polynomialfit(DATA->largo, NUM_COEF,valores[0], valores[1],coef);

        int j,k;
        double Polinomio;
        for(k=0,Polinomio=0 ; k<NUM_COEF ; k++)
           Polinomio+=  coef[k]*pow(valores[0][0],k);
        DATA->min  =   double(valores[1][0])-Polinomio;
        DATA->max  =   double(valores[1][0])-Polinomio;

        for( j=0 ; j<i ; j++)
        {                                               // Resta valores del polinomio
        for(k=0,Polinomio=0 ; k<NUM_COEF ; k++)
           Polinomio+=  coef[k]*pow(valores[0][j],k);
             DATA->datos[0][j] = double(valores[0][j]);
             DATA->datos[1][j] = double(valores[1][j])-Polinomio;
             if( DATA->datos[1][j] < DATA->min)
          DATA->min = DATA->datos[1][j];
             if( DATA->datos[1][j] > DATA->max)
             DATA->max = DATA->datos[1][j];
        }


        DATA->spline = spline_cubic_set(   DATA->largo,DATA->datos[0],
                           DATA->datos[1],0,0,0,0 );        //Crea data para el spline
        archivos->Set_BASE( DATA );                         // Carga datos de la sección eficaz
        m_base->Set_Curva(*DATA);                           // Grafica la sección eficaz
        Curva->Borra_datos();                               // Borra datos de análisis anteriores
        double deltaY = (DATA->max - DATA->min)*0.03;       // Ajusta la ventana a la curva
        m_plot->Fit(getXmin()-delta,getXmax()+delta,
                DATA->min-deltaY,DATA->max+deltaY,NULL,NULL);
        m_plot->UpdateAll();
        free(DATA);
}
void MyFrame::Abrir(wxCommandEvent& event)
{
        int i,nCont, Fecha_Leida;
        wxString texto,text2,text1;
        wxFileDialog fileDialog(this,wxT("Abre Archivos"),wxT(""),wxT(""),
                wxT("Cualquier Archivo (*)|*"),wxOPEN | wxFILE_MUST_EXIST | wxFD_MULTIPLE);

    if (fileDialog.ShowModal() == wxID_OK)
        {
                wxArrayString dir_archivo, nombre_archivo;
                fileDialog.GetPaths(dir_archivo);
                fileDialog.GetFilenames(nombre_archivo);
                size_t nArchivos = dir_archivo.GetCount();
                wxDateTime     Fecha_Ahora = wxDateTime::Now();
                for ( nCont=0 ; nCont< (int) nArchivos ; nCont++)
                {
                    texto.Printf(_T("Archivo a abrir:\n"));
                    texto.Append(dir_archivo[nCont].c_str());
                    FILE *data;
                    data = fopen(dir_archivo[nCont].c_str(),"r");
                    if (data == NULL)
                    { texto.Printf(_T("No se pudo abrir el archivo.")       );
                      wxMessageBox(texto, _T("Error de Apertura"), wxOK | wxICON_INFORMATION , this);
                      return;  }
                }
```

```
    st_data *DATA=(st_data *)malloc(sizeof(st_data));
    Fecha_Leida       =      0;             // Determina si se leyó la fecha
wxChar temp[100];
do{
    fgets(temp, 200, data);
        if(temp[0]=='D' && temp[2]=='t')                    // Lee la fecha
        {
                Fecha_Leida=1;
                wxString Fecha;
                switch(temp[10])                    //Fija el mes del  año
                {
                case 'J':
                  if(temp[11]=='a')             DATA->Fecha = Meses.Ene;
                  else if(temp[12]=='n') DATA->Fecha = Meses.Jun;
                  else                                      DATA->Fecha = Meses.Jul;
                  break;
                case 'F':                                   DATA->Fecha = Meses.Feb;
                  break;
                case 'M':
                  if(temp[12]=='r')             DATA->Fecha = Meses.Mar;
                  else                                      DATA->Fecha = Meses.May;
                  break;
                case 'A':
                  if(temp[11]=='p')             DATA->Fecha = Meses.Abr;
                  else                                      DATA->Fecha = Meses.Ago;
                  break;
                case 'S':                                   DATA->Fecha = Meses.Sep;
                  break;
                case 'O':                                   DATA->Fecha = Meses.Oct;
                  break;
                case 'N':                                   DATA->Fecha = Meses.Nov;
                  break;
                case 'D':                                   DATA->Fecha = Meses.Dic;
                  break;
                }

                Fecha.Printf("%c%c",temp[14],temp[15]);
                i = atoi(Fecha.c_str());
                DATA->Fecha.SetDay( i );            // Carga el día del mes
                Fecha.Printf("%c%c",temp[17],temp[18]);
                i = atoi(Fecha.c_str());            // Carga la hora de medición
                DATA->Fecha.SetHour( i );

                Fecha.Printf("%c%c",temp[20],temp[21]);
                i = atoi(Fecha.c_str());                    // Carga el minuto
                DATA->Fecha.SetMinute( i );

                Fecha.Printf("%c%c",temp[23],temp[24]);
                i = atoi(Fecha.c_str());                    // Carga el segundo
                DATA->Fecha.SetSecond( i );

                Fecha.Printf("%c%c%c%c",temp[30],temp[31],temp[32],temp[33]);
                i = atoi(Fecha.c_str());                    // Carga el año
                DATA->Fecha.SetYear( i );
        }
}while(temp[0]!= '>');
float valores[2][4000];
i=-1;
do{
        if( !(i>0 && valores[0][i] <= valores[0][i-1]) )
          i++;
        valores[0][i]=0;
        fscanf(data,"%f\t%f", &valores[0][i], &valores[1][i]);
                if(i%3645 == 0)
                        texto.Clear();
}while( valores[0][i]!= 0 && i<4000-1);      // Lee los datos de la medición guardada

        if( !Fecha_Leida )              // Si no leyó la fecha, la fija
                DATA->Fecha     =       Fecha_Ahora + wxTimeSpan::Seconds(nCont);

DATA->largo     =       i;              // Carga los datos del archivo leído
DATA->datos[0] =        (double *)malloc(sizeof(double)*DATA->largo);
DATA->datos[1] =        (double *)malloc(sizeof(double)*DATA->largo);
```

```
                DATA->rapida    =       NULL;
                DATA->spline    =       NULL;
                fclose(data);                           //Cierra el archivo abierto.

                int j;                                          // Busca los valores mínimos
                DATA->min  =   double(valores[1][0]);           // y máximos de los datos
                DATA->max  =   double(valores[1][0]);
                for( j=0 ; j<i ; j++)
                {
                        DATA->datos[0][j] = double(valores[0][j]);
                        DATA->datos[1][j] = double(valores[1][j]);
                        if( DATA->datos[1][j] < DATA->min)
                                        DATA->min = DATA->datos[1][j];
                        if( DATA->datos[1][j] > DATA->max)
                                DATA->max = DATA->datos[1][j];
                }

                Calculo_LMS(DATA, nCont==(nArchivos-1));     // Calcula el valor de concentración
                texto.Printf("%s ~%s~ C:% 6.3lf [ug/m^3 N]\n",
                        DATA->Fecha.Format(),nombre_archivo[nCont].c_str(),
                        DATA->C*(Peso_Atom*1e12)/(100*Conf.Distancia*NUM_AVOGADRO*1.1772));
                        Alarma->AppendText(texto);    // Imprime fecha y concentración calculada
                }
    }
}
void MyFrame::Calculo_LMS(st_data* DATA, bool fin)
{
        //      Ajuste por Mínimos cuadrados    LMS
        int i,j,min_error;
        double xinicio  =   getXmin();                      // Inicio del rango de longitud de onda
        double xfinal   =   getXmax();                      // Fin del rango
        int delta=0;
        for( i=0 ; DATA->datos[0][i] <= xinicio-delta ; i++); // Guarda desde inicio de datos
        int i_ini   =   i;
        for( j=i ; DATA->datos[0][j] <  xfinal+delta  ; j++);   // Guarda hasta fin de datos
        int i_fin   =   j;
        int largo_dato  =   i_fin-i_ini;
        double x1=0,x2=0,y1=0,xy=0,n,den,m,b,y2;
        double minimo=1E10;
        for( i=i_ini ; i<i_fin  ;   i++)
        {
                if( DATA->datos[1][i]<minimo)
                        minimo = DATA->datos[1][i];
        }

        double *Valores[2], *Polinomio, *Rapida, *Base;
        Valores[0]      =   (double *)malloc(sizeof(double)*largo_dato);     // Longitud de Onda
        Valores[1]      =   (double *)malloc(sizeof(double)*largo_dato);     // Intensidad de Luz
        Polinomio       =   (double *)malloc(sizeof(double)*largo_dato);     // Polinomio evaluado
        Rapida          =       (double *)malloc(sizeof(double)*largo_dato); // Valores parte rápida
        Base            =       (double *)malloc(sizeof(double)*largo_dato); // Valores parte Base
        for( i=i_ini ; i<i_fin  ;   i++)        {
                Valores[0][i-i_ini]  =   DATA->datos[0][i];
                Valores[1][i-i_ini]  =   DATA->datos[1][i];
        }

        int NUM_COEF=Conf.Poli_Med;                 // Selecciona el orden del polinomio
    double *coef;
    coef    = (double *)malloc(sizeof(double)*NUM_COEF);
    polynomialfit(    largo_dato, NUM_COEF,Valores[0], Valores[1],coef);
        for( i=0 ; i<largo_dato  ;   i++)
        {       for(j=0,Polinomio[i]=0 ; j<NUM_COEF ; j++)
                Polinomio[i]    +=  coef[j]*pow(Valores[0][i],j);
                Rapida[i]       =   log(Polinomio[i]/Valores[1][i]);  // Se obtiene la parte rápida
        }                                                             //de la medición
        st_data BASE = archivos->Get_BASE();
        double ypval, yppval;

        int ret;
        double coef2[2];
        double corrimiento=-1,corr_old=1000, opts[LM_OPTS_SZ], info[LM_INFO_SZ],opcion = 1E-15;
        opts[0]=opcion;               //      Tau             = Factor de escala para Mu inicial
        opts[1]=1E-20;                //      Epsylon1= Límite de detención para ||J^T e||_inf
```

```
opts[2]=1E-20;                  //      Epsylon2= Límite de detención para ||Dp||_2
opts[3]=1E-30;                  //      Epsylon3= Límite de detención para ||e||_2
opts[4]=-1;                     //      Delta   = Paso de la aproximación del Jacobiano
st_LevMar LevMar_DATA;
LevMar_DATA.Lambda     =       Valores[0];
LevMar_DATA.Espectro=  Rapida;
LevMar_DATA.stBASE     =       BASE;
LevMar_DATA.Orden      =       Conf.Orden;

while (abs(corrimiento-corr_old)>0.01)              // Realiza corrección por
{                                                   //desplazamiento de longitud de onda
        ret = -1;
        corr_old=corrimiento;
        for( i=0 ; i<largo_dato ; i++)
                Base[i] =spline_cubic_val(BASE.largo, BASE.datos[0],Valores[0][i]+corrimiento,
                                BASE.datos[1], BASE.spline, &ypval, &yppval );
                //Carga valores de sección eficaz de referencia con ajuste de corrimiento
        polynomialfit( largo_dato, 2,Base, Rapida,coef2);
        m=coef2[1];
        DATA->C         =       m;
        LevMar_DATA.coef        =       m;
        while(ret==-1)                  // Optimización mediante Levenberg-Marquard
        {       // Búsqueda de valor de corrimiento
                ret =   dlevmar_dif(    Ajuste_Lambda2,         // Función a cargar
                                        &corrimiento,           // Parámetro a ajustar
                                        NULL,                   // Valores deseados de la función
                                        1,                      // Dimensión de los parámetros
                                        largo_dato,             // Dimensión de valores deseados
                                        1000,                   // Número máximo de iteraciones
                                        opts,                   // Opciones
                                        info,                   // Vector de retorno información
                                        NULL,NULL,
                                        &LevMar_DATA);          //estructura con datos a función
                switch((int) info[6])
                {
                case 5:
                        corrimiento=0;
                case 4:
                        opts[0]*=2;
                        ret = -1;
                        break;
                default:
                        opts[0]=opcion;
                        break;
                }
        }               }
if (DATA->C*Factor_Unidad > Conf.Max)
{
    wxString texto;
    texto.Printf(_T("\tMedición de fecha \"%s\" tiene una concentración mayor al máximo "
                    "\n\t\tConcentración: %f \tMáximo: %i\n"),
                    DATA->Fecha.Format(),DATA->C*Factor_Unidad,Conf.Max);
                    Alarma->AppendText(texto);
}

archivos->Add_DATA(DATA,strBASE, Historia);  // Agrega concentración al histórico diario
free(DATA->datos[0]); DATA->datos[0] = NULL;
free(DATA->datos[1]); DATA->datos[1] = NULL;
Historia->Add_Curva((float) ( (DATA->Fecha.GetDayOfYear()-1)*24*60*60
                    + DATA->Fecha.GetHour()*60*60 + DATA->Fecha.GetMinute()*60 +
                    DATA->Fecha.GetSecond()),(float) DATA->C);//Actualiza curva a graficar
double deltaY;
ymin    =       Historia->Get_MinY();
ymax    =       Historia->Get_MaxY();
deltaY = (ymax - ymin)*0.005;
Hist_plot->Fit(         Historia->Get_MinX(),Historia->Get_MaxX(),
                        ymin - deltaY,ymax + deltaY,NULL,NULL);// Ajusta gráfica a la curva
Hist_plot->UpdateAll();

if(fin)                                 // Si es el último dato
{                                       // Actualiza curvas para comparación del ajuste
        st_data DATA2  = *DATA;
        DATA2.largo             = largo_dato;
```

```
                DATA2.datos[0] = (double *)malloc(sizeof(double)*largo_dato);
                for(i=0 ; i<largo_dato; i++)
                        DATA2.datos[0][i]       =       Valores[0][i]+corrimiento;
                DATA2.datos[1] = Rapida;
                double minY,maxY;

                DATA2.datos[1][0] /=m;                  // Para comparación con referencia
                for(i=1, minY=DATA2.datos[1][0], maxY=minY; i<DATA2.largo ; i++)
                {
                        DATA2.datos[1][i] /= m;
                        if(DATA2.datos[1][i] < minY)
                                minY = DATA2.datos[1][i];
                        if(DATA2.datos[1][i] > maxY)
                                maxY = DATA2.datos[1][i];
                }
                Curva->Set_Curva(DATA2);
                deltaY = (maxY - minY)*0.03;
                m_plot->Fit(getXmin()-1+corrimiento,getXmax()+1+corrimiento,
                                minY-deltaY,maxY+deltaY,NULL,NULL);
                m_plot->UpdateAll();
                free(DATA2.datos[0]);

                DATA2.datos[0] = Valores[0];
                DATA2.datos[1] = Valores[1];
                for(i=1, minY=DATA2.datos[1][0], maxY=minY; i<DATA2.largo ; i++)
                {
                        if(DATA2.datos[1][i] < minY)
                                minY = DATA2.datos[1][i];
                        if(DATA2.datos[1][i] > maxY)
                                maxY = DATA2.datos[1][i];
                }
                cDatos->Set_Curva(DATA2);       // Comparación con polinomio
                DATA2.datos[1] = Polinomio;
                cPoli->Set_Curva(DATA2);
                deltaY = (maxY - minY)*0.03;
                m_plotFit->Fit(getXmin()-1,   getXmax()+1,minY-deltaY,maxY+deltaY,NULL,NULL);
                m_plotFit->UpdateAll();
        }
        free(Polinomio);
        free(Valores[0]);
        free(Valores[1]);
        free(Rapida);
        free(coef);
        free(Base);
}
void MyFrame::Menu_NO2(wxCommandEvent& event)
{
        strBASE.Printf(_T("NO2_"));
        SetXlimits(wxInt32(NO2_MIN),wxInt32(NO2_MAX));
        Peso_Atom       =       NO2_PA;
        strREF.Printf(_T(NO2_BASE));
        Carga_SeccEfic(strREF.c_str());
}

void MyFrame::Menu_CH4(wxCommandEvent& event)
{
        strBASE.Printf(_T("CH4_"));
        SetXlimits(wxInt32(CH4_MIN),wxInt32(CH4_MAX));
        Peso_Atom       =       CH4_PA;
    strREF.Printf(_T(CH4_BASE));
        Carga_SeccEfic(strREF.c_str());
}

void MyFrame::Menu_SO2(wxCommandEvent& event)
{
        strBASE.Printf(_T("SO2_"));
        SetXlimits(wxInt32(SO2_MIN),wxInt32(SO2_MAX));
        Peso_Atom       =       SO2_PA;
        strREF.Printf(_T(SO2_BASE));
        Carga_SeccEfic(strREF.c_str());
}
void MyFrame::Borra_Hist(wxCommandEvent& event)
{
```

```
        archivos->Borra();                                      // Borra concentraciones guardadas
        Historia->Borra_datos(); Hist_plot->UpdateAll();        // Actualiza gráficas
        cPoli->Borra_datos(); cDatos->Borra_datos(); m_plotFit->UpdateAll();
        Curva->Borra_datos();
        int delta = 15;
        double deltaY = (m_base->Get_MaxY() - m_base->Get_MinY())*0.03;
        m_plot->Fit(getXmin()-delta,getXmax()+delta,m_base->Get_MinY()-deltaY,
                              m_base->Get_MaxY()+deltaY,NULL,NULL);
        m_plot->UpdateAll();
        Alarma->Clear();                                // Borra el panel de informaciones y alarmas
}

void MyFrame::Test_Espectro(wxCommandEvent& event)
{
        wxString texto;
        int numberOfSpectrometersAttached;
        numberOfSpectrometersAttached = wrapper.openAllSpectrometers();
        st_data *DATA  = crea_st();
        if(numberOfSpectrometersAttached == 0)
        {
                texto.Append(_T("\n\n\tNO HAY ESPECTRÓMETRO CONECTADO."));
                wxMessageBox(texto, _T("Test Espectrómetro"), wxOK | wxICON_INFORMATION , this);
                return;
        }
        int spectrometerIndex= 0;
        DoubleArray spectrumArray =   wrapper.getSpectrum(spectrometerIndex);// Obtención de espectro
        DoubleArray wavelengthArray = wrapper.getWavelengths(spectrometerIndex);
        int     numberOfPixels =       spectrumArray.getLength();           //Número de datos

        int i = 0;
        DATA->datos[0] =       (double *)malloc(sizeof(double)*numberOfPixels);
        DATA->datos[1] =       (double *)malloc(sizeof(double)*numberOfPixels);
        double *X,*Y;
        X       =       wavelengthArray.getDoubleValues();
        Y       =       spectrumArray.getDoubleValues();
        DATA->min  =   X[0];
        DATA->max  =   Y[0];
        for( i=0 ; i<numberOfPixels ; i++ )
        {
                DATA->datos[0][i]      =       X[i];   // Guarda las mediciones en la estructura
                DATA->datos[1][i]      =       Y[i];

                if( DATA->datos[1][i] < DATA->min)
            DATA->min = DATA->datos[1][i];
                if( DATA->datos[1][i] > DATA->max)
                DATA->max = DATA->datos[1][i];
        }
        DATA->largo    = numberOfPixels;
        Guarda_medicion(*DATA);                                 // Guarda los datos de la medición
        Calculo_LMS(DATA,true);                                 // Calcula la concentración
}
void MyFrame::Guarda(wxCommandEvent& event)
{
        wxString texto;                                         // Guarda el historial del día
        switch(archivos->Guarda(strBASE))
        {
        case 0: texto.Printf("Guardado\n"); break;
        case 1: texto.Printf("No se pudo abrir el archivo para escribir.\n"); break;
        case 2: texto.Printf("No se pudo crear el directorio \"Históricos\"\n"); break;
        default:texto.Printf("Error al guardar\n");
        }
        Alarma->AppendText(texto);
}
void MyFrame::Abrir_Hist(wxCommandEvent& event)
{
        wxString texto;
    wxFileDialog fileDialog(this,wxT("Abre Archivo"),wxT(""),wxT(""),
                            wxT("Todos los archivos (*)|*"),wxOPEN | wxFILE_MUST_EXIST);
    if (fileDialog.ShowModal() == wxID_OK)                      // Abre ventana de carga de archivos
    {
                texto.Printf(_T("Archivo a abrir:"));
                texto = fileDialog.GetPath();
                texto.Append(fileDialog.GetPath().c_str());
```

```
FILE *data;
data = fopen(fileDialog.GetPath().c_str(),"r");

if (data == NULL)                               // Si no se puede abrir el archivo
{       texto.Printf(_T("No se pudo abrir el archivo."));
        wxMessageBox(texto, _T("Error de Apertura"),
                wxOK | wxICON_INFORMATION , this);
        return;                                 // Aparece mensaje de error
}

int dia, mes, anio, mediciones;
int hora, min, seg;
float concentracion;
wxChar temp[100];
do{
        fgets(temp, 200, data);
        if(temp[0]=='F' && temp[1]=='e')        // Carga la fecha desde el archivo
        {
                wxString Fecha;
                Fecha.Printf("%c%c",temp[19],temp[20]);
                dia = atoi(Fecha.c_str());

                Fecha.Printf("%c%c",temp[22],temp[23]);
                mes = atoi(Fecha.c_str());

                Fecha.Printf("%c%c%c%c",temp[25],temp[26], temp[27], temp[28]);
                anio = atoi(Fecha.c_str());
        }
        else if(temp[0]=='N' && temp[2]=='m')
                mediciones = atoi(temp+32);
}while(temp[0]!= '>');

//<--           Crea la página donde dibujar  -->
mpWindow *Panel = Crea_Pag(Libro, fileDialog.GetFilename(),mpX_TIME,
                                _T("Hora (hh:mm:ss)"),_T("Unidad"));
mpCurva *Curva = new mpCurva(wxString(_T("History")),mpALIGN_NE);
Panel->AddLayer(Curva);
Panel->EnableDoubleBuffer(true);
Panel->SetMPScrollbars(true);
Curva->Borra_datos();
wxPen Lapiz(*wxRED, 1, wxSOLID);
Curva->SetPen(Lapiz);
Curva->SetContinuity(true);
//<--           Página creada          -->

int i;
for( i=0 ; i<mediciones ; i++)
{
        fscanf(data,"%i\t%f\n",&seg,&concentracion);
        Curva->Add_Curva( (float) seg, concentracion);
}
fclose(data);

double histYmin = Curva->Get_MinY();
double histYmax = Curva->Get_MaxY();
double deltaY = (histYmax - histYmin)*0.03;
if (deltaY !=0)
        Panel->Fit(    Curva->Get_MinX(),Curva->Get_MaxX(),histYmin - deltaY,
                                histYmax + deltaY,NULL,NULL);
else
        Panel->Fit();                   //Ajusta el espacio a las curvas.

//      <--             Agrega la Curva al listado de Curvas (para el cambio de unidades)   -->
numHistoricos++;
mpCurva **newHist = (mpCurva **)malloc(sizeof(mpCurva *)*numHistoricos);
for( i=0 ; i<numHistoricos-1 ; i++)
        newHist[i]      =       Curva_Historicos[i];
newHist[i]      =       Curva;
free(Curva_Historicos);
Curva_Historicos = newHist;

mpWindow **newWindow = (mpWindow **)malloc(sizeof(mpWindow *)*numHistoricos);
```

```
                for( i=0 ; i<numHistoricos-1 ; i++)
                        newWindow[i]    =       Plot_Historicos[i];
                newWindow[i]    =       Panel;
                free(Plot_Historicos);
                Plot_Historicos = newWindow;
                Set_Unidad(Conf.Conc_Unidad);
        }
}
void MyFrame::Cierra_Pagina(wxCommandEvent& event)
{
        int paginas = 4;                                // Número de páginas no borrables
        int pag_actual =        Libro->GetSelection(); // Página actual que se desea borrar
        if(pag_actual < paginas)                        // Si pág a borrar no es de histórico
                wxMessageBox(_T("Sólo puede cerrar una pestaña de históricos.\n"
                        "No puede cerrar la pestaña de mediciones actuales."));// Mensaje de error
        else
        {                       // Si sí se puede borrar, ajusta listado de páginas de históricos
                mpCurva **newHist       = (mpCurva **) malloc(sizeof(mpCurva *)*numHistoricos-1);
                mpWindow **newWindow= (mpWindow **)malloc(sizeof(mpWindow*)*numHistoricos-1);

                for(int i=0, j=0 ; i<numHistoricos ; i++)    // Busca y borra la página actual
                {
                        if( i !=pag_actual-paginas )
                        {
                                newHist[j]      =       Curva_Historicos[i];
                                newWindow[j]=   Plot_Historicos[i];
                                j++;
                        }
                }
                free(Curva_Historicos); Curva_Historicos=newHist;    // Actualiza las listas
                free(Plot_Historicos);   Plot_Historicos=newWindow;
                numHistoricos--;
                Libro->DeletePage(pag_actual);
        }
}
void MyFrame::Configuracion(wxCommandEvent& event)
{
        wxPropertySheetDialog Config;
        Config.Create(          this,wxID_ANY,_T("Configuracion"),wxDefaultPosition,
                wxDefaultSize,wxDEFAULT_DIALOG_STYLE|wxRESIZE_BORDER);
        Config.CreateButtons(wxOK | wxCANCEL);          // Crea ventana de configuración

//      < --                            Ventana "General"                                -- >
        wxPanel* panel = new wxPanel(Config.GetBookCtrl());
        wxBoxSizer      *conf1_base=    new wxBoxSizer(wxVERTICAL);
        wxStaticBox     *conf1_1box=    new wxStaticBox(panel,wxID_ANY,_T("Configuración Programa:"));
        wxBoxSizer      *conf1_1box_base=new wxStaticBoxSizer(conf1_1box,wxVERTICAL);
                wxBoxSizer      *conf1_1item    =       new wxBoxSizer(wxHORIZONTAL);
                conf1_1item->Add(new wxStaticText(panel,wxID_ANY,_T("Unidad de despliegue:\t")),
                                                        0,wxALL|wxALIGN_CENTER,5);
                conf1_1item->Add(15,5,0,wxALL,0);
                wxArrayString Concentracion_Unidad;
                Concentracion_Unidad.Add(_T("moleculas/cm^2"));
                Concentracion_Unidad.Add(_T("p.p.m."));
                Concentracion_Unidad.Add(_T("p.p.b."));
                Concentracion_Unidad.Add(_T("p.p.t."));
                Concentracion_Unidad.Add(_T("ug/m^3 N"));
                wxChoice*conf1_unidad= new xChoice(panel,wxID_ANY,wxDefaultPosition,
                                                        wxDefaultSize,Concentracion_Unidad);
                conf1_unidad->Select(Conf.Conc_Unidad);
                conf1_1item->Add(conf1_unidad,0,wxALL|wxALIGN_CENTER_VERTICAL,5);
        conf1_1box_base->Add(conf1_1item,wxALL|wxALIGN_CENTER_VERTICAL);

                wxBoxSizer      *conf1_2item    =       new wxBoxSizer(wxHORIZONTAL);
                conf1_2item->Add(new wxStaticText(panel,wxID_ANY,_T("Límite de concentración:\t")),
                                                        0,wxALL|wxALIGN_CENTER,5);
                conf1_2item->Add(5,5,0,wxALL,0);
                wxSpinCtrl *conf1_max=new wxSpinCtrl(panel,wxID_ANY,wxEmptyString,
                                                        wxDefaultPosition,wxSize(60,wxDefaultCoord));
                conf1_max->SetRange(0,1000);
                conf1_max->SetValue(Conf.Max);
                conf1_2item->Add(conf1_max,0,wxALL|wxALIGN_CENTER_VERTICAL,5);
```

```
        conf1_1box_base->Add(conf1_2item,wxALL|wxALIGN_CENTER_VERTICAL);
    conf1_base->Add(conf1_1box_base,wxALL|wxALIGN_CENTER_VERTICAL);

        wxBoxSizer     *conf1_3item   =       new wxBoxSizer(wxHORIZONTAL);
        conf1_3item->Add(      new wxStaticText(panel,wxID_ANY,_T("Distancia de Medición: ")),
                                              0, wxALL|wxALIGN_CENTER, 5);
        conf1_3item->Add(15,5,0,wxALL,0);
        wxSpinCtrl     *conf1_distancia=new wxSpinCtrl(panel,wxID_ANY,wxEmptyString,
                                              wxDefaultPosition,wxSize(60,wxDefaultCoord));
        conf1_distancia->SetRange(0,10000);
        conf1_distancia->SetValue(Conf.Distancia);
        conf1_3item->Add(conf1_distancia,0,wxALL|wxALIGN_CENTER_VERTICAL,5);
        conf1_3item->Add(new wxStaticText(panel,wxID_ANY,_T("metros.")),
                                              0, wxALL|wxALIGN_CENTER, 5);
    conf1_base->Add(conf1_3item,wxALL|wxALIGN_CENTER_VERTICAL);

    panel->SetSizer(conf1_base);
    Config.GetBookCtrl()->AddPage(panel, "General");
//  < --                        Cierre de Página: "General"          -- >

//  < --                        Ventana Espectro                              -- >
    panel = new wxPanel(Config.GetBookCtrl());
    wxBoxSizer     *conf2_base           =       new wxBoxSizer(wxVERTICAL);
    int Spec_andando = wrapper.openAllSpectrometers();
    wxSpinCtrl     *conf2_tiempo,*conf2_timer;
    wxChoice       *conf2_unidad;
    wxCheckBox     *conf2_timerST;
    if ( Spec_andando==0 )
        conf2_base->Add(new wxStaticText(panel,wxID_ANY,
            _T("No hay espectrómetro conectado")),0,wxALL|wxALIGN_CENTER_VERTICAL,5);
    else{
        wxStaticBox *conf2_1box =new wxStaticBox(panel,wxID_ANY,
                    _T("Configuración Espectro:"),wxDefaultPosition,wxDefaultSize);
        wxBoxSizer     *conf2_1box_base=new wxStaticBoxSizer(conf2_1box,wxVERTICAL);
        wxBoxSizer     *conf2_1item   =new wxBoxSizer(wxHORIZONTAL);
        conf2_1item->Add(      new wxStaticText(panel,wxID_ANY,_T("Tiempo de Integración:")),
                                              0,wxALL|wxALIGN_CENTER_VERTICAL,5);
        conf2_1item->Add(5,5,0,wxALL,0);

        conf2_tiempo=new wxSpinCtrl(panel,wxID_ANY,wxEmptyString,
                                wxDefaultPosition,wxSize(60,wxDefaultCoord));
        conf2_tiempo->SetRange(0,999);
        conf2_tiempo->SetValue(Conf.Tiempo);
        conf2_1item->Add(conf2_tiempo,0,wxALL|wxALIGN_CENTER_VERTICAL      , 5);

        conf2_1item->Add(5,5,0,wxALL,0);

        wxArrayString Tiempo_Unidad;
        Tiempo_Unidad.Add(_T("seg"));
        Tiempo_Unidad.Add(_T("ms"));
        Tiempo_Unidad.Add(_T("us"));
        conf2_unidad   =       new wxChoice(panel,wxID_ANY,wxDefaultPosition,
                                wxDefaultSize,Tiempo_Unidad);
        conf2_unidad->Select(Conf.Unidad);
        conf2_1item->Add(conf2_unidad,0,wxALL|wxALIGN_CENTER_VERTICAL,5);
        conf2_1box_base->Add(conf2_1item,wxALL|wxALIGN_CENTER_VERTICAL);
        wxBoxSizer *conf2_2item = new wxBoxSizer(wxHORIZONTAL);
        conf2_timerST          =  new wxCheckBox(panel,wxID_ANY,_T("Activar Temporizador: "));
        conf2_timerST->SetValue(Temporizador->IsRunning());
        conf2_2item->Add(conf2_timerST,0,wxALL|wxALIGN_CENTER_VERTICAL,5);
        conf2_2item->Add(5,5,0,wxALL,0);

        conf2_timer    =new wxSpinCtrl(panel,wxID_ANY,wxEmptyString,
                                wxDefaultPosition,wxSize(60,wxDefaultCoord));
        conf2_timer->SetRange(2,86400);
        conf2_timer->SetValue(Conf.Timer);
        conf2_2item->Add(conf2_timer,0,wxALL|wxALIGN_CENTER_VERTICAL, 5);
        conf2_2item->Add(2,5,0,wxALL,0);
        conf2_2item->Add(new wxStaticText(panel,wxID_ANY,_T("seg.")),0,
                                wxALL|wxALIGN_CENTER_VERTICAL, 5);
        conf2_1box_base->Add(conf2_2item,wxALL|wxALIGN_CENTER_VERTICAL);
        conf2_base->Add(conf2_1box_base,wxALL|wxALIGN_CENTER_VERTICAL);
    }
```

```
        panel->SetSizer(conf2_base);
    Config.GetBookCtrl()->AddPage(panel, "Espectro");
//      < --                        Cierre de Página: "Espectro"  -- >

//      < --                        Ventana Parámetros de Algoritmo      -->
        panel = new wxPanel(Config.GetBookCtrl());
        wxBoxSizer     *conf3_base           =      new wxBoxSizer(wxVERTICAL);

        wxSpinCtrl     *conf3_poli_medicion,*conf3_poli_referencia;
        wxStaticBox    *conf3_1box = new wxStaticBox(panel,wxID_ANY,_T("Orden de polinomio de:"),
                                     wxDefaultPosition,wxDefaultSize);
        wxBoxSizer     *conf3_1box_base=new wxStaticBoxSizer(conf3_1box,wxVERTICAL);
        wxBoxSizer     *conf3_1item   = new wxBoxSizer(wxHORIZONTAL);
        conf3_1item->Add(     new wxStaticText(panel,wxID_ANY,_T("-el espectro medido:")),
                                     0,wxALL|wxALIGN_CENTER_VERTICAL,5);
        conf3_1item->Add(80,5,0,wxALL,0);
               conf3_poli_medicion    = new wxSpinCtrl(panel,wxID_ANY,wxEmptyString,
                                     wxDefaultPosition, wxSize(60,wxDefaultCoord));
               conf3_poli_medicion->SetRange(1,10);
               conf3_poli_medicion->SetValue(Conf.Poli_Med-1);
               conf3_1item->Add(conf3_poli_medicion,0,wxALL|wxALIGN_CENTER_VERTICAL, 5);

        conf3_1box_base->Add(conf3_1item,wxALL|wxALIGN_CENTER_VERTICAL);

        wxBoxSizer     *conf3_2item = new wxBoxSizer(wxHORIZONTAL);
        conf3_2item->Add(new wxStaticText(panel,wxID_ANY,
               _T("-la sección transversal de referencia:")),0,wxALL|wxALIGN_CENTER_VERTICAL,5);
        conf3_poli_referencia  = new wxSpinCtrl( panel,wxID_ANY,wxEmptyString,wxDefaultPosition,
                                     wxSize(60,wxDefaultCoord));
        conf3_poli_referencia->SetRange(1,10);
        conf3_poli_referencia->SetValue(Conf.Poli_Ref-1);
        conf3_2item->Add(conf3_poli_referencia,0,wxALL|wxALIGN_CENTER_VERTICAL, 5);
        conf3_1box_base->Add(conf3_2item,wxALL|wxALIGN_CENTER_VERTICAL);
        conf3_base->Add(conf3_1box_base,wxALL|wxALIGN_CENTER_VERTICAL);
        conf3_base->Add(5,5,0,wxALL,0);

        wxStaticBox *conf3_2box=new wxStaticBox(panel,wxID_ANY,
                     _T("Selección de rango espectral: "),wxDefaultPosition,wxDefaultSize);
        wxBoxSizer     *conf3_2box_base= new wxStaticBoxSizer(conf3_2box,wxVERTICAL);
        wxBoxSizer     *rango_secc = new wxBoxSizer(wxHORIZONTAL);
        rango_secc->Add(new wxStaticText(panel,wxID_ANY,_T("Desde: ")),0,wxALL|wxALIGN_CENTER,5);
        wxSpinCtrl     *rango_min            =      new wxSpinCtrl(
        panel,wxID_ANY,wxEmptyString,wxDefaultPosition,wxSize(60,wxDefaultCoord));
        rango_min->SetRange(200,700);
        rango_min->SetValue(getXmin());
        rango_secc->Add(rango_min,0,wxALL|wxALIGN_CENTER_VERTICAL,5);

        rango_secc->Add(25,5,0,wxALL,0);

        rango_secc->Add(new wxStaticText(panel,wxID_ANY,_T("Hasta: ")),0,wxALL|wxALIGN_CENTER,5);
        wxSpinCtrl     *rango_max     =new wxSpinCtrl(panel,wxID_ANY,wxEmptyString,
                                     wxDefaultPosition,wxSize(60,wxDefaultCoord));
        rango_max->SetRange(200,700);
        rango_max->SetValue(getXmax());
        rango_secc->Add(rango_max,0,wxALL|wxALIGN_CENTER_VERTICAL,5);
        conf3_2box_base->Add(rango_secc,wxALL|wxALIGN_CENTER_VERTICAL);
        conf3_base->Add(conf3_2box_base,wxALL|wxALIGN_CENTER_VERTICAL);

        panel->SetSizer(conf3_base);
        Config.GetBookCtrl()->AddPage(panel, "Parámetros de Algoritmo");
//      < --                        Cierre de Página: "Parámetros de Algoritmo"          -->


    Config.LayoutDialog();
        wxString texto,text2;
        switch(Config.ShowModal())
        {
        case wxID_OK:
               Conf.Conc_Unidad       =      conf1_unidad->GetSelection();
               Set_Unidad(Conf.Conc_Unidad);
               Conf.Distancia         =      conf1_distancia->GetValue();
               Conf.Max               =      conf1_max->GetValue();
               Conf.Poli_Med          =      conf3_poli_medicion->GetValue()+1;
```

```
                if (Conf.Poli_Ref != conf3_poli_referencia->GetValue()+1||
                        getXmin()       != rango_min->GetValue()          ||
                        getXmax()       != rango_max->GetValue())
                {
                        Conf.Poli_Ref = conf3_poli_referencia->GetValue()+1;
                        SetXlimits(rango_min->GetValue(), rango_max->GetValue());
                        partida = 0;
                        Carga_SeccEfic(strREF.c_str());
                }

                if(Spec_andando!=0)
                {
                        Conf.Unidad           =       conf2_unidad->GetSelection();
                        Conf.Tiempo           =       conf2_tiempo->GetValue();
                        Conf.Timer            =       conf2_timer->GetValue();
                        if(conf2_timerST->IsChecked())
                        {       if( !(Temporizador->IsRunning()) )
                                        Temporizador->Start(Conf.Timer*1000,false);                }
                        else
                        {       if( Temporizador->IsRunning() )
                                        Temporizador->Stop();                                      }
                        wrapper.setIntegrationTime(0, Conf.Tiempo*pow((float) 1000,
                                (int) 2-Conf.Unidad)); // Tiempo de integración en microsegundos (uS)
                }
                break;
        default:
                break;
        }
}
void MyFrame::Func_Temporizador(wxTimerEvent &event)
{
        Test_Espectro(wxCommandEvent());      // Mide con espectrómetro
}
void MyFrame::Func_Temp_Espectro(wxTimerEvent &event)
{
        Int numberOfSpectrometersAttached = wrapper.openAllSpectrometers();

        st_data DATA;
        DATA.rapida    =       NULL;

        if(numberOfSpectrometersAttached == 0)              // Si no hay espectrómetro
                return;                                     // Retorna

        int spectrometerIndex= 0;
        DoubleArray spectrumArray     =       wrapper.getSpectrum(spectrometerIndex);// Medición
        DoubleArray wavelengthArray   =       wrapper.getWavelengths(spectrometerIndex);
        int     numberOfPixels        =       spectrumArray.getLength();    //Número de datos

        int i = 0;
        DATA.datos[0]  =       (double *)malloc(sizeof(double)*numberOfPixels);
        DATA.datos[1]  =       (double *)malloc(sizeof(double)*numberOfPixels);
        double *X,*Y;
        X       =       wavelengthArray.getDoubleValues();
        Y       =       spectrumArray.getDoubleValues();

        DATA.min  =   X[0];
        DATA.max  =   Y[0];
        for( i=0 ; i<numberOfPixels ; i++ )
        {
                DATA.datos[0][i]       =       X[i];
                DATA.datos[1][i]       =       Y[i];
                if( DATA.datos[1][i] < DATA.min)
            DATA.min = DATA.datos[1][i];
                if( DATA.datos[1][i] > DATA.max)
                DATA.max = DATA.datos[1][i];
        }
        DATA.largo            =       numberOfPixels;
        cEspectro->Set_Curva(DATA);
        mEspectro->UpdateAll();
        free(DATA.datos[0]); DATA.datos[0] = NULL;
        free(DATA.datos[1]); DATA.datos[1] = NULL;
}
```

```
void MyFrame::Guarda_medicion(st_data Data)
{
	wxString Dir_old, Dir_nuevo;
	Dir_old		=	wxGetCwd();
	wxString texto;
	if( wxDirExists("Mediciones"))
	{
		Dir_nuevo	=	Dir_old;
		Dir_nuevo.Append("\\");
		Dir_nuevo.Append("Mediciones");		// crea carpeta de Mediciones
		wxSetWorkingDirectory(Dir_nuevo);
		wxString dia,temp;
		dia.Printf("%04i-%02i-%02i",Data.Fecha.GetYear(),Data.Fecha.GetMonth()+1,
				Data.Fecha.GetDay()); // Crea carpeta con la fecha como nombre
		if ( !wxDirExists( dia.c_str()) )
			wxMkdir(dia.c_str());
		Dir_nuevo.Append("\\");
		Dir_nuevo.Append(dia);
		wxSetWorkingDirectory(Dir_nuevo);		// cambia de directorio de trabajo

		dia.Printf("%02i.%02i.%02i.txt",Data.Fecha.GetHour(),
				Data.Fecha.GetMinute(),Data.Fecha.GetSecond());
		FILE *Archivo = fopen(dia.c_str(),"w");		// Crea el archivo a guardar
		if( Archivo != NULL)
		{
			texto.Printf("Espectro Medido\n++++++++++++++++++++++++++++++++++\nDate: ");
			switch(Data.Fecha.GetWeekDay()+1)
			{
				case 1: temp.Printf("Sun "); break;
				case 2: temp.Printf("Mon "); break;
				case 3: temp.Printf("Tue "); break;
				case 4: temp.Printf("Wed "); break;
				case 5: temp.Printf("Thu "); break;
				case 6: temp.Printf("Fri "); break;
				case 7: temp.Printf("Sat "); break;
				default: break;
			}
			texto.Append(temp);
			switch(Data.Fecha.GetMonth()+1)
			{
				case 1: temp.Printf("Jan "); break;
				case 2: temp.Printf("Feb "); break;
				case 3: temp.Printf("Mar "); break;
				case 4: temp.Printf("Apr "); break;
				case 5: temp.Printf("May "); break;
				case 6: temp.Printf("Jun "); break;
				case 7: temp.Printf("Jul "); break;
				case 8: temp.Printf("Aug "); break;
				case 9: temp.Printf("Sep "); break;
				case 10:temp.Printf("Oct "); break;
				case 11:temp.Printf("Nov "); break;
				case 12:temp.Printf("Dec "); break;
				default: break;
			}
			texto.Append(temp);
			temp.Printf("%2i %02i:%02i:%02i CET %04i\n",Data.Fecha.GetDay(),
				Data.Fecha.GetHour(), Data.Fecha.GetMinute(),
				Data.Fecha.GetSecond(), Data.Fecha.GetYear() );
			texto.Append(temp);				// Escribe la fecha

			temp.Printf("Number of Pixels in Processed Spectrum: %i\n", Data.largo);
			texto.Append(temp);				// Escribe el número de datos
			temp.Printf(">>>>>Begin Processed Spectral Data<<<<<\n");
			texto.Append(temp);				// Empieza a guardar los datos

			Alarma->AppendText(texto);
			int i;
			for (i=0 ; i<Data.largo ; i++)
			{
				temp.Printf("%.2f\t%.2f\n",Data.datos[0][i],Data.datos[1][i]);
				texto.Append(temp);
			}
			temp.Printf("0\t0\0");
```

```
                        texto.Append(temp);
                        fprintf(Archivo,texto.c_str());
                        fclose(Archivo);
                        wxSetWorkingDirectory(Dir_old);       // Restaura el directorio de trabajo
                }
                else
                        wxMessageBox(_T("No se pudo abrir el archivo para escribir"),
                                                _T("Error de escritura"), wxOK | wxICON_INFORMATION);
        }
}
void MyFrame::Set_Unidad(int Unidad)
{
        double factor=(Peso_Atom*1e6)/(100*Conf.Distancia*NUM_AVOGADRO);    // [g/m^3]
        double Temperatura     = 25,   Razon_Temperatura      =       ((Temperatura+273.15)/273.15);
        double Presion_atm     = 760,  Razon_Presion          =       760/Presion_atm;       //[mmHg]

        switch(Unidad)
        {
        case 1:         //p.p.m.        == [g/m^3]
                factor  =       factor*1e3*22.4136* Razon_Temperatura * Razon_Presion / (Peso_Atom);
                break;
        case 2:         //p.p.b         ==      [mg/m^3]
                factor  =       factor*1e6*22.4136* Razon_Temperatura * Razon_Presion / (Peso_Atom);
                break;
        case 3:         //p.p.t         ==      [ug/m^3]
                factor  =       factor*1e9*22.4136* Razon_Temperatura * Razon_Presion / (Peso_Atom);
                break;
        case 4:         // [ug/m^3 N]   =>      [molec/cm^2]/Num_avogadro [mol/molec] * Peso_atom
                factor  *=      (1e6/1.1772);
                break;
        default:
                factor = 1;
                break;
        }
        Historia->Set_factor(factor);                 // Actualiza las unidades de todas las curvas
        double ymin     =       Historia->Get_MinY();
        double ymax     =       Historia->Get_MaxY();
        double deltaY = (ymax - ymin)*0.005;
        Hist_plot->Fit( Historia->Get_MinX(),Historia->Get_MaxX(),ymin - deltaY,
                        ymax + deltaY,NULL,NULL);
        Hist_plot->UpdateAll();
        for(int i=0 ; i<numHistoricos ; i++)
        {
                Curva_Historicos[i]->Set_factor(factor);
                ymin    =       Curva_Historicos[i]->Get_MinY();
                ymax    =       Curva_Historicos[i]->Get_MaxY();
                deltaY  = (ymax - ymin)*0.005;
                Plot_Historicos[i]->Fit(Curva_Historicos[i]->Get_MinX(),
                        Curva_Historicos[i]->Get_MaxX(),ymin - deltaY,ymax + deltaY,NULL,NULL);
                Plot_Historicos[i]->UpdateAll();
        }
        Factor_Unidad  =       factor;
}
```

### A.2.4 PlotCurve.cpp

```
#include "PlotCurve.h"
void mpCurva::Set_Curva(st_data DataIn)
{                                                       // Fija los valores a graficar
        xs.clear();     xs.reserve(DataIn.largo);
        y_base.clear();y_base.reserve(DataIn.largo);
        ys.clear();     ys.reserve(DataIn.largo);
        MinX    =       DataIn.datos[0][0];
        MinY    =       DataIn.datos[1][0];
        MaxY    =       DataIn.datos[1][0];
        int i;
        for( i=0 ; i<DataIn.largo ; i++)
        {
                xs.push_back(DataIn.datos[0][i]);
                y_base.push_back(DataIn.datos[1][i]);
                ys.push_back(DataIn.datos[1][i]*factor);
```

```
                if( MinY > ys[i] ) MinY = ys[i];
                if( MaxY < ys[i] ) MaxY = ys[i];
        }
        MaxX    =       DataIn.datos[0][i];
        SetData(xs,ys);
}

void mpCurva::Add_Curva(float x, float y)
{                                               //      Agrega un par de datos para graficar
        xs.push_back(x);
        y_base.push_back(y);
        ys.push_back(y*factor);
        SetData(xs,ys);

        if(xs.size() == 1)
        {
                MinX    =       x;
                MaxX    =       x;
                MinY    =       ys[ys.size()-1];
                MaxY    =       ys[ys.size()-1];
        }
        else
        {
                if ( MinX > x )                 MinX = x;
                if ( MaxX < x )                 MaxX = x;
                if ( MinY > ys[ys.size()-1] ) MinY = ys[ys.size()-1];
                if ( MaxY < ys[ys.size()-1] ) MaxY = ys[ys.size()-1];
        }
}

void mpCurva::Set_factor(float Unidad)
{                                               // Actualiza unidad de despliegue
        factor = Unidad;
        ys.clear();
        int i;
        if (y_base.size() !=0)
        {
                MinY    =       y_base[0]*factor;
                MaxY    =       MinY;
        }
        for( i=0 ; i< xs.size() ; i++)
        {
                ys.push_back(y_base[i]*factor);
                if( MinY > ys[i] ) MinY = ys[i];
                if( MaxY < ys[i] ) MaxY = ys[i];
        }
        SetData(xs,ys);
}

void mpCurva::Borra_datos()
{                                               // Borra los datos de la curva
        y_base.clear();
        xs.clear();
        ys.clear();
        SetData(xs,ys);
}
```