Universidad de Alcalá
Escuela Politécnica Superior

Alejandro Moreno López

Trabajo Fin de Grado

VISUALIZADOR PYTHON PARA LAS MEDIDAS DEL MONITOR DE NEUTRONES DE CASTILLA-LA MANCHA(CaLMa)

Autor: Alejandro Moreno López

Tutor/es: Juan José Blanco Ávalos

2017/2018

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

Grado en Ingeniería Telemática

Trabajo Fin de Grado

VISUALIZADOR PYTHON PARA LAS MEDIDAS DEL MONITOR DE NEUTRONES DE CASTILLA-LA MANCHA(CaLMa)

Autor: Alejandro Moreno López

Tutor/es: Juan José Blanco Ávalos

TRIBUNAL:

Presidente: Raúl Gómez Herrero

Vocal 1º: Oscar García Población

Vocal 2º: Juan José Blanco Ávalos

FECHA: 20-02-2018

Índice

Resumen

En este TFG se busca realizar un programa para la visualización de datos del monitor de neutrones de Castilla La Mancha (CaLMa). Se partirá de un lenguaje de programación sin restricciones de licencias, en este caso Python y de módulos software libre para este lenguaje. Los objetivos son que disponga de una interfaz gráfica sencilla e intuitiva con la que realizar los cálculos, representaciones gráficas, guardar los datos en ASCII y las imágenes en jpeg, y además que se pueda conectar a internet para comparar con otras estaciones de monitores de neutrones.

Este trabajo de fin de grado ha sido desarrollado dentro del proyecto, "Observatorio de Rayos Cósmicos Antártico (ref: CTM2016-77325-C2-1-P)."

Palabras clave

Neutrones, Python, Representaciones, Castilla-La Mancha, Monitor

Abstract

The goal of this project is the design of a visualisation tool for the Castilla-La Mancha neutron monitor(CaLMa)'s data. Starting with a programming language without license restriction, in this case Python and free software libraries for Python. The project objectives are an easy use and intuitive graphical interface to make calculus, graphic representations, save data on ASCII format and images on JPEG. In addition, the software should have an internet connection to compare the data and graphics between others neutron monitor's stations.

This final proyect has been developed within the proyect, "Observatorio de Rayos Cósmicos Antártico (ref: CTM2016-77325-C2-1-P)."

Keywords

Neutrons, Python, Plots, Castilla-La Mancha, Monitor

Introducción

Un monitor de neutrones [1] es un detector diseñado para medir neutrones producidos por la interacción de rayos cósmicos con los elementos de la atmósfera. De forma histórica estas partículas son llamadas "rayos cósmicos". Los monitores de neutrones registran los rayos cósmicos y su variación debido al ciclo de manchas solares que dura 11 años y el ciclo magnético de 22 años.

Los monitores de neutrones fueron inventados por John A. Simpson de la Universidad de Chicago en 1948. El monitor NM64 de "18 tubos" es el estándar internacional actualmente.

Con objetivo de facilitar datos de estos monitores existe la NMDB (Neutron Monitor Data Base) a la que se puede acceder desde internet e incluso visualizar los datos en una interfaz sencilla online.

La existencia del CaLMa, la NMDB y el open source ha motivado la posibilidad de desarrollar un Software abierto para visualizar los datos de este monitor, que puedan ser comparados con los de otros lugares del mundo y su posterior mejora para implementar nuevas funcionalidades que resulten necesarias en la investigación de los “rayos cósmicos”.

CaLMa

El monitor de neutrones de Castilla La Mancha [2], instalado en Guadalajara, se compone actualmente de 15 tubos de medición funcionales de los cuales doce son LND 2061 y tres BP28. Los LND 2061 han sido diseñados para tener un volumen de cuentas similar al BP28 y ser compatibles con los amplificadores Canberra ACHNA98. En un primer momento el CalMa iba a ser un monitor de 18 tubos NM64 estándar pero el gas Helio-3 que llevan este tipo de monitores incrementó su precio a un 625% y se optó por tubos con gas de Trifluoruro de Boro. El monitor utiliza el mismo sistema de adquisición que el de la Universidad de Kiel, llamado Kiel2 en la NMDB.

El sistema obtiene los datos de los 15 tubos contadores y de un medidor de presión BM35. Los datos se almacenan en cuentas/minuto para su posterior tratamiento y se llevan enviando a la base de datos del NMDB desde febrero de 2013. (Véase Ilustración 1)

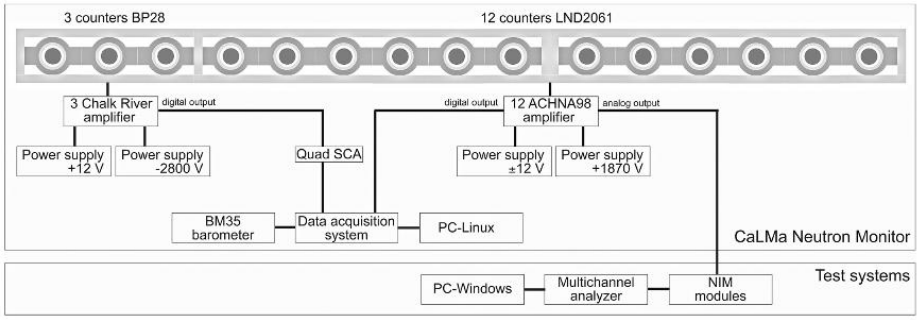

Ilustración 1:Sistema de adquisición [2]

CaLMante

El CaLMante es un software visualizador de datos programado en IDL que permite calcular y mostrar los datos en gráficas mediante una interfaz visual. Se compone de un menú principal donde seleccionar la fecha de los datos para empezar a realizar los cálculos y una pestaña plot para elegir las figuras que se desean mostrar (Ilustración 2).

A modo de ejemplo podemos observar que se pueden visualizar los canales individualmente con las cuentas partidas por segundo en el eje Y, y el rango de las fechas en el eje X (Ilustración 3), las medianas que como en el caso de la captura se puede elegir entre valores reales o escala relativa (Ilustración 4), histogramas (Ilustración 5) y el ajuste de la presión (Ilustración 6).

El programa incluía otras funcionalidades como transformadas, visualización de la presión, etc que se han omitido en este apartado puesto que el objetivo es familiarizarnos con el antiguo entorno programado en IDL y no una demostración exhaustiva de este.

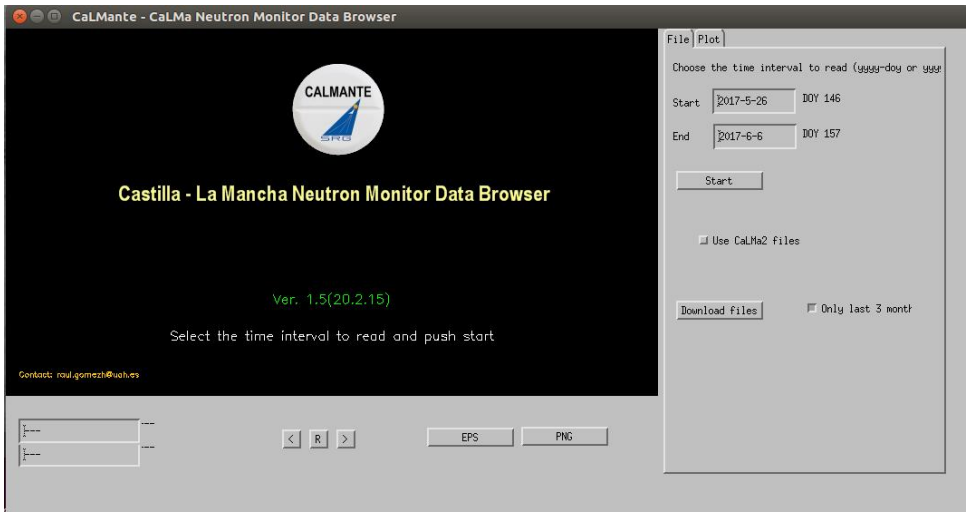

Ilustración 2:Pantalla principal del CaLMante

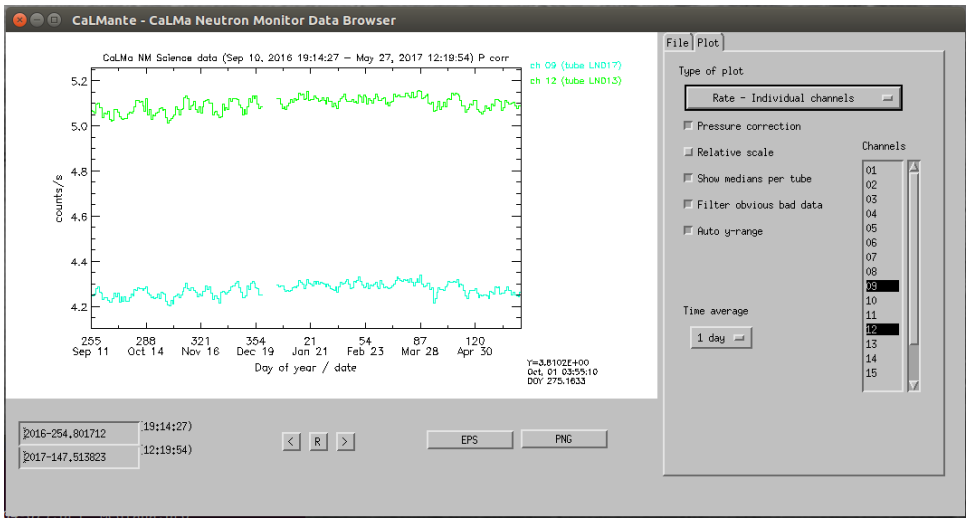

Ilustración 3: Representación de canales

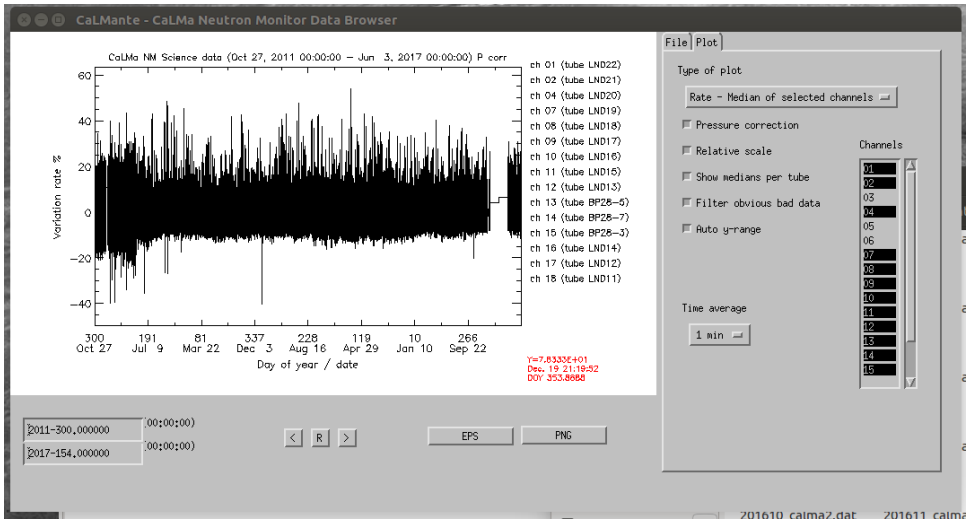

Ilustración 4:Cálculo de la mediana relativa

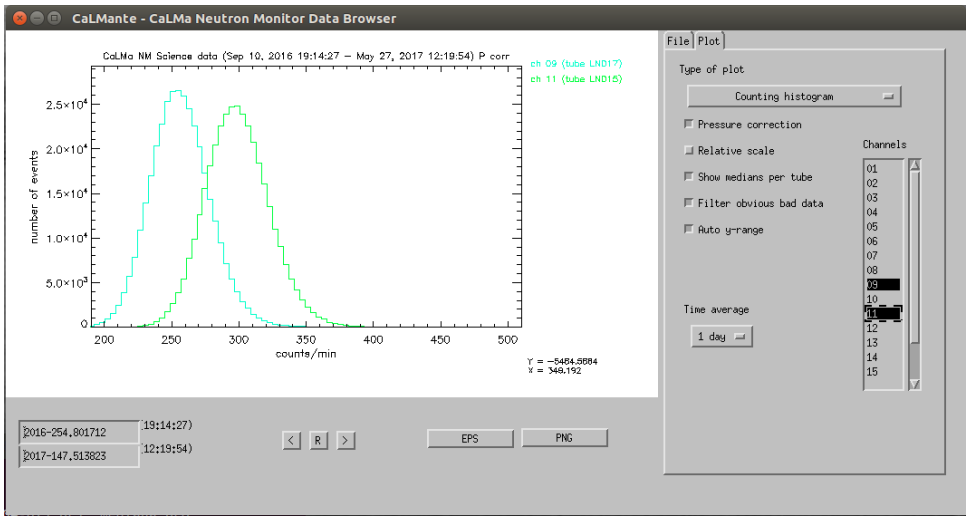

Ilustración 5:Histograma

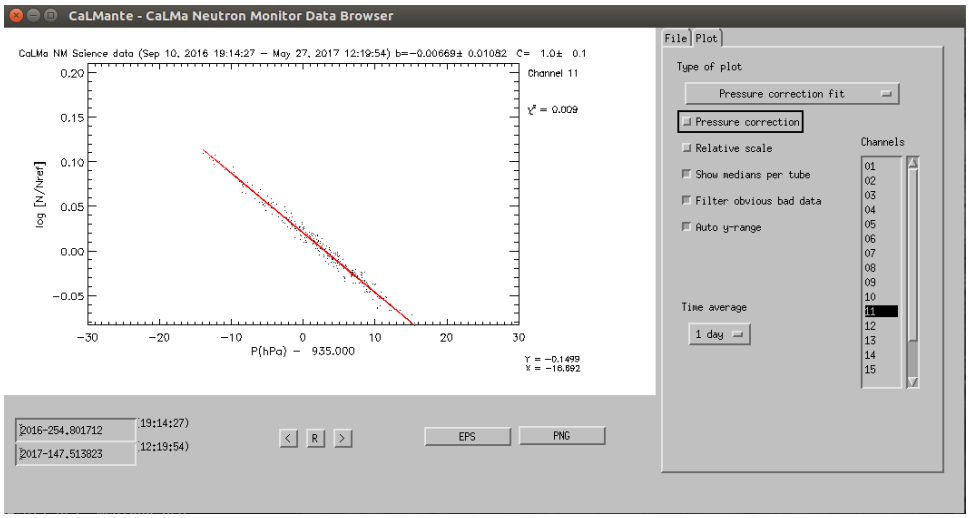

Ilustración 6: Ajuste de presión

El nuevo software

El nuevo software, programado ahora en Python, trata de imitar la interfaz y las características básicas del programa anterior, pero añade cambios y mejoras significativas. En el programa anterior solamente se podían exportar las capturas a PNG y EPS, en el nuevo gracias a matplotlib te permite exportarlo a EMF, EPS, PDF, PNG, PS, RAW, RGBA, SVG y SVGZ.

Otra novedad con respecto al anterior software es la exportación de los datos tratados a ASCII, lo que permite poder trabajar con ellos en otros visualizadores de datos o incluso en tablas de Excel.

Cabe destacar también la parte telemática del nuevo visualizador que incorpora tanto un script para descargar los ficheros del servidor como otro para visualizar e importar a ASCII los datos del NMDB de forma múltiple (Para una mayor profundización véase el apartado “descripción del programa”).

IDL VS Python

IDL

IDL [3] es un lenguaje para procesamiento de datos basado en Fortran y C. Permite entre otras cosas realizar gráficas 2d y 3d, animaciones, tratamiento de imágenes y cálculos matemáticos. Tiene una gran lista de formatos admisibles y no es necesario declarar el tipo de dato antes de usarlo.

El lenguaje incluye su propio entorno de desarrollo, multiplataforma y con su propio debugger.

IDL incorpora una función para leer archivos en formato ASCII y un asistente específico para ello.

Puede representar imágenes en 2D con una sola línea de comandos y permite visualizar múltiples gráficas a la vez.

Los tipos de datos más importantes de IDL están basados en vectores y matrices, ya que se diseñó para trabajar con estás de forma sencilla y natural.

Su principal desventaja es que no es software libre, su uso está supeditado a abonar una licencia de un alto valor económico y que hay que renovar. Esto limita su uso más allá de lo que la empresa propietaria permite, reduciendo la variedad de bibliotecas externas creadas por terceros.

Python

Python [4] es un lenguaje interpretado de alto nivel, enfocado en la claridad del código y en la metodología KISS (Keep It Simple Stupid).

A las bibliotecas se les denomina módulos, que brindan de un gran abanico de funcionalidades al programa.

El intérprete de Python contiene además un modo interactivo en el que ir escribiendo código en la consola en tiempo real.

Por defecto los scripts están codificados en UTF-8, siendo esto configurable en los comentarios iniciales del código.

Los tipos de datos por defecto son [5]:

- Str: Cadenas.
- Unicode: str pero en codificación Unicode.
- List: Listas.
- Tuple: Listas no modificables en tiempo de ejecución.
- Set: Lista no ordenada y sin contenido duplicado
- Frozenset: Similar a set pero no modificable en tiempo de ejecución.
- Dict: Diccionarios, Clave:Valor.
- Int: Enteros.
- Long: Similar a entero, pero de precisión arbitraria.
- Float: Decimales con coma flotante.
- Complex: Números complejos.
- Bool: Booleanos.

En Python son todo objetos, incluido las clases.

Comparativa

Analizando los pros y los contras de los dos lenguajes se ha llegado a una tabla comparativa (Tabla 1). Como se puede apreciar, Python facilita en gran medida el desarrollo de software y su gratuidad y su grado de acogida por los usuarios lo convierten en un lenguaje para tener en cuenta.

En cambio, IDL no tiene una comunidad tan grande y su mayor punto a destacar es su enfoque empresarial. Es decir, su soporte técnico y su entorno de desarrollo dedicado a través de la licencia propietaria. Siendo solamente recomendable para una empresa que facture una gran cantidad de dinero al año y necesite un entorno sin complicaciones ni cambios fuera del estándar para trabajar con datos.

	Python	IDL
Pros	• Lenguaje multipropósito. • Enfocado en la claridad de lectura. • Gran comunidad de desarrollo. • Libre de licencias.	• Pensado para tratamiento de datos. • Entorno de desarrollo propio. • Soporte por la empresa.
Contras	• Soporte exclusivamente altruista. • No todos los módulos tienen desarrollo profesional. • Dificultad para trabajar con hilos a ciertos niveles.	• Licencia propietaria. • Alto precio. • No es un lenguaje de propósito general. • Limitado en bibliotecas.

Tabla 1:Comparativa Python vs IDL

Metodología [6]

La metodología empleada ha sido un sistema propio de la ingeniería de software que se ha dividido en cuatro fases, como se observa en la ilustración 7 y que es importante explicar por separado:

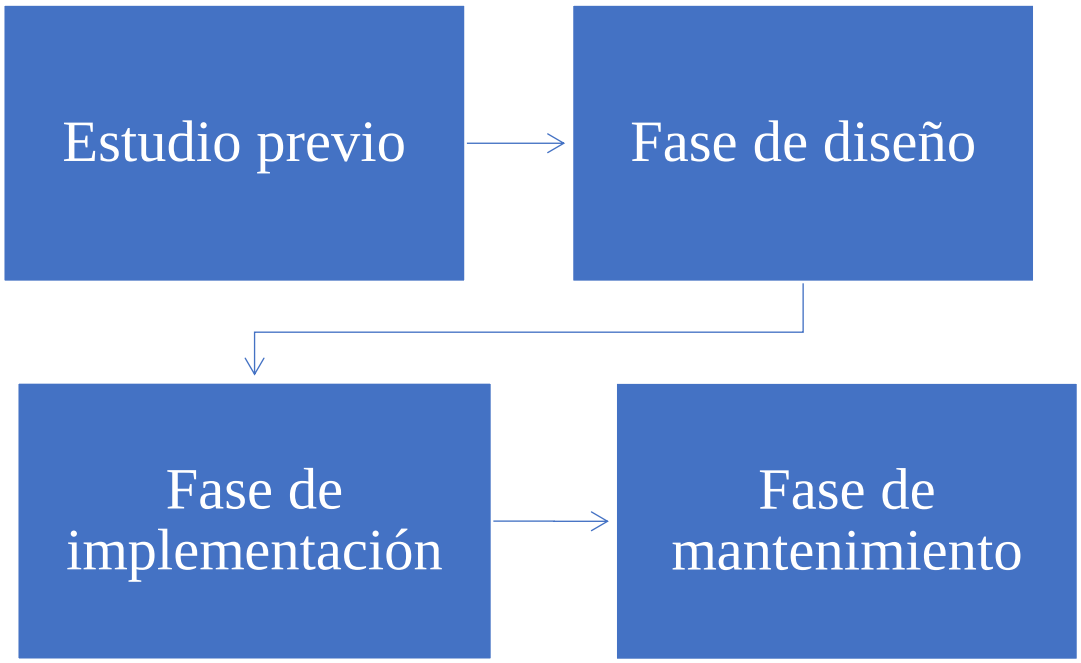

Ilustración 7: Esquema de las fases

- Estudio previo:

En esta fase mediante reuniones con el tutor se llegó a la conclusión de las necesidades que tenía el software a desarrollar, el lenguaje de programación que había que utilizar, se recogió la documentación previa adecuada y el objetivo de este. También en esta fase mediante investigación del alumno se buscó información sobre la viabilidad del programa, las funcionalidades requeridas y los módulos de Python necesarios.

- Fase de diseño:

La fase consistió en desarrollar la interfaz gráfica, los algoritmos para los cálculos, concatenación de fichero, desarrollo de módulos de forma independiente y escritura del código en general.

- Fase de implementación:

Los algoritmos y módulos diseñados en la fase anterior se unen para crear la versión “beta” del programa, ya funcional, pero sin haber corregido la mayoría de los fallos. Lo que permite ya ser utilizado en cuestiones puntuales y observar si todo se ha diseñado de forma correcta y compatible entre sí.

- Fase de mantenimiento:

En esta última fase se revisan todos los "*bugs*" que pudiera tener el programa. En un primer momento en el ordenador del alumno, posteriormente, en el ordenador del tutor, donde se revisó aparte de que todas las funcionalidades del software fueran idénticas a las del alumno, que todos los módulos fueran compatibles e instalables también en ese PC sin ningún tipo de fallo adicional.

Descripción del programa

1. La interfaz gráfica

Para diseñar la interfaz gráfica utilizaremos el módulo pyQT5 basado en la API QT escrita en C++. Una vez instalada, instalaremos también las developer tools entre las que se incluye el QT Designer (Ilustración 8), una aplicación que nos permite editar la interfaz de forma gráfica y así ahorrarnos el trabajo de codificarlo directamente. Esta interfaz se nos guarda en formato .ui y escribiendo en la consola el comando "*pyuic5 -x nombreinterfaz.ui -o nombrepython.py*" lo pasaremos a Python.

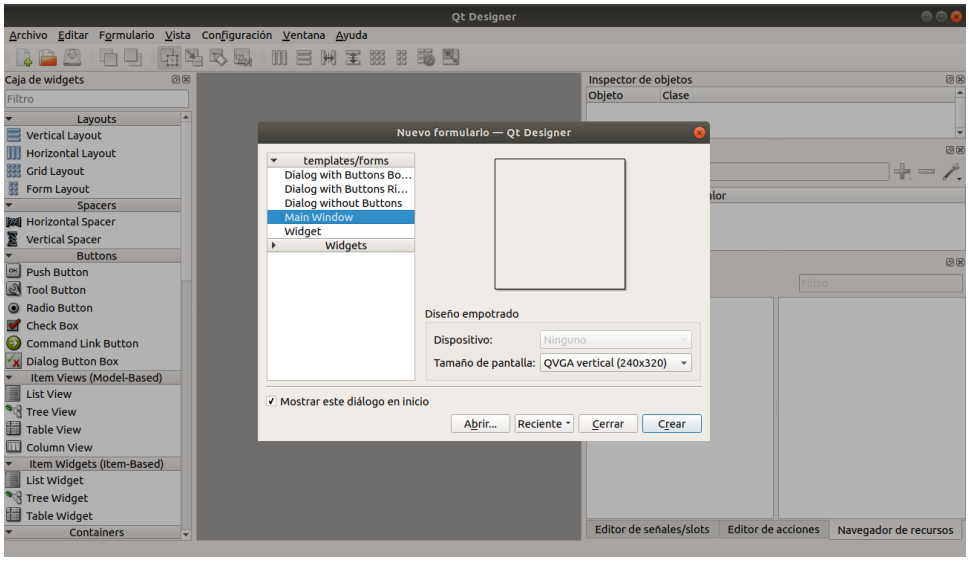

Ilustración 8: QT Designer, editor gráfico de la interfaz

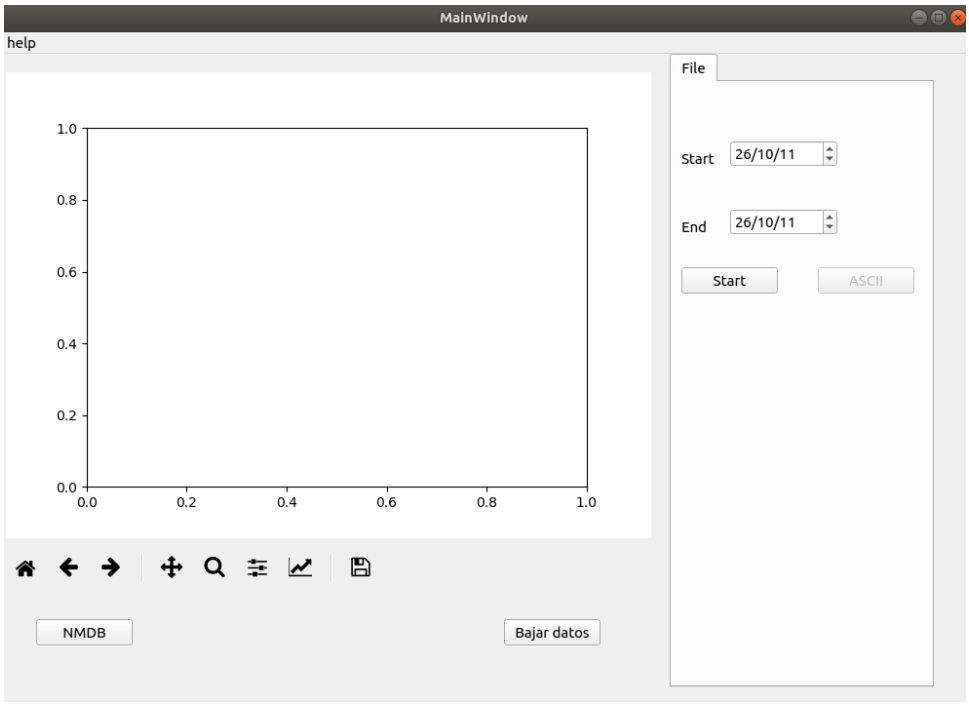

Ilustración 9: Menú principal

Para la interfaz se ha añadido un plugin de matplotlib [7], dos modificadores de fecha, varios botones, dos pestañas, menús desplegables y botones selectores redondos que nos servirán de ayuda para implementar todas las funcionalidades (Ilustración 9 y 10).

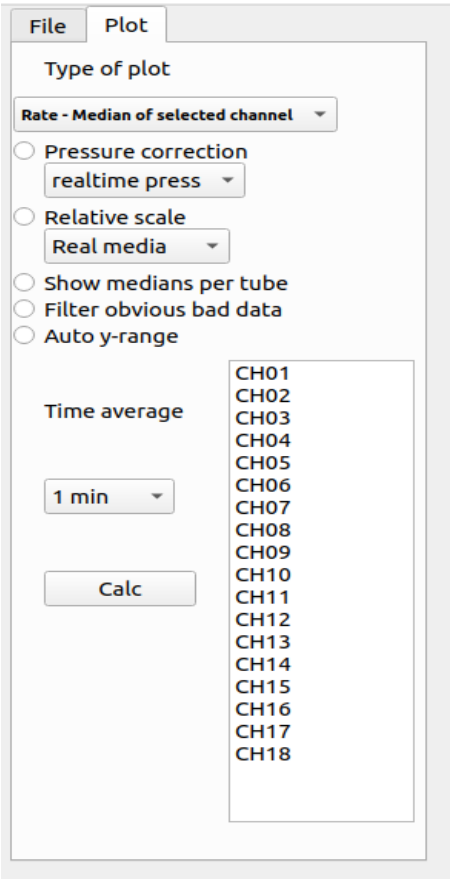

Ilustración 10:Pestaña Plot

2. Carga de datos

El primer paso que hay que realizar a la hora de trabajar con los datos es concatenarlos, para ello realizaremos un bucle anidado y utilizaremos el módulo pandas que convertirá las columnas de los datos en dataframes que es un tipo de datos que se puede dividir en columnas.

Habrá que comprobar si la fecha de inicio de los datos es menor que la fecha de fin. En caso afirmativo se procede a cargar los datos utilizando un bucle con condicionales para todos los rangos de fechas posibles. (Véase Anexo I: Fichero de interfaz y carga de datos)

3. Representación de los canales

Para representar cada canal, se crea una función que nos permita seleccionar los canales deseados a representar, en el eje de las x introduciremos el índice de el dataframe que coincide con las fechas y en el eje de las y deberemos poner el dataframe del canal seleccionado dividido entre 60 para que los valores se expresen en cuentas/segundo. Y con el método "*loc[DIni:DFin]*" , siendo Dini la fecha de inicio seleccionada y DFin la fecha de fin seleccionada, acotaremos los datos al rango de fechas elegido. También habrá que asignar un color a cada canal para que no existan equivocaciones a la hora de representarlos. (Véase Anexo I: Representación de canales) En la ilustración 11 se puede apreciar el resultado final con el eje Y medido en cuentas partidas por segundo y en el eje X su fecha y su DOY (Day Of Year).

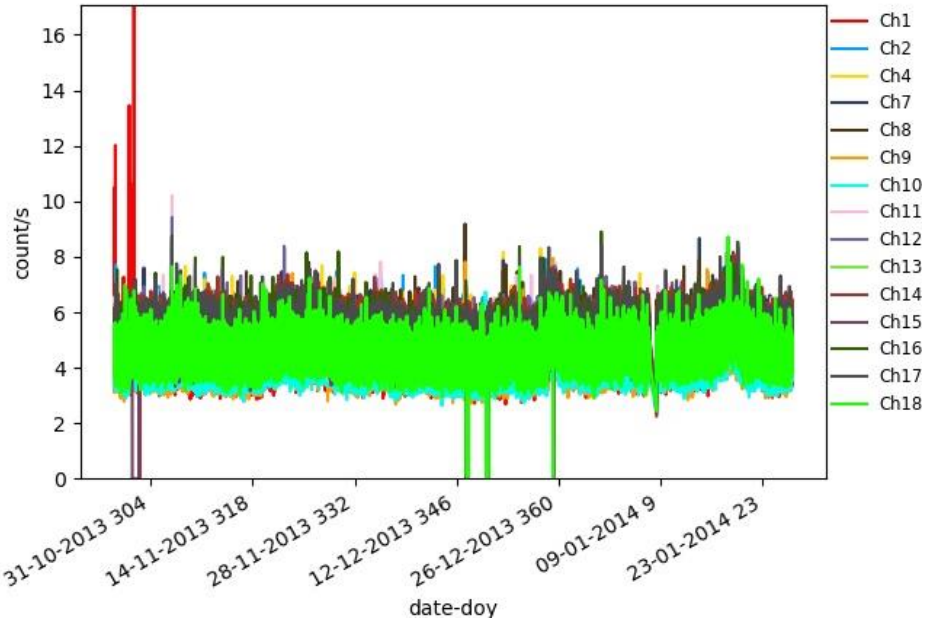

Ilustración 11:Representación de cada canal individual

4. Cálculo de la mediana

Para calcular la mediana utilizaremos la función de numpy *"median()"* que es extrapolable a los dataframes de pandas, la función en su uso normal calcularía la mediana de cada columna. En este caso se necesita para calcular la mediana de las filas de datos en cada unidad temporal, por

ello el procedimiento realizado es aislar las columnas que se refieran solo y exclusivamente a los canales de datos del CaLMa, indexados en el tiempo y se trasponen para que la función trabaje calculando la mediana de cada instante de tiempo entre los canales seleccionados. (Véase Anexo I: Mediana) La gráfica quedará como se ve en la ilustración 12, mostrando en el eje Y las cuentas partidas por minuto, y en el eje X la fecha y el DOY al que pertenecen los datos.

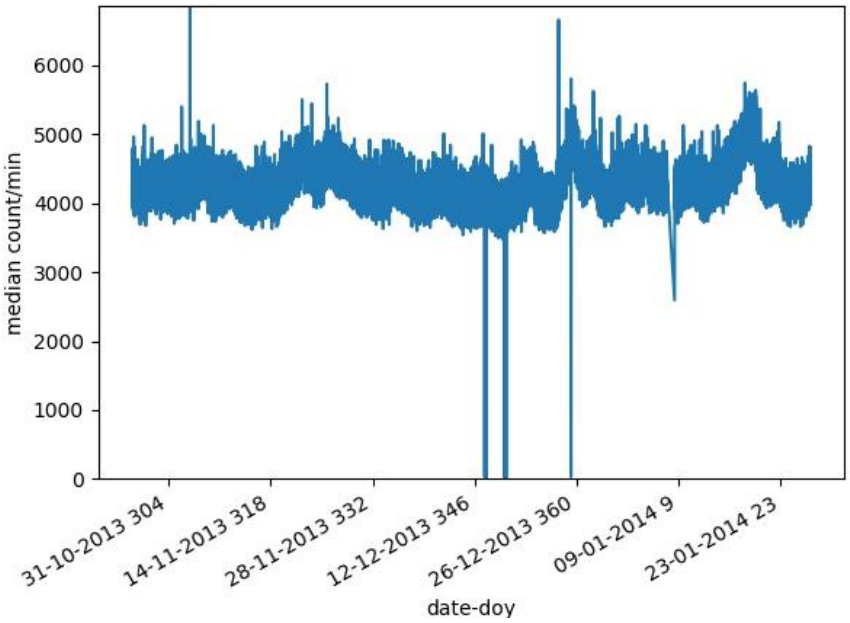

Ilustración 12: Representación de la mediana

5. Histograma

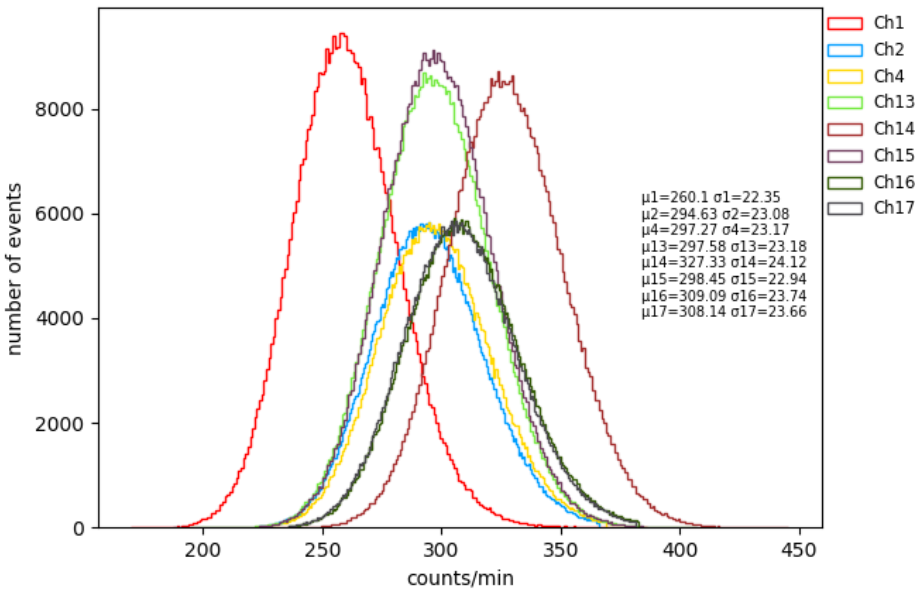

Ilustración 13: Histograma

Para representar el histograma (Ilustración 13) utilizaremos la función *"hist"* del módulo matplotlib, que calcula y dibuja el histograma de una serie dada, representando en el eje Y el número de eventos y en el eje X las cuentas partidas por minuto. En este caso las series serán las columnas de cada canal de datos por separado. La función es similar a la histogram de numpy, por ello a la salida de los datos en ASCII obtendremos los datos de salida del hist de numpy. De bin se ha utilizado 225 que se saca de la cuenta (1000-100)/4 que se usaba en el antiguo programa de visualización de datos programado en IDL. Para facilitar el trabajo, el tráfico muestra la media y la desviación típica de los histogramas seleccionados (Véase Anexo I: Histograma).

6. Centrado de histograma

El centrado del histograma (Ilustración 14) se realiza de forma similar a la del histograma normal pero en este caso los datos deberán ser restados por su media del canal, para lo que se usa la función "*median()*" del numpy que viene en el módulo pandas por defecto. (Véase Anexo I: Histograma centrado)

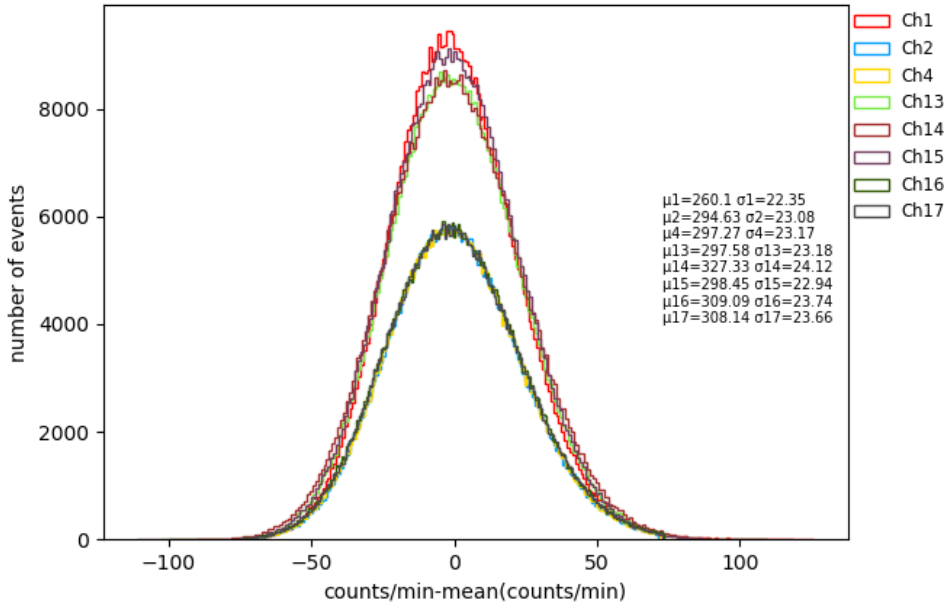

Ilustración 14:Histograma centrado

7. Pressure correction fit

Para hallar la recta de la beta de la presión, lo primero que tenemos que hacer es la mediana de los canales seleccionados de la misma forma que en el apartado de cálculo de la mediana. Esta mediana habrá que dividirla por la media de los valores de referencia de los canales y posteriormente se hará el logaritmo neperiano del cálculo anterior y lo dejaremos como eje Y.

En el eje X se representará la presión menos la presión referencia de la estación.

Por último utilizando la función "*LinearRegression*" de sklearn, trazaremos la recta de regresión y para capturar el valor de la beta almacenaremos el valor coef_[0] que corresponde con la beta

de la estación. (Véase Anexo I: Ajuste de la presión) El resultado final se puede ver en la ilustración 15 donde se representan en el eje X la resta entre la presión y la presión referencia, y en el eje Y el logaritmo neperiano del valor entre el valor referencia, encabezando la gráfica se muestra la beta que ha sido calculada.

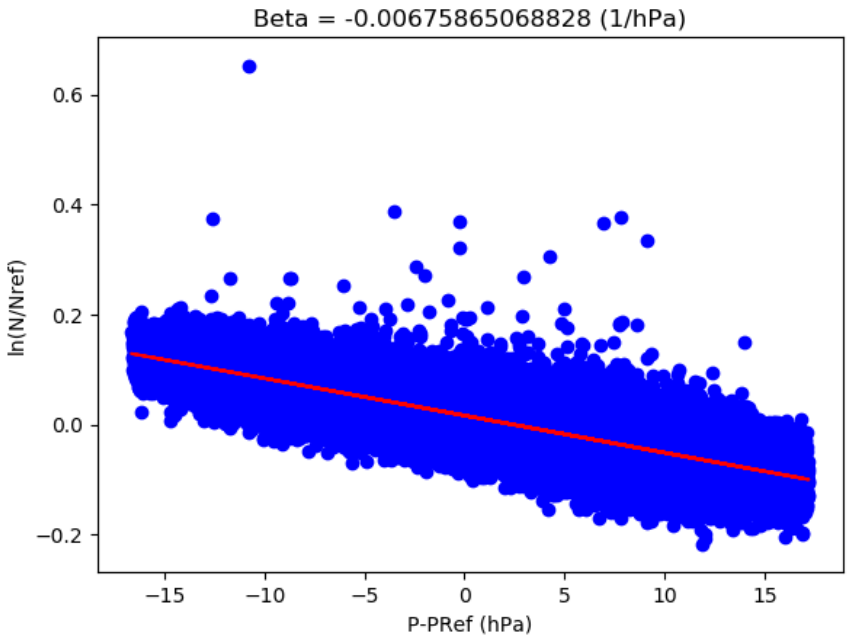

Ilustración 15: Pressure correction fit

8. Promediado

Para hacer los promedios se ha utilizado la función *"resample"* de pandas, que permite promediar instantes de tiempo de forma configurable por el usuario. Hemos configurado el programa para que promedie en 1 minuto, 5 minutos, 30 minutos, 1 hora, 1 día y 27 días. Como tipo de promediado hemos seleccionado a través de la media de los datos del intervalo, en intervalo abierto y de principio de los datos a fin. (Véase Anexo I: Representación de canales y Anexo I: Mediana).

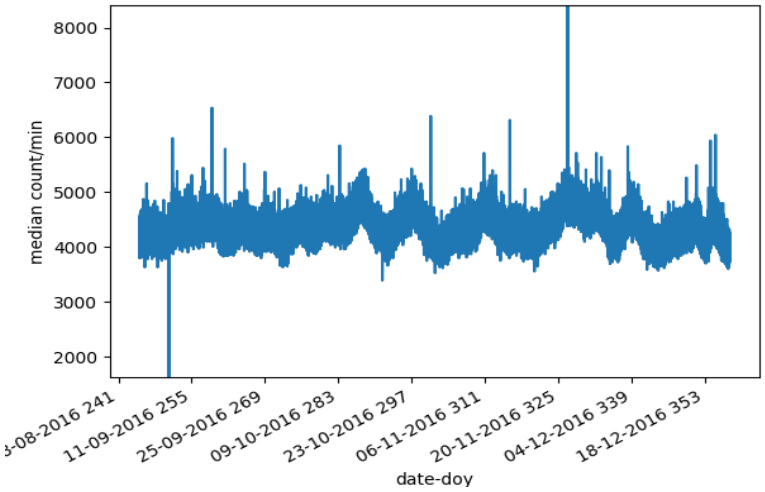

Ilustración 16:Promediado a 1 minuto

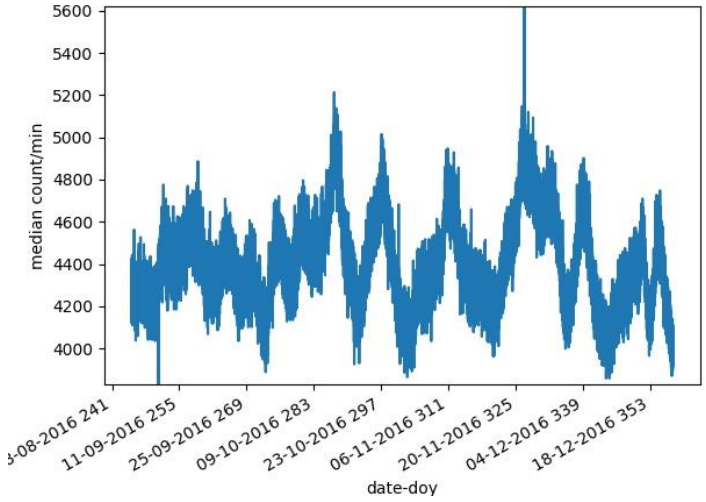

Ilustración 17:Promediado a 5 minutos

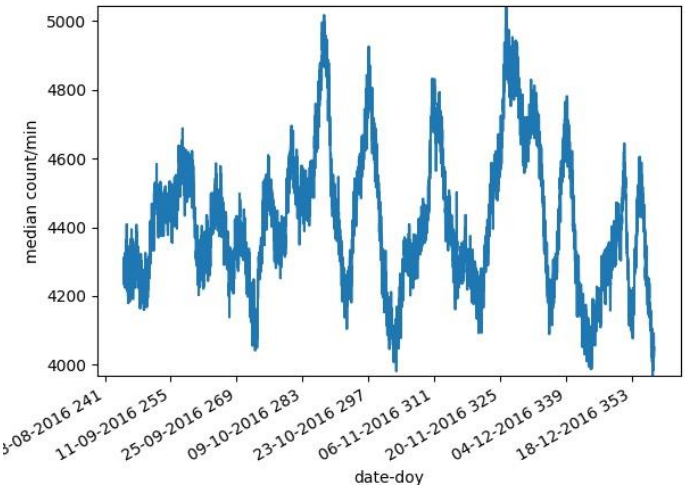

Ilustración 18:Promediado a 30 minutos

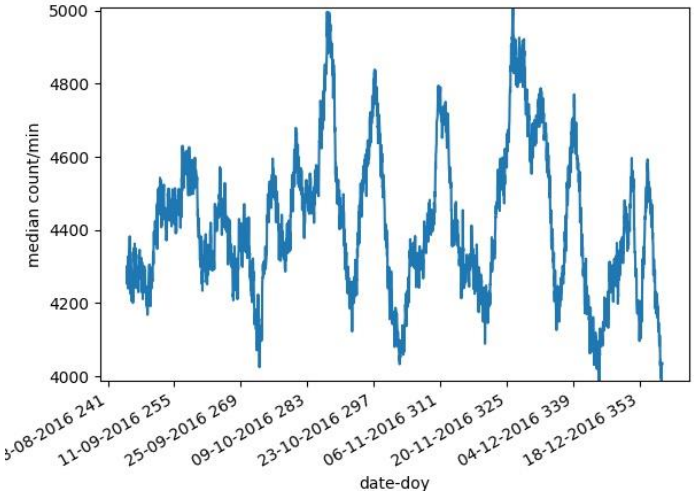

Ilustración 19:Promediado a 1 hora

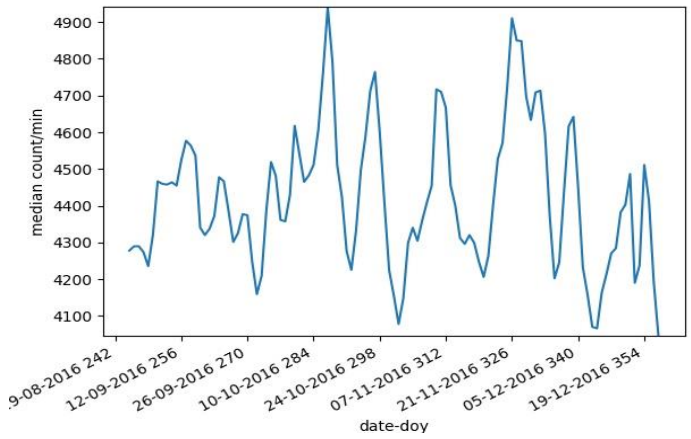

Ilustración 20:Promediado a 1 día

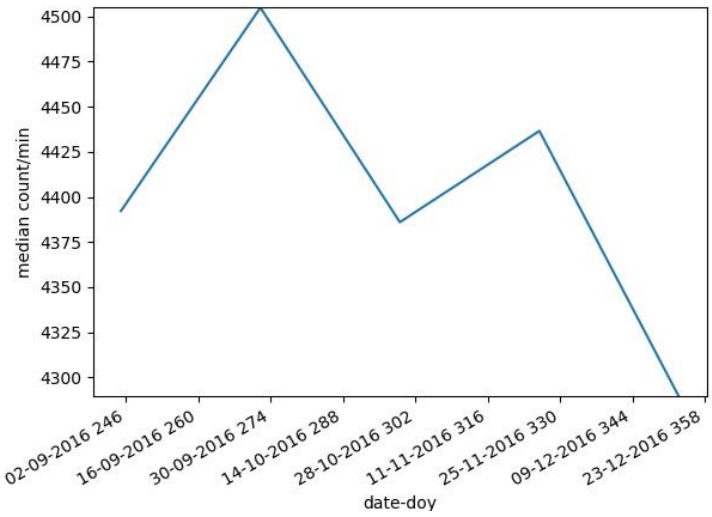

Ilustración 21:Promediado a 27 días

De la ilustración 16 a 21 (Mediana de las cuentas partidas por minuto en eje Y, fecha-DOY eje X) se puede apreciar un ejemplo de promediado con la mediana partiendo de 1 minuto a 27 días, en el intervalo del 1/9/2016 a 26/12/2016. En las ilustraciones se puede ver a simple vista como se modifica la gráfica a medida que elegimos un promedio de tiempo más grande, en la ilustración 16 la gráfica es muy gruesa y ruidosa, con muchos cambios y en la 21 prácticamente sólo podemos apreciar lo cambios entre máximos y mínimos.

9. Escala relativa

En la escala relativa, cambiamos el eje Y por el porcentaje que varían los datos de los canales respecto a la media, tenemos para ello dos modos. El primer modo (Ilustración 22) calcula el porcentaje que se desvían los datos con respecto a la media de los datos en el intervalo seleccionado y el segundo modo (Ilustración 23) con respecto a la media de los datos según un valor estimado en la estación.

La fórmula aplicada es la misma para los dos casos:

$$Valor_{nuevo} = \frac{(Valor\ del\ canal - Media\ del\ canal)}{Media\ del\ canal}$$ (Véase Anexo I: Escala relativa)

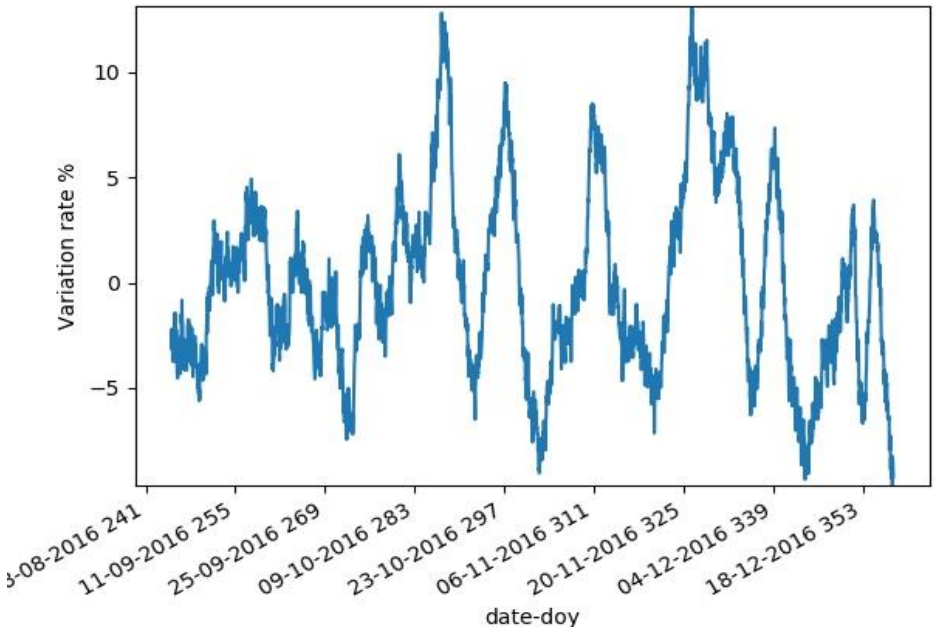

Ilustración 22:Relativo con media del intervalo

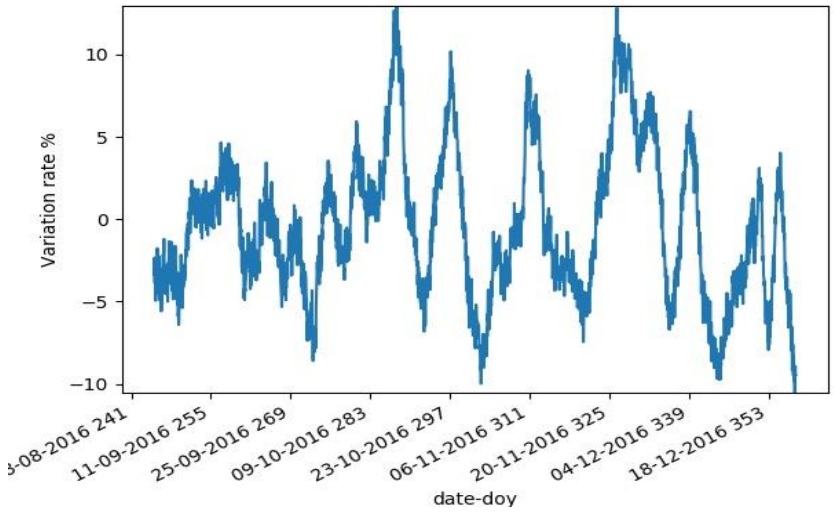

Ilustración 23:Escala relativa con media fija

10. Corrección de presión

En la corrección de presión nos guiaremos de la siguiente fórmula $N=N0 * e^{-\beta(P-P0)}$ siendo N el valor del canal corregido, N0 el valor del canal sin corregir, P el valor de la presión, P0 la presión referencia y β el coeficiente de la presión. (Véase Anexo I: Corrección de la presión)

Tenemos dos formas de ajustar la presión:

- Con valores fijos (Ilustración 24):

  En esta forma se tienen unos valores de β calculados previamente y simplemente habrá que sustituir en cada canal aplicando la formula anterior.

- Con valores variables (Ilustración 25):

  De esta manera se vuelve a utilizar la función "*LinearRegression*" de sklearn, esta vez con cada canal independientemente y se captura su valor coef_[0] como β y se aplica la fórmula explicada más arriba.

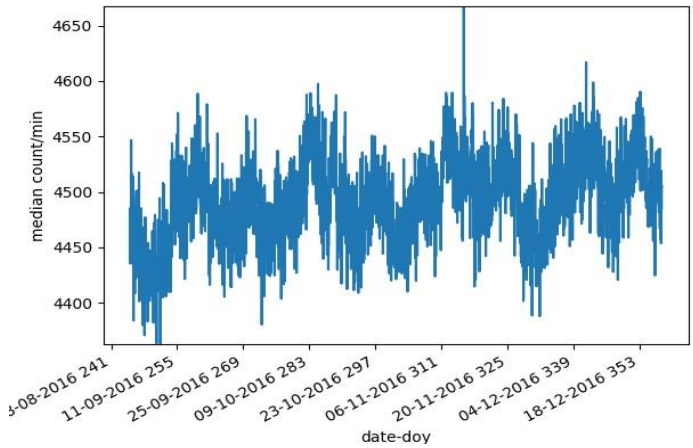

Ilustración 24:Ajuste de presión con valores fijos

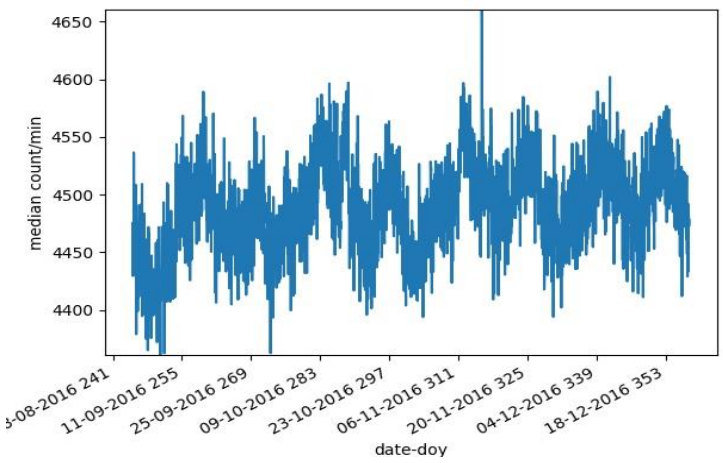

Ilustración 25: Ajuste de presión con valores variables

11. Borrado de datos erróneos

El criterio que se ha seguido es eliminar todos los valores mayores de la media de los canales más 3 veces la desviación típica y todos los valores menores que la media de los canales menos 3 veces la desviación típica. Para ello se calcula la media y la desviación típica, y por último se buscan los valores que no cumplan la condición mediante un bucle. (Véase Anexo I: Eliminar datos obviamente malos).

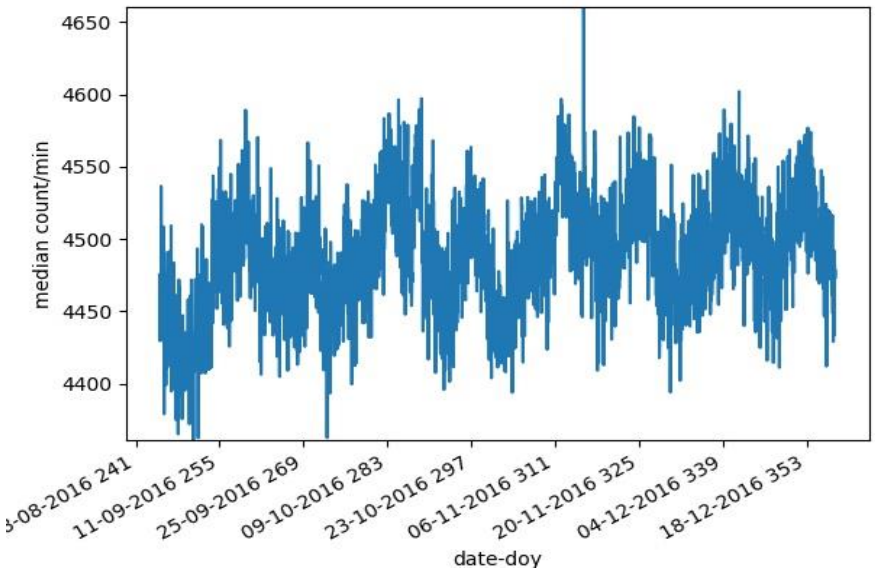

Ilustración 26: Sin eliminación de datos obviamente malos

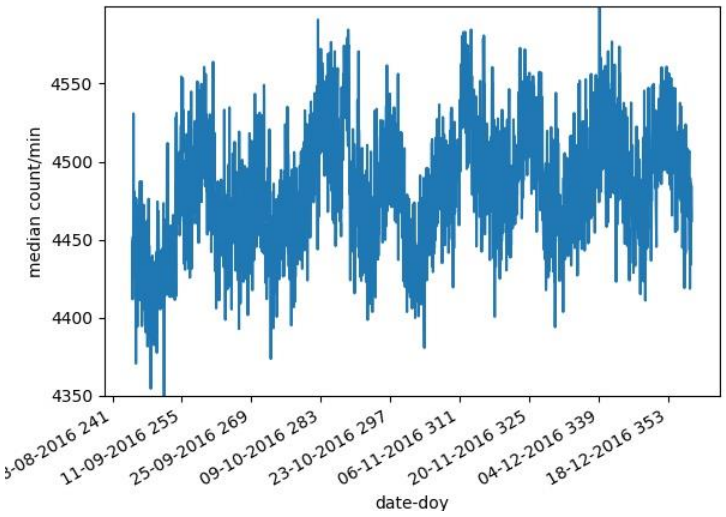

Ilustración 27: Con datos obviamente malos eliminados

Como se puede apreciar en la ilustración 26 no se han eliminado los datos obviamente malos y en la ilustración 27, eliminando picos máximos y mínimos que con tenían sentido si lo comparamos con la gráfica en su conjunto.

12. Otros ajustes

Si queremos ajustar las gráficas a los valores máximos y mínimos del eje Y, simplemente tendremos que utilizar el comando de matplotlib "*autoscale*" con la siguiente notación ax.autoscale(enable=True, axis='y', tight=True).

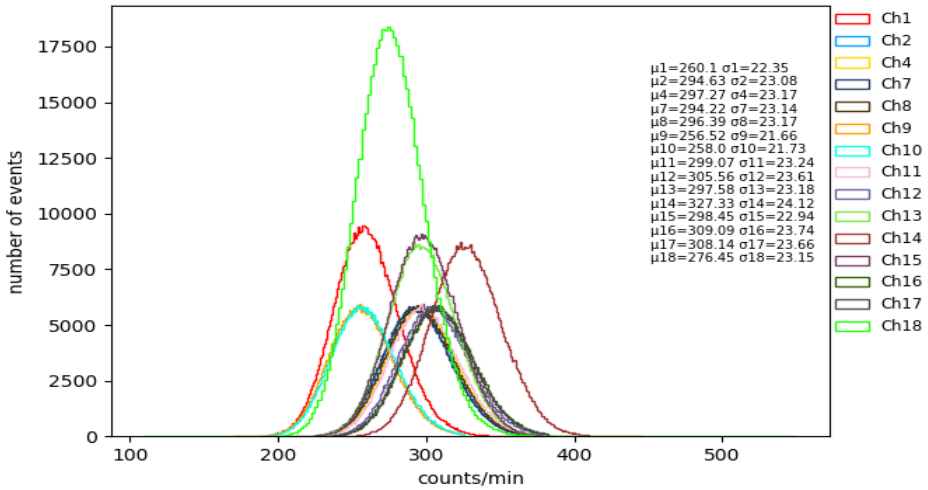

Ilustración 28:Sin autoY range

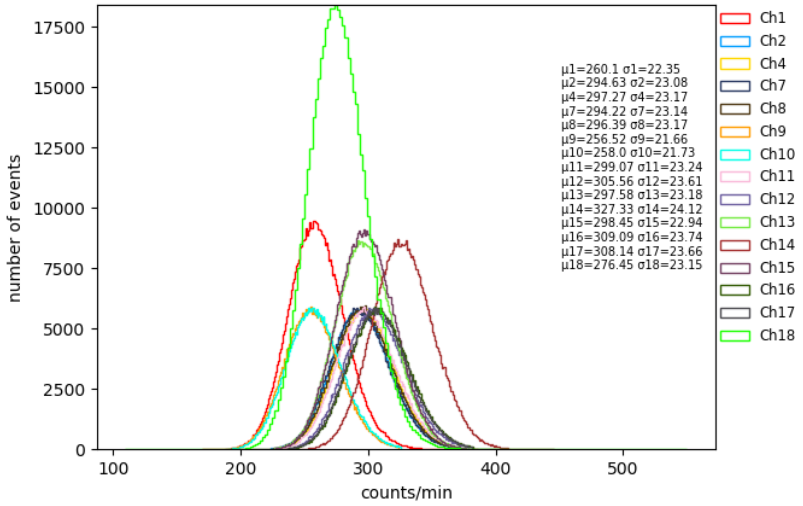

Ilustración 29:Con auto Y range

En la ilustración 28 no tenemos seleccionado el auto Y-range, en la 29 sí adaptándose el eje Y al máximo del histograma más alto.

13. Descarga de ficheros

La descarga de ficheros de datos desde el servidor se divide en dos partes. La primera, la creación de un menú intuitivo con una fecha de inicio y una fecha de fin para poder descargar varios ficheros en cadena. La segunda, es la propia descarga de ficheros, para esto haremos un bucle anidado que recorra todas las fechas del rango y utilizaremos la función "*urllib.request.urlretrieve*" de python que hará la petición del fichero al servidor.

Estos ficheros se almacenarán en la carpeta "*data"* que, de no existir, se nos creará por defecto al realizar la petición. (Véase Anexo I: Descarga de los ficheros de datos del servidor)

14. Menú de conexión con la NMBD

Para esta funcionalidad se crearon dos módulos, el primero es el menú donde se creará la interfaz, la función que almacena todos los parámetros de los datos solicitados y la de guardar los datos recibidos en ASCII. El otro módulo es el que recibe los parámetros, descarga los ficheros del NMDB, gracias al módulo "*BeautifulSoup"* que busca entre las etiquetas los datos válidos, crea el dataframe con "*pandas*" y los representa con matplotlib. (Véase Anexo I: Módulo de descarga y representación de datos del NMDB, y Anexo I: Menú para solicitar datos al NMBD)

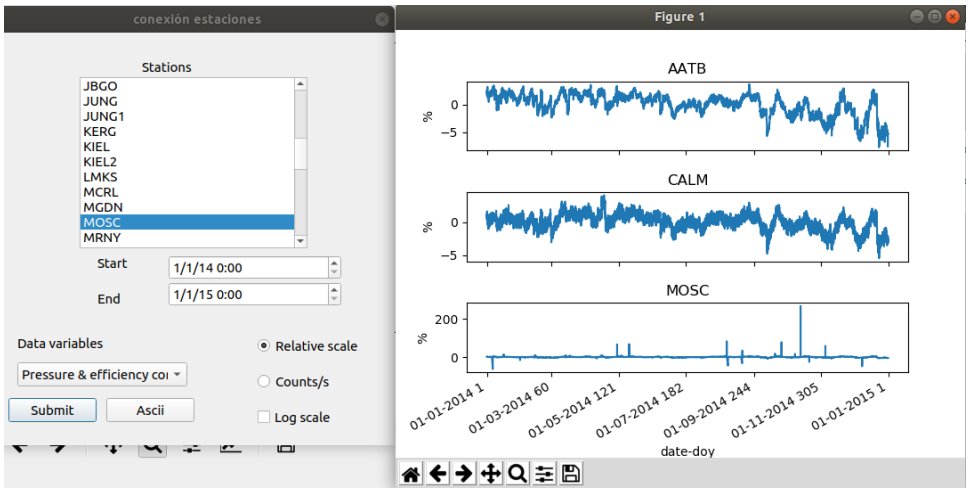

Ilustración 30:Menú NMDB

En la ilustración 30 podemos ver como se muestra el menú de opciones y la representación de las gráficas cuando solicitamos datos al NMDB.

15. Exportar a ASCII

Para la función de exportar a ASCII primero debemos crear un menú que nos permita seleccionar el directorio y el nombre con el que queremos guardar los datos en ASCII, PyQT ya viene con este dialogo creado. Una vez hecho esto, pandas permite importar sus dataframe a CSV con el método *"to_csv"* y si se selecciona encoding = 'ascii' se nos codificará automaticamente en ASCII. (Véase Anexo I: Fichero de interfaz y carga de datos, función llamada “def ascii:”)

Manual de instalación

Lo primero que hay que instalar es Python 3, como el sistema para el que fue diseñado esta aplicación es Ubuntu, no será necesario de instalar puesto que ya viene por defecto.

El siguiente paso es instalar el módulo PyQT5 y sus herramientas de desarrollador por si queremos modificar algo posteriormente, para ello utilizamos los comandos en la shell de linux: *"sudo apt-get install python3-pyqt5", "sudo apt-get install pyqt5-dev-tools"*y *"sudo apt-get install qttools5-dev-tools"*.

Otro módulo necesario es el matplotlib, también instalable mediante "*sudo apt-get install python3-matplotlib*".

Si no tenemos el módulo pandas instalado habrá que escribir en la shell *"sudo apt-get install python3-pandas*". (Véase ilustración 31)

Para que funcione correctamente la descarga de datos del NMDB hay que instalar el beautifulsoup 4 con el comando "*sudo apt-get install python3-bs4*".

Y para finalizar el scikit learn que instalaremos usando *"sudo apt-get install build-essential python3-dev python3-setuptools \ python3-numpy python3-scipy \ libatlas-dev libatlas3gf-base"*, este comando además nos permite tener actualizado numpy y scipy también necesarios de forma secundaria.

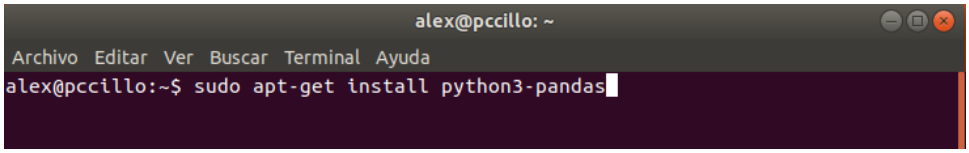

Ilustración 31:Ejemplo de comando en la shell

Ahora deberemos descomprimir el fichero comprimido (Véase ilustración 32) donde guardamos nuestro programa y nos aseguramos de que en su interior se encuentre una carpeta data en la que se guardan nuestros archivos de datos.

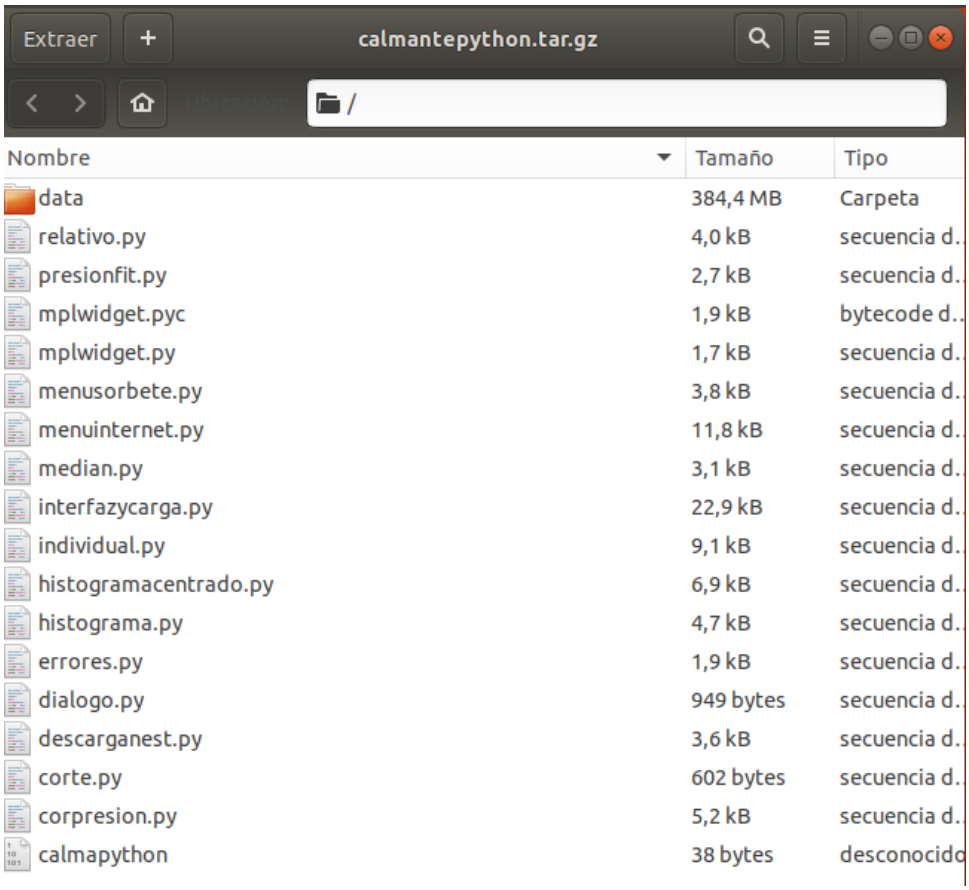

Ilustración 32:Descompresión de los datos

Y si se ha realizado todo correctamente ya solo habría que ejecutar el programa mediante la shell con *"./calmapython"* (Véase ilustración 33)

```
alex@pccillo:~/Escritorio/Tfg$ ./calmapython
```

Ilustración 33: Ejecución del programa

Manual de usuario

Ejecución normal

Ejecutamos el script y se nos cargará el menú principal (Ilustración 34)

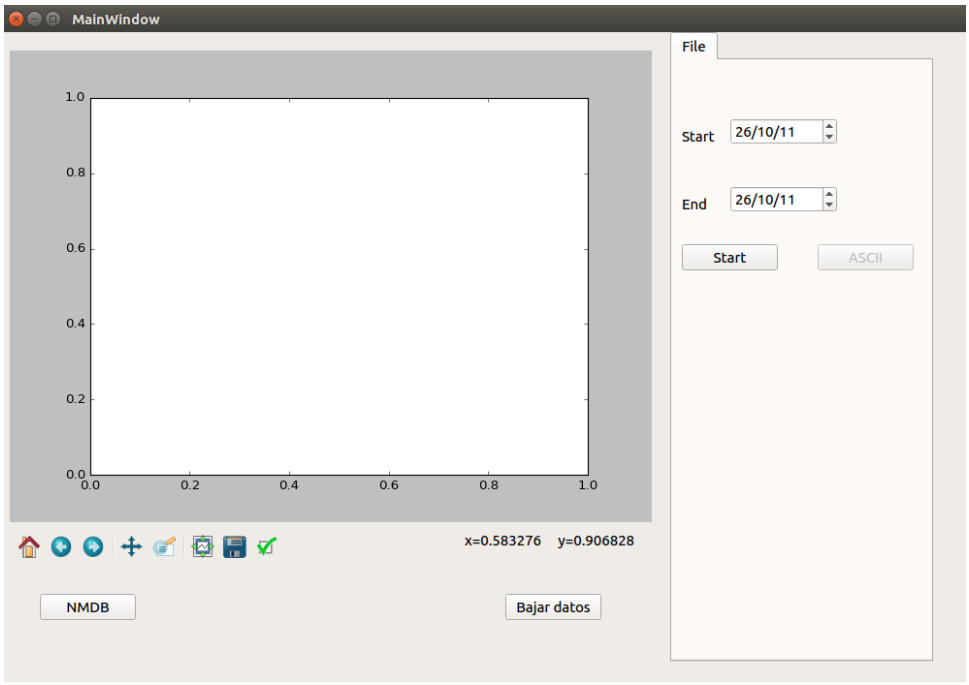

Ilustración 34:Menú principal

Para hacerlo funcionar, con previamente los archivos de datos de la estación almacenados en la carpeta data, se selecciona una fecha de inicio en el apartado Start (1) y una fecha de fin en el apartado End (2). Posteriormente habrá que presionar el botón Start (3) y esperar a que carguen los datos.

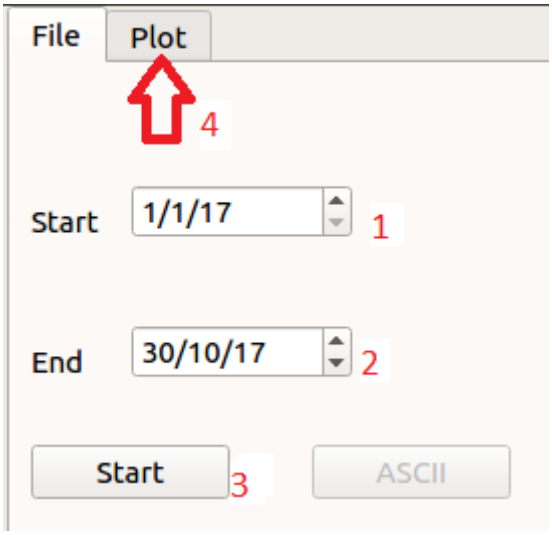

Ilustración 35:Proceso de carga de los datos

Si la carga ha sido correcta y se ha seguido el orden numérico de la Ilustración 35, se habrá desbloqueado la pestaña Plot (4) y se podrá trabajar con los datos de la estación.

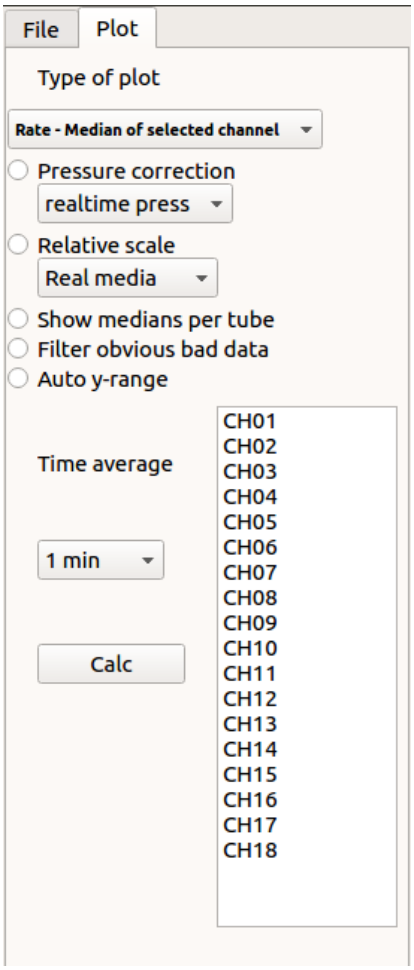

Ilustración 36:Pestaña plot

En el primer apartado "Type of plot" (Ilustración 37) se selecciona el tipo de gráfico y cálculos que se desean realizar:

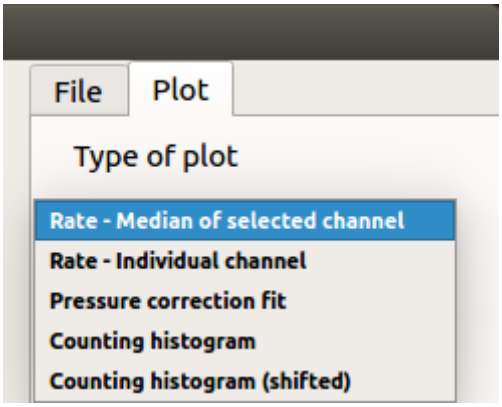

Ilustración 37: Type of plot, opciones

- Rate-Median of selected channel:

  Representa la mediana de los canales seleccionado

- Rate-Individual channel:

Representa cada canal seleccionado de forma individual

- Pressure correction fit:

  Representa en el eje X la diferencia entre la presión y la presión de la estación, y en el eje Y el logaritmo neperiano del dato de la estación entre el dato referencia, además de la recta de ajuste de la presión.

- Counting histogram:

  Es el histograma de los datos de la estación por canal.

- Counting histogram (shifted):

  Idéntico a Counting histogram pero centrado en 0.

Debajo aparecen una lista de funciones seleccionables (Ilustración 38):

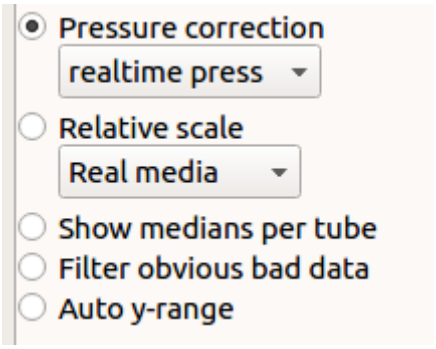

Ilustración 38:Funciones adicionales

- Pressure correction (Ilustración 39):

Corrige los datos con la beta adecuada para contrarrestar los efectos de la presión, tiene dos opciones adicionales que son "realtime press" y "ref value", "realtime press" utilizará una beta acorde con los datos cargados actuales y "ref value" utilizará la beta calculada de la estación.

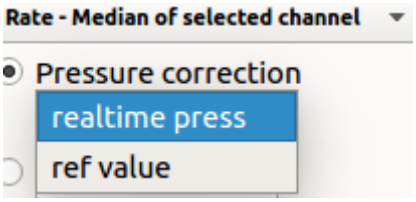

Ilustración 39:Pressure correction, opciones

- Relative scale (Ilustración 40):

Fuerza las representaciones de los canales a ser dibujadas en función de su porcentaje con respecto a la media. También contiene dos opciones adicionales "Real media" y "Station media", en "Real media" se representan los datos en función de las medias reales de los datos actuales de los canales y en "Station media" en función de las medias calculadas de la estación.

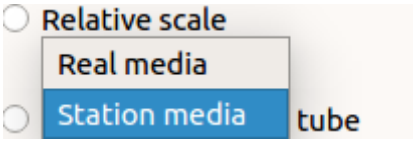

Ilustración 40:Relative scale, opciones

- Show median per tube:

  Sirve para representar la mediana pura sin contar el número de canales.

- Filter obvious bad data:

  Elimina los datos que se desvían demasiado de lo que se considera "normal".

- Auto y-range:

  Ajusta la gráfica al máximo y mínimo del eje Y.

Posteriormente para el correcto funcionamiento de la representación gráfica se deberán elegir los canales en el recuadro de abajo a la derecha. (Ilustración 41)

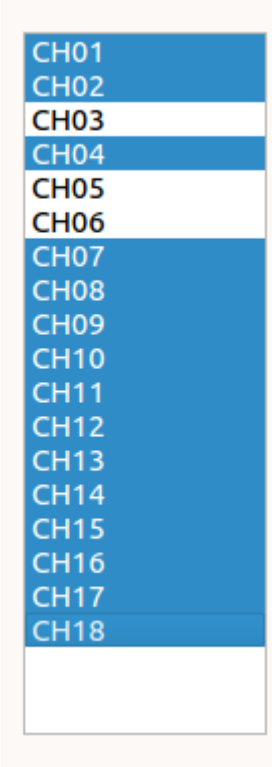

Ilustración 41: Lista de canales

Por último, en "time average" se seleccionará el promediado de tiempo de los datos de los canales. Pudiéndose elegir 1 minuto, 5 minutos, 1 hora, 1 día y 27 días. (Ilustración 42)

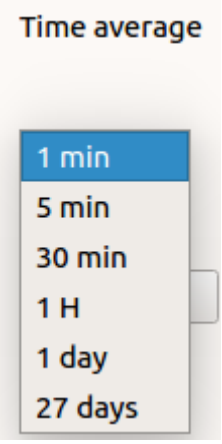

Ilustración 42: Lista de promedios

Con todos los pasos realizados correctamente se deberá pulsar el botón "calc" que ejecutará la función de representación. (Vease Ilustración 43)

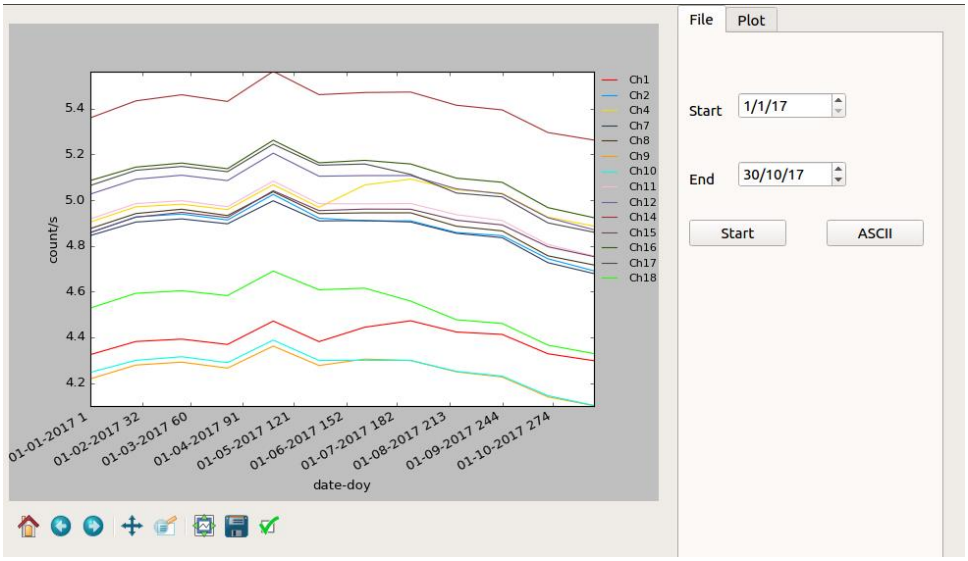

Ilustración 43:Ejemplo de representación

Si los pasos anteriores se han realizado con éxito, quedará habilitado el botón ASCII en el menú file para poder guardar los datos en un fichero de texto. (Ilustración 43)

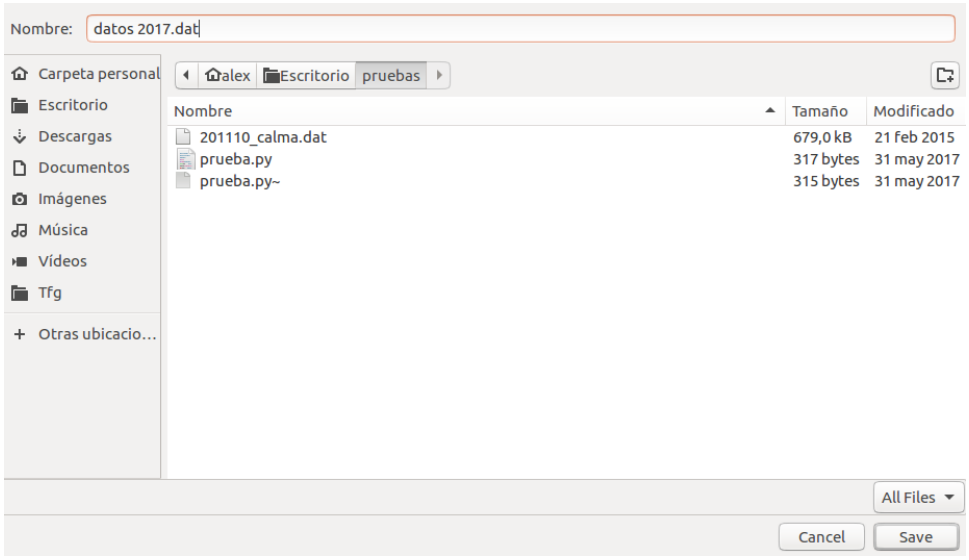

Ilustración 44:guardado de datos

Se le deberá asignar un nombre, seleccionar el directorio de almacenamiento y pulsar el botón "Save" para guardar. El resultado será un fichero en texto plano ASCII. (Ilustración 45)

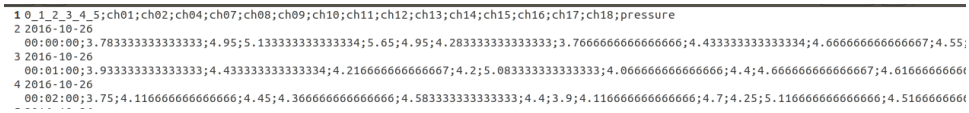

Ilustración 45:Fichero de datos ASCII

Descarga de datos de la estación

En el menú principal deberá ser pulsado el botón "Bajar datos", se desplegará un menú con un rango de fechas (Ilustración 46), en "inicio" se debe seleccionar la fecha del primer dato que queremos descargar del servidor y en "fin" la del último. Al pulsar "download" se descargarán del servidor los ficheros de datos que contengan el intervalo seleccionado, guardándose automáticamente en la carpeta "data" del programa.

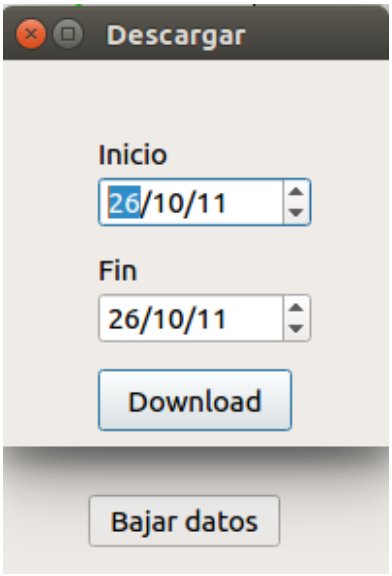

Ilustración 46: Menú de descarga de datos de la estación

Visualización de otras estaciones a través del NMDB

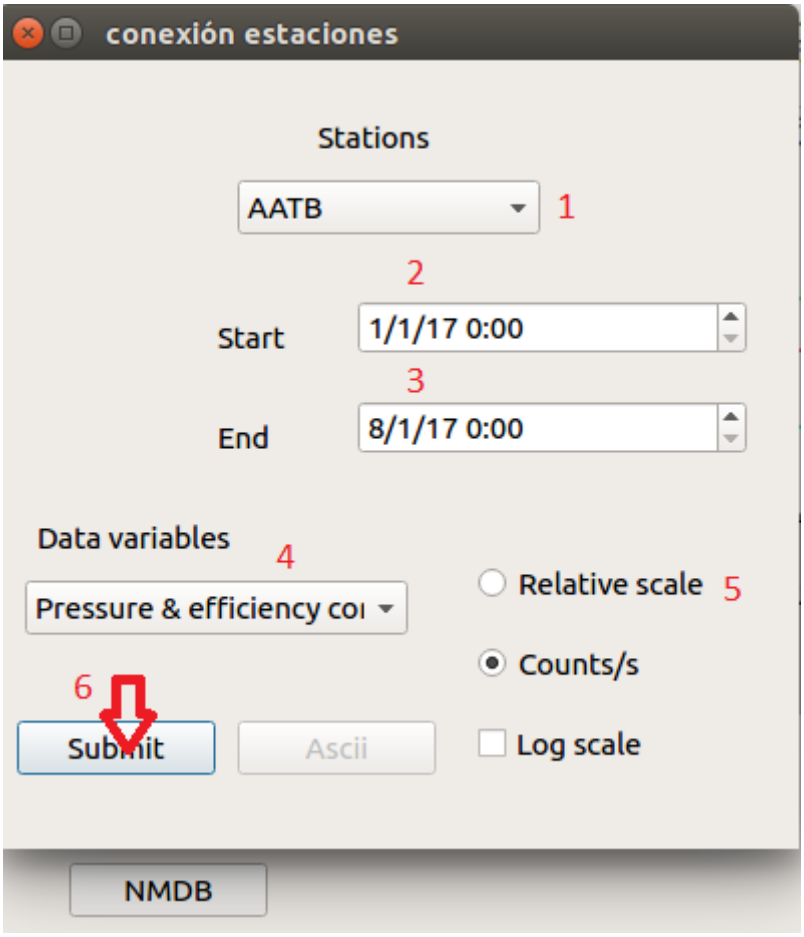

Ilustración 47: Menú de descarga de datos del NMDB

Presionando en el botón NMDB, aparecerá un menú que permite descargar los datos de otras estaciones a través de la base de datos de NMDB. (Ilustración 47)

"Station"(1) es un menú desplegable en el que se elige la estación de la que se requieren los datos.

Introduciendo una fecha válida en "Start"(2) y "End"(3) se selecciona el intervalo de datos que queremos descargar.

En "Datas variables"(4) se elige cómo de tratados están los datos que se requieren de la estación.

Por último, se ha de seleccionar si los datos vendrán en escala relativa o en cuentas partido segundo, seleccionando entre "Relative Scale"(5) y "Counts/s"

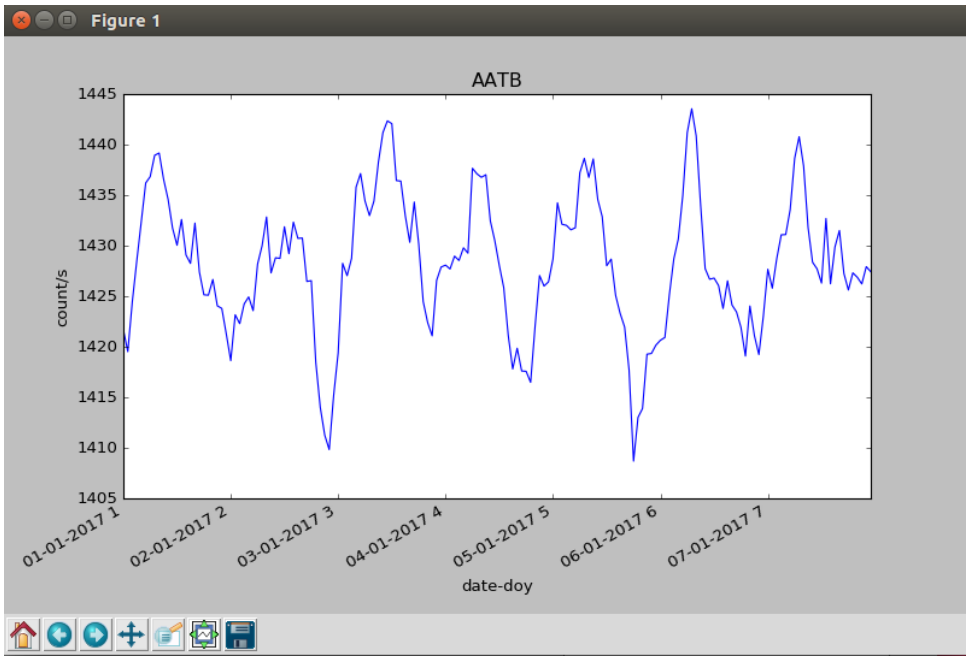

Ilustración 48:Gráfica con los datos de una estación del NMDB

Si todo se ha realizado correctamente al pulsar el botón "Submit"(6) se descargarán los datos de la estación correspondiente visualizándose en una gráfica (Ilustración 48) y se activará el botón "Ascii" (Ilustración 49) si se desea almacenar los datos en un fichero en texto plano.

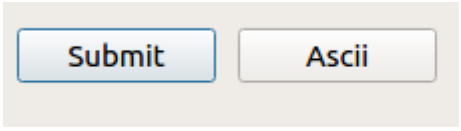

Ilustración 49:Botones habilitados

Módulos de Python

Pandas

Pandas [8] es un módulo creada para facilitar el trabajo con estructuras de datos.

Trabaja con datos tabulados en columnas, series temporales ordenadas y desordenadas, matrices de datos arbitrarios con filas y columnas, y datos estadísticos.

Tiene dos tipos de estructuras principales, las series para una dimensión y los dataframes para dos dimensiones. Los dataframes están basados en los del lenguaje R, por lo que trata de contener todas sus funcionalidades más alguna adicional gracias a que está construida sobre el módulo Numpy. (Véase Numpy).

Como características principales tiene que: Representa los datos perdidos como NaN (Not a number), facilita la opción de borrar e insertar columnas de los Dataframes, permite agrupamiento de datos por patrones en común, indexado de datos automáticos, lectura y escritura de ficheros planos, CSV, Excel, bases de datos y HDF5, funciones específicas para series temporales y uniones sencillas de secuencias de datos.

Numpy

Numpy [9] es un módulo para cálculos matemáticos y científicos, los datos se almacenan en un array N-dimensional, que contiene una colección de ítems de un mismo tipo.

El array es homogéneo, cada ítem ocupa el mismo tamaño de memoria.

Es al uso un módulo matemático basado en vectores y matrices como pudiera suceder con la de Matlab, pero libre y para el lenguaje Python.

BeautifulSoup

BeautifulSoup [10] [11] es un módulo de Python que facilita el análisis de web HTML. Permite extraer con facilidad fragmentos de código de páginas web, modificarlos y codificarlos.

Se desarrolló con el objetivo de hacer la vida más fácil a los programadores que quisieran extraer y analizar partes de páginas web en pocas líneas de código.

Según el autor se ha utilizado desde su creación para aplicaciones como extraer puntuaciones deportivas, grabar streamings de radio por internet, bajar datos astronómicos y APIs no oficiales de extracción de datos.

URLLIB [12]

Es un módulo que permite hacer peticiones web en alto nivel a través del lenguaje Python. También incorpora funciones para trabajar con ficheros online como si de ficheros locales se tratasen.

PyQT [13]

Tiene una licencia libre siempre que no se utilice para aplicaciones comerciales. Pertenece a la empresa Riverbank computing.

PyQT es un binding de el módulo de C++, QT.

PyQT contiene más de 620 clases, que permiten, entre otras cosas, diseñar interfaces gráficas, manejar XML, comunicaciones en red, manejo de bases de datos SQL, navegación web entre otras cosas que se pueden hacer en el QT de C++.

Scikit-learn [14]

Módulo para científicos de datos y machine learning. Construido con módulos de NumPy, SciPy y matplotlib. De código abierto y licencia libre comercial.

Entre otras cosas permite clasificación de datos, cálculos de regresiones, reducción dimensional, selección de modelos, preprocesamiento y clustering.

Presupuesto

Como todo el software utilizado es open source o libre, el presupuesto se reduce a las horas de trabajo del programador. Si un programador junior en España cobra 17642 € al año de media [15], el cálculo sería el siguiente:

$$\frac{17642\ Euros\ al\ año}{14\ pagas} = 1260{,}14\ Euros\ al\ mes$$

$$\frac{1260{,}14\ Euros\ al\ mes}{4\ semanas * 5\ días\ laborables} = 63\ Euros\ al\ día$$

$$\frac{63\ Euros\ al\ día}{8\ Horas\ al\ día} = 7{,}87\ Euros/Hora$$

Teniendo en cuenta que el TFG son 12 créditos de los cuales podríamos considerar unos 8 del desarrollo del proyecto y que 1 crédito ECTS son 25 horas tendríamos un total de 8 * 25= 200 Horas.

$$Valor\ total\ del\ proyecto\ =\ 200\ Horas\ *\ 7{,}87\ Euros/Hora\ =\ 1574\ Euros$$

Conclusiones

Partiendo de los datos expuestos durante toda la memoria se ha llegado a las siguientes conclusiones:

- El desarrollo de este Software de visualización ha ahorrado dinero al grupo de investigación.
  Esto se extrae de los precios que costaba mantener la licencia de IDL y el apartado de "presupuesto". Si nos fijamos en este último, se puede observar que el gasto no es elevado y además en este caso ha pasado a coste cero debido a que formaba parte de un TFG, por otra parte, excluyendo el valor de la licencia de IDL y pensando exclusivamente en el valor del capital humano, el uso de este lenguaje en lugar de Python lo habría encarecido por la dificultad de encontrar un programador familiarizado con el lenguaje.

- Utilizar un lenguaje de programación de propósito general y libre como Python enriquece las posibilidades de lo que se puede llegar a hacer con el proyecto.
  Si leemos el apartado dedicado a CaLMante, al nuevo software y a IDL vs Python, se concluye que el uso de módulos abiertos(bibliotecas), la amplia comunidad que anda detrás del desarrollo del lenguaje y la gran cantidad de documentación que existe, facilita que este visualizador no se limite exclusivamente a visualizar y a calcular a la antigua usanza, si no que sea además es sencillo programarlo para adquirir recursos online, importar a varios formatos y a usar algoritmos de cálculo de forma automática, permitiendo utilizar todo este tiempo ahorrado en centrarse en la investigación.

- Este es solo el principio.
  Observando toda la memoria en general, como conclusión final, se puede decir que el proyecto abre un amplio abanico de posibilidades. Mejorar el Software optimizando los cálculos, añadiendo otro tipo de gráficas y operaciones, además de la importación de datos del NMDB también la exportación de datos del CaLMa a la Neutron Monitor Data Base, incorporación de otro tipo de datos como temperaturas y voltajes, etc.

Bibliografía

[1] «Neutron monitor, Wikipedia,» [En línea]. Available: en.wikipedia.org/wiki/Neutron_monitor. [Último acceso: 19 2 2018].

[2] J. Medina, J. J. Blanco, O. García, R. Gómez-Herrero, E. J. Catalán, I. García, S. Sanchez, J. Rodriguez-Pacheco, M. Prieto, M. A. Hidalgo y D. Mezia, «Castilla-La Mancha neutron monitor,» *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 727, pp. 97-103, 2013.

[3] Research Systems, Inc., «Getting started with IDL,» September 2000. [En línea]. Available: http://astro.uni-wuerzburg.de/~mshupp/IDL/IDL_docs/getstart.pdf.

[4] Python Software Foundation, «Tutorial de Python,» 2017. [En línea]. Available: http://docs.python.org.ar/tutorial/3/real-index.html.

[5] «Python, Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Python. [Último acceso: 19 2 2018].

[6] *Apuntes Tema 2 Introducción a la ingeniería de Software, Asignatura Ingenieria de Software, titulación Grado En Ingeniería Informática En Ingeniería Del Software,* Universidad de Extremadura, 2017.

[7] S. Tosi, Matplotlib for Python Devolopers, Packt Publishing, 2009.

[8] Wes McKinney & PyData Development Team, «pandas: powerful Python data analysis,» 30 Diciembre 2017. [En línea]. Available: http://pandas.pydata.org/pandas-docs/stable/pandas.pdf.

[9] NumPy community, «NumPy Reference,» 29 Mayo 2016. [En línea]. Available: https://docs.scipy.org/doc/numpy-1.11.0/numpy-ref-1.11.0.pdf.

[10] L. Richardson, «Tool Safety,» 2017.

[11] L. Richardson, «Beautiful Soup,» [En línea]. Available: https://www.crummy.com/software/BeautifulSoup/.

[12] Python Software Foundation, «urllib — Open arbitrary resources by URL,» [En línea]. Available: https://docs.python.org/2/library/urllib.html.

[13] «PyQT, wikipython,» [En línea]. Available: https://wiki.python.org/moin/PyQt. [Último acceso: 19 2 2018].

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot y Duchesna, «Scikit-learn: Machine Learning in Python,» *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[15] Indeed, «Sueldos en Programador/a junior en España,» [En línea]. Available: https://www.indeed.es/salaries/Programador/a-junior-Salaries?period=yearly.

Indice de ilustraciones

Indice de tablas

Anexo I: Código fuente.

Fichero de interfaz y carga de datos:

```
import matplotlib.dates as mdates
import pandas as pd
import individual
import histograma
import histogramacentrado as histcent
import median
import menusorbete
import menuinternet
import corte
import errores
import relativo
import corpresion
import presionfit
import presiontiempo
import time
from mplwidget import MplWidget
from PyQt5 import QtCore, QtGui, QtWidgets


class Ui_MainWindow(object):
    def setupUi(self, MainWindow):
        MainWindow.setObjectName("MainWindow")
        MainWindow.resize(1000, 700)
        self.centralwidget = QtWidgets.QWidget(MainWindow)
        self.centralwidget.setObjectName("centralwidget")
        self.tabWidget = QtWidgets.QTabWidget(self.centralwidget)
        self.tabWidget.setGeometry(QtCore.QRect(690, 0, 271, 650))
        self.tabWidget.setObjectName("tabWidget")
        self.tab = QtWidgets.QWidget()
        self.tab.setObjectName("tab")
        self.startlabel = QtWidgets.QLabel(self.tab)
        self.startlabel.setGeometry(QtCore.QRect(10, 70, 67, 17))
        self.startlabel.setObjectName("startlabel")
        self.endlabel = QtWidgets.QLabel(self.tab)
        self.endlabel.setGeometry(QtCore.QRect(10, 140, 67, 17))
        self.endlabel.setObjectName("endlabel")
        self.boton_start = QtWidgets.QPushButton(self.tab)
        self.boton_start.setGeometry(QtCore.QRect(10, 190, 99, 27))
        self.boton_start.setObjectName("boton_start")
        self.boton_ascii = QtWidgets.QPushButton(self.tab)
        self.boton_ascii.setGeometry(QtCore.QRect(150, 190, 99, 27))
        self.boton_ascii.setObjectName("boton_ascii")
        self.boton_ascii.setEnabled(False)
        self.fechainicio = QtWidgets.QDateEdit(self.tab)
        self.fechainicio.setGeometry(QtCore.QRect(60, 60, 110, 27))
        self.fechainicio.setMinimumDate(QtCore.QDate(2011, 10, 1))
        self.fechainicio.setDate(QtCore.QDate(2011, 10, 26))
        self.fechainicio.setObjectName("fechainicio")
        self.fechafin = QtWidgets.QDateEdit(self.tab)
        self.fechafin.setGeometry(QtCore.QRect(60, 130, 110, 27))
        self.fechafin.setMinimumDate(QtCore.QDate(2011, 10, 1))
        self.fechafin.setDate(QtCore.QDate(2011, 10, 26))
        self.fechafin.setObjectName("fechafin")
        self.tabWidget.addTab(self.tab, "")
        self.tab_2 = QtWidgets.QWidget()
        self.tab_2.setObjectName("tab_2")
        self.tipolabel = QtWidgets.QLabel(self.tab_2)
        self.tipolabel.setGeometry(QtCore.QRect(20, 10, 91, 17))
        self.tipolabel.setObjectName("tipolabel")
        self.radioButton_presion = QtWidgets.QRadioButton(self.tab_2)
        self.radioButton_presion.setGeometry(QtCore.QRect(0, 70, 161, 22))
        self.radioButton_presion.setAutoExclusive(False)
        self.radioButton_presion.setObjectName("radioButton_presion")
        self.radioButton_Escala = QtWidgets.QRadioButton(self.tab_2)
        self.radioButton_Escala.setGeometry(QtCore.QRect(0, 120, 131, 22))
        self.radioButton_Escala.setAutoExclusive(False)
        self.radioButton_Escala.setObjectName("radioButton_Escala")
        self.radioButton_medianaportubo = QtWidgets.QRadioButton(self.tab_2)
        self.radioButton_medianaportubo.setGeometry(QtCore.QRect(0, 170, 191, 22))
        self.radioButton_medianaportubo.setAutoExclusive(False)
        self.radioButton_medianaportubo.setObjectName("radioButton_medianaportubo")
        self.radioButton_datosmalos = QtWidgets.QRadioButton(self.tab_2)
```

```
self.radioButton_datosmalos.setGeometry(QtCore.QRect(0, 190, 191, 22))
self.radioButton_datosmalos.setAutoExclusive(False)
self.radioButton_datosmalos.setObjectName("radioButton_datosmalos")
self.radioButton_autoy = QtWidgets.QRadioButton(self.tab_2)
self.radioButton_autoy.setGeometry(QtCore.QRect(0, 210, 117, 22))
self.radioButton_autoy.setAutoExclusive(False)
self.radioButton_autoy.setObjectName("radioButton_autoy")
self.tiempolabel = QtWidgets.QLabel(self.tab_2)
self.tiempolabel.setGeometry(QtCore.QRect(20, 270, 101, 17))
self.tiempolabel.setObjectName("tiempolabel")
self.selectormediatiempo = QtWidgets.QComboBox(self.tab_2)
self.selectormediatiempo.setGeometry(QtCore.QRect(20, 330, 85, 27))
self.selectormediatiempo.setObjectName("selectormediatiempo")
self.selectormediatiempo.addItem("")
self.selectormediatiempo.addItem("")
self.selectormediatiempo.addItem("")
self.selectormediatiempo.addItem("")
self.selectormediatiempo.addItem("")
self.selectormediatiempo.addItem("")
self.selectormodo = QtWidgets.QComboBox(self.tab_2)
self.selectormodo.setGeometry(QtCore.QRect(0, 40, 211, 27))
font = QtGui.QFont()
font.setPointSize(8)
font.setBold(True)
font.setWeight(75)
self.selectormodo.setFont(font)
self.selectormodo.setObjectName("selectormodo")
self.selectormodo.addItem("")
self.selectormodo.addItem("")
self.selectormodo.addItem("")
self.selectormodo.addItem("")
self.selectormodo.addItem("")
self.selectormodo.addItem("")
self.relmod = QtWidgets.QComboBox(self.tab_2)
self.relmod.setGeometry(QtCore.QRect(20, 140, 121, 27))
self.relmod.setObjectName("relmod")
self.relmod.addItem("")
self.relmod.addItem("")
self.presmod = QtWidgets.QComboBox(self.tab_2)
self.presmod.setGeometry(QtCore.QRect(20, 90, 131, 27))
self.presmod.setObjectName("presmod")
self.presmod.addItem("")
self.presmod.addItem("")
self.listWidget = QtWidgets.QListWidget(self.tab_2)
self.listWidget.setGeometry(QtCore.QRect(140, 240, 121, 351))
self.listWidget.setSelectionMode(QtWidgets.QAbstractItemView.MultiSelection)
self.listWidget.setObjectName("listWidget")
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
```

```
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        self.botocalcular = QtWidgets.QPushButton(self.tab_2)
        self.botocalcular.setGeometry(QtCore.QRect(20, 400, 99, 27))
        self.botocalcular.setObjectName("botocalcular")
        self.tabWidget.addTab(self.tab_2, "")
        self.widget_graficas = MplWidget(self.centralwidget)
        self.widget_graficas.setGeometry(QtCore.QRect(0, 10, 680, 550))
        self.widget_graficas.setObjectName("widget_graficas")
        self.boton_inet = QtWidgets.QPushButton(self.centralwidget)
        self.boton_inet.setGeometry(QtCore.QRect(40, 580, 99, 27))
        self.boton_inet.setObjectName("boton_inet")
        self.boton_down = QtWidgets.QPushButton(self.centralwidget)
        self.boton_down.setGeometry(QtCore.QRect(520, 580, 99, 27))
        self.boton_down.setObjectName("boton_down")
        MainWindow.setCentralWidget(self.centralwidget)
        self.menubar = QtWidgets.QMenuBar(MainWindow)
        self.menubar.setGeometry(QtCore.QRect(0, 0, 1000, 25))
        self.menubar.setObjectName("menubar")
        MainWindow.setMenuBar(self.menubar)
        self.statusbar = QtWidgets.QStatusBar(MainWindow)
        self.statusbar.setObjectName("statusbar")
        MainWindow.setStatusBar(self.statusbar)
        self.tabWidget.removeTab(1)

        self.retranslateUi(MainWindow)
        self.tabWidget.setCurrentIndex(1)
        self.boton_start.clicked.connect(self.carga)
        self.boton_ascii.clicked.connect(self.ascii)
        self.boton_down.clicked.connect(menusorbete.llamada)
        self.boton_inet.clicked.connect(self.inet)
        self.botocalcular.clicked.connect(self.calculos)
        QtCore.QMetaObject.connectSlotsByName(MainWindow)

    def carga(self):
        # Comparamos si la fecha es valida
        if self.fechainicio.date() <= self.fechafin.date():
            self.tabWidget.addTab(self.tab_2, "Plot")
            self.df1 = []
            for i in range(self.fechainicio.date().year(),
                           self.fechafin.date().year() + 1):
                if self.fechainicio.date().year() == self.fechafin.date().year():
                    for j in range(self.fechainicio.date().month(),
                                   self.fechafin.date().month() + 1):
                        df = pd.read_csv('data/' + str(i) + str(j).zfill(2) +
'_calma.dat', sep=' ',
                                         header=None, skiprows=list(range(0, 1)),
                                         index_col=[0],
                                         parse_dates=[[0, 1, 2, 3, 4, 5]],
                                         date_parser=lambda x: pd.datetime.strptime(x,
'%Y %m %d %H %M %S'))
                        self.df1.append(df)
                elif self.fechainicio.date().year() == i:
                    for j in range(self.fechainicio.date().month(), 12 + 1):
                        df = pd.read_csv('data/' + str(i) + str(j).zfill(2) +
'_calma.dat', sep=' ',
                                         header=None, skiprows=list(range(0, 1)),
                                         index_col=[0],
                                         parse_dates=[[0, 1, 2, 3, 4, 5]],
                                         date_parser=lambda x: pd.datetime.strptime(x,
'%Y %m %d %H %M %S'))
                        self.df1.append(df)
                elif self.fechainicio.date().year() != i and i !=
self.fechafin.date().year():
                    for j in range(1, 12 + 1):
                        df = pd.read_csv('data/' + str(i) + str(j).zfill(2) +
'_calma.dat', sep=' ',
                                         header=None, skiprows=list(range(0, 1)),
                                         index_col=[0],
                                         parse_dates=[[0, 1, 2, 3, 4, 5]],
                                         date_parser=lambda x: pd.datetime.strptime(x,
'%Y %m %d %H %M %S'))
                        self.df1.append(df)
```

```
                else:
                    for j in range(1, self.fechafin.date().month() + 1):
                        df = pd.read_csv('data/' + str(i) + str(j).zfill(2) +
' calma.dat', sep=' ',
                                         header=None, skiprows=list(range(0, 1)),
                                         index_col=[0],
                                         parse_dates=[[0, 1, 2, 3, 4, 5]],
                                         date_parser=lambda x: pd.datetime.strptime(x,
'%Y %m %d %H %M %S'))
                        self.df1.append(df)
            self.df1 = pd.concat(self.df1, axis=0)
            if self.fechainicio.date().toString("yyyyMMdd") <= ("20120608"):
                diaini = self.fechainicio.date().toString("yyyyMMdd")
                diacambio = ("20120608003400")
                aaa = self.df1[18].loc[diaini:diacambio]
                bbb = self.df1[19].loc[diaini:diacambio]
                ccc = self.df1[20].loc[diaini:diacambio]
                ddd = self.df1[23].loc[diaini:diacambio]
                self.df1[18].loc[diaini:diacambio] = self.df1[6].loc[diaini:diacambio]
                self.df1[19].loc[diaini:diacambio] = self.df1[7].loc[diaini:diacambio]
                self.df1[20].loc[diaini:diacambio] = self.df1[8].loc[diaini:diacambio]
                self.df1[23].loc[diaini:diacambio] = self.df1[11].loc[diaini:diacambio]
                self.df1[6].loc[diaini:diacambio] = aaa
                self.df1[7].loc[diaini:diacambio] = bbb
                self.df1[8].loc[diaini:diacambio] = ccc
                self.df1[11].loc[diaini:diacambio] = ddd
            self.nocor = corte.nocor(self.df1)
            self.pvarnocor = corpresion.corre(self.nocor)
            self.pfijnocor = corpresion.fija(self.nocor)
            self.badata = errores.baddatasn(self.nocor)
            self.badatapv = errores.baddatasn(self.pvarnocor)
            self.badatapf = errores.baddatasn(self.pfijnocor)
            self.botocalcular.setEnabled(True)
        else:
            import dialogo
            dialog = dialogo.Error()
            value = dialog.exec_()
            print ("fecha no valida")
            self.botocalcular.setEnabled(False)
            print (self.fechainicio.date())
            print (self.fechafin.date())

    def calculos(self):
        self.mihilo = hilo()
        self.mihilo.start()

    def inet(self):
        # https://stackoverflow.com/questions/1807299/open-a-second-window-in-pyqt
        self.dialogoinet = QtWidgets.QDialog()
        self.dialogoinet.ui = menuinternet.Ui_Dialog()
        self.dialogoinet.ui.setupUi(self.dialogoinet)
        self.dialogoinet.setAttribute(QtCore.Qt.WA_DeleteOnClose)
        self.dialogoinet.show()


# Funcion que representa segun las opciones
    def representaciones(self):
        self.boton_ascii.setEnabled(True)
        self.botocalcular.setEnabled(False)
        diaini = self.fechainicio.date().toString("yyyyMMdd")
        diafin = self.fechafin.date().toString("yyyyMMdd")
        meditubo = self.radioButton_medianaportubo.isChecked()
        rel = self.radioButton_Escala.isChecked()
        dmalos = self.radioButton_datosmalos.isChecked()
        press = self.radioButton_presion.isChecked()
        ax = self.widget_graficas.canvas.ax
        widget = self.listWidget
        self.widget_graficas.canvas.ax.clear()
        daf = self.nocor
        if self.selectormediatiempo.currentIndex() == 0:
            promedio = '1T'
        elif self.selectormediatiempo.currentIndex() == 1:
            promedio = '5T'
        elif self.selectormediatiempo.currentIndex() == 2:
            promedio = '30T'
        elif self.selectormediatiempo.currentIndex() == 3:
            promedio = '60T'
```

```
        elif self.selectormediatiempo.currentIndex() == 4:
            promedio = '1440T'
        elif self.selectormediatiempo.currentIndex() == 5:
            promedio = '38880T'
        if dmalos:
            if press:
                if self.presmod.currentIndex() == 0:
                    daf = self.badatapv
                elif self.presmod.currentIndex() == 1:
                    daf = self.badatapf
            else:
                daf = self.badata
        elif press:
            if self.presmod.currentIndex() == 0:
                    daf = self.pvarnocor
            elif self.presmod.currentIndex() == 1:
                    daf = self.pfijnocor
        else:
            daf = self.nocor
        if not self.listWidget.selectedItems():
            self.widget_graficas.canvas.ax.clear()
            if self.selectormodo.currentIndex() == 5:
                self.dfimport = presiontiempo.pretime(widget, ax, daf,
                                                      diaini, diafin, promedio)
                self.widget_graficas.canvas.fig.autofmt_xdate()
        elif self.selectormodo.currentIndex() == 0:
            if rel:
                if self.relmod.currentIndex() == 0:
                    daf = relativo.relativo(daf)
                elif self.relmod.currentIndex() == 1:
                    daf = relativo.fijo(daf)
            self.dfimport = median.mediana(widget, ax, daf, diaini,
                                           diafin, meditubo, rel, promedio)
            self.widget_graficas.canvas.fig.autofmt_xdate()
        elif self.selectormodo.currentIndex() == 1:
            if rel:
                if self.relmod.currentIndex() == 0:
                    daf = relativo.relativo(daf)
                elif self.relmod.currentIndex() == 1:
                    daf = relativo.fijo(daf)
            self.dfimport = individual.rateindividual(widget, ax, daf, diaini,
                                                      diafin, rel, promedio)
            self.widget_graficas.canvas.fig.autofmt_xdate()
        elif self.selectormodo.currentIndex() == 2:
            self.dfimport = presionfit.pf(widget, ax, self.nocor, diaini,
                                          diafin, meditubo, rel, promedio)
        elif self.selectormodo.currentIndex() == 3:
            self.dfimport = histograma.histograma(widget, ax,
                                                  daf, diaini, diafin)
        elif self.selectormodo.currentIndex() == 4:
            self.dfimport = histcent.histogramacentrado(widget, ax,
                                                        daf, diaini, diafin)
        elif self.selectormodo.currentIndex() == 5:
            self.dfimport = presiontiempo.pretime(widget, ax, daf,
                                                  diaini, diafin, promedio)
            self.widget_graficas.canvas.fig.autofmt_xdate()
        if self.radioButton_autoy.isChecked():
            ax.autoscale(enable=True, axis='y', tight=True)
        self.widget_graficas.canvas.draw()
        self.widget_graficas.canvas.show()
        self.botocalcular.setEnabled(True)

    def ascii(self):
        fileName, _ = QtWidgets.QFileDialog.getSaveFileName()
        if fileName:
            self.dfimport.to_csv(fileName, sep=";", encoding='ascii')

    def retranslateUi(self, MainWindow):
        _translate = QtCore.QCoreApplication.translate
        MainWindow.setWindowTitle(_translate("MainWindow", "MainWindow"))
        self.startlabel.setText(_translate("MainWindow", "Start"))
        self.endlabel.setText( translate("MainWindow", "End"))
        self.boton_start.setText(_translate("MainWindow", "Start"))
        self.boton_ascii.setText(_translate("MainWindow", "ASCII"))
        self.boton_inet.setText(_translate("MainWindow", "NMDB"))
        self.boton_down.setText(_translate("MainWindow", "Bajar datos"))
```

```
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab),
_translate("MainWindow", "File"))
        self.tipolabel.setText(_translate("MainWindow", "Type of plot"))
        self.radioButton_presion.setText(_translate("MainWindow", "Pressure
correction"))
        self.radioButton_Escala.setText(_translate("MainWindow", "Relative scale"))
        self.radioButton_medianaportubo.setText(_translate("MainWindow", "Show medians
per tube"))
        self.radioButton_datosmalos.setText(_translate("MainWindow", "Filter obvious bad
data"))
        self.radioButton_autoy.setText(_translate("MainWindow", "Auto y-range"))
        self.tiempolabel.setText(_translate("MainWindow", "Time average"))
        self.selectormediatiempo.setItemText(0, _translate("MainWindow", "1 min"))
        self.selectormediatiempo.setItemText(1, _translate("MainWindow", "5 min"))
        self.selectormediatiempo.setItemText(2, _translate("MainWindow", "30 min"))
        self.selectormediatiempo.setItemText(3,  translate("MainWindow", "1 H"))
        self.selectormediatiempo.setItemText(4, _translate("MainWindow", "1 day"))
        self.selectormediatiempo.setItemText(5, _translate("MainWindow", "27 days"))
        self.selectormodo.setItemText(0, _translate("MainWindow", "Rate - Median of
selected channel"))
        self.selectormodo.setItemText(1, _translate("MainWindow", "Rate - Individual
channel"))
        self.selectormodo.setItemText(2, _translate("MainWindow", "Pressure correction
fit"))
        self.selectormodo.setItemText(3, _translate("MainWindow", "Counting histogram"))
        self.selectormodo.setItemText(4, _translate("MainWindow", "Counting histogram
(shifted)"))
        self.selectormodo.setItemText(5, _translate("MainWindow", "Pressure-time"))
        self.relmod.setItemText(0, _translate("MainWindow", "Real media"))
        self.relmod.setItemText(1, _translate("MainWindow", "Station media"))
        self.presmod.setItemText(0, _translate("MainWindow", "realtime press"))
        self.presmod.setItemText(1, _translate("MainWindow", "ref value"))
        __sortingEnabled = self.listWidget.isSortingEnabled()
        self.listWidget.setSortingEnabled(False)
        item = self.listWidget.item(0)
        item.setText(_translate("MainWindow", "CH01"))
        item = self.listWidget.item(1)
        item.setText(_translate("MainWindow", "CH02"))
        item = self.listWidget.item(2)
        item.setText(_translate("MainWindow", "CH03"))
        item = self.listWidget.item(3)
        item.setText(_translate("MainWindow", "CH04"))
        item = self.listWidget.item(4)
        item.setText(_translate("MainWindow", "CH05"))
        item = self.listWidget.item(5)
        item.setText(_translate("MainWindow", "CH06"))
        item = self.listWidget.item(6)
        item.setText(_translate("MainWindow", "CH07"))
        item = self.listWidget.item(7)
        item.setText(_translate("MainWindow", "CH08"))
        item = self.listWidget.item(8)
        item.setText(_translate("MainWindow", "CH09"))
        item = self.listWidget.item(9)
        item.setText(_translate("MainWindow", "CH10"))
        item = self.listWidget.item(10)
        item.setText(_translate("MainWindow", "CH11"))
        item = self.listWidget.item(11)
        item.setText(_translate("MainWindow", "CH12"))
        item = self.listWidget.item(12)
        item.setText(_translate("MainWindow", "CH13"))
        item = self.listWidget.item(13)
        item.setText(_translate("MainWindow", "CH14"))
        item = self.listWidget.item(14)
        item.setText(_translate("MainWindow", "CH15"))
        item = self.listWidget.item(15)
        item.setText( translate("MainWindow", "CH16"))
        item = self.listWidget.item(16)
        item.setText(_translate("MainWindow", "CH17"))
        item = self.listWidget.item(17)
        item.setText(_translate("MainWindow", "CH18"))
        self.listWidget.setSortingEnabled(__sortingEnabled)
        self.botocalcular.setText( translate("MainWindow", "Calc"))
        self.tabWidget.setTabText(self.tabWidget.indexOf(self.tab_2),
_translate("MainWindow", "Plot"))


# https://nikolak.com/pyqt-threading-tutorial/
```

```
class hilo(QtCore.QThread):

    def __init__(self):
        QtCore.QThread.__init__(self)

    def __del__(self):
        self.wait()

    def run(self):
        ui.representaciones()
        time.sleep(1)


if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    MainWindow = QtWidgets.QMainWindow()
    ui = Ui_MainWindow()
    ui.setupUi(MainWindow)
    MainWindow.show()
    sys.exit(app.exec_())
```

Representación de canales:

```
import matplotlib.dates as mdates
from matplotlib.font_manager import FontProperties


# Funcion para visualizar cada canal de forma independiente
def rateindividual(wisel, ax, df, DIni, DFin, rel, promedio):
    # Para promedio de 1
    if promedio is '1T':
        union = []
        # Se selecciona cada canal y se divide entre 60(count/s)
        if wisel.item(0).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch01'].loc[DIni:DFin] / 60.,
                    label='Ch1', color='#ff0000')
            union.append('ch01')
        if wisel.item(1).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch02'].loc[DIni:DFin] / 60.,
                    label='Ch2', color='#009eff')
            union.append('ch02')
        if wisel.item(2).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch03'].loc[DIni:DFin] / 60.,
                    label='Ch3', color='#ff00bd')
            union.append('ch03')
        if wisel.item(3).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch04'].loc[DIni:DFin] / 60.,
                    label='Ch4', color='#ffd800')
            union.append('ch04')
        if wisel.item(4).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch05'].loc[DIni:DFin] / 60.,
                    label='Ch5', color='#00ff3a')
            union.append('ch05')
        if wisel.item(5).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch06'].loc[DIni:DFin] / 60.,
                    label='Ch6', color='#4c764e')
            union.append('ch06')
        if wisel.item(6).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch07'].loc[DIni:DFin] / 60.,
                    label='Ch7', color='#2b3b63')
            union.append('ch07')
        if wisel.item(7).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch08'].loc[DIni:DFin] / 60.,
```

```
                label='Ch8', color='#4e3815')
            union.append('ch08')
        if wisel.item(8).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch09'].loc[DIni:DFin] / 60.,
                    label='Ch9', color='#ff9d00')
            union.append('ch09')
        if wisel.item(9).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch10'].loc[DIni:DFin] / 60.,
                    label='Ch10', color='#00ffe8')
            union.append('ch10')
        if wisel.item(10).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch11'].loc[DIni:DFin] / 60.,
                    label='Ch11', color='#fabada')
            union.append('ch11')
        if wisel.item(11).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch12'].loc[DIni:DFin] / 60.,
                    label='Ch12', color='#6e60a0')
            union.append('ch12')
        if wisel.item(12).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch13'].loc[DIni:DFin] / 60.,
                    label='Ch13', color='#77eb44')
            union.append('ch13')
        if wisel.item(13).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch14'].loc[DIni:DFin] / 60.,
                    label='Ch14', color='#a53030')
            union.append('ch14')
        if wisel.item(14).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch15'].loc[DIni:DFin] / 60.,
                    label='Ch15', color='#734061')
            union.append('ch15')
        if wisel.item(15).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch16'].loc[DIni:DFin] / 60.,
                    label='Ch16', color='#345c06')
            union.append('ch16')
        if wisel.item(16).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch17'].loc[DIni:DFin] / 60.,
                    label='Ch17', color='#4c4b50')
            union.append('ch17')
        if wisel.item(17).isSelected():
            ax.plot(df.loc[DIni:DFin].index,
                    df['ch18'].loc[DIni:DFin] / 60.,
                    label='Ch18', color='#1bff00')
            union.append('ch18')
            # ax.hist(df[7].values)
    # Para promediados se hace asi:
    else:
        union = []
        if wisel.item(0).isSelected():
            ax.plot(df['ch01'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch1', color='#ff0000')
            union.append('ch01')
        if wisel.item(1).isSelected():
            ax.plot(df['ch02'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch2', color='#009eff')
            union.append('ch02')
        if wisel.item(2).isSelected():
            ax.plot(df['ch03'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch3', color='#ff00bd')
            union.append('ch03')
        if wisel.item(3).isSelected():
            ax.plot(df['ch04'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch4', color='#ffd800')
            union.append('ch04')
        if wisel.item(4).isSelected():
```

```
            ax.plot(df['ch05'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch5', color='#00ff3a')
            union.append('ch05')
        if wisel.item(5).isSelected():
            ax.plot(df['ch06'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch6', color='#4c764e')
            union.append('ch06')
        if wisel.item(6).isSelected():
            ax.plot(df['ch07'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch7', color='#2b3b63')
            union.append('ch07')
        if wisel.item(7).isSelected():
            ax.plot(df['ch08'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch8', color='#4e3815')
            union.append('ch08')
        if wisel.item(8).isSelected():
            ax.plot(df['ch09'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch9', color='#ff9d00')
            union.append('ch09')
        if wisel.item(9).isSelected():
            ax.plot(df['ch10'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch10', color='#00ffe8')
            union.append('ch10')
        if wisel.item(10).isSelected():
            ax.plot(df['ch11'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch11', color='#fabada')
            union.append('ch11')
        if wisel.item(11).isSelected():
            ax.plot(df['ch12'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch12', color='#6e60a0')
            union.append('ch12')
        if wisel.item(12).isSelected():
            ax.plot(df['ch13'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch13', color='#77eb44')
            union.append('ch13')
        if wisel.item(13).isSelected():
            ax.plot(df['ch14'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch14', color='#a53030')
            union.append('ch14')
        if wisel.item(14).isSelected():
            ax.plot(df['ch15'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch15', color='#734061')
            union.append('ch15')
        if wisel.item(15).isSelected():
            ax.plot(df['ch16'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch16', color='#345c06')
            union.append('ch16')
        if wisel.item(16).isSelected():
            ax.plot(df['ch17'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch17', color='#4c4b50')
            union.append('ch17')
        if wisel.item(17).isSelected():
            ax.plot(df['ch18'].loc[DIni:DFin]
                    .dropna().resample(promedio, how='mean') / 60.,
                    label='Ch18', color='#1bff00')
            union.append('ch18')

    # Formateado, nombrado, etc de ejes y leyenda
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%d-%m-%Y %-j'))
    fontP = FontProperties()
    fontP.set_size('small')
    ax.legend(bbox_to_anchor=(1, 1), loc=2, borderaxespad=0.,
              prop=fontP, frameon=False)
    if rel:
```

```
        ax.set_ylabel('Variation rate %')
    else:
        ax.set_ylabel('count/s')
        ax.set_xlabel('date-doy')
    if promedio is '1T':
        return df[union].loc[DIni:DFin] / 60
    else:
        return df[union].loc[DIni:DFin].dropna().resample(promedio, how='mean') / 60
```

Mediana:

```
from matplotlib.font_manager import FontProperties
import pandas as pd
import matplotlib.dates as mdates


def mediana(wisel, ax, df, DIni,
            DFin, meditubo, rel, promedio):
    # Hacemos una criba con los canales disponibles
    i = 0
    dfm = pd.DataFrame(index=df.index)
    if wisel.item(0).isSelected():
        dfm['ch01'] = df['ch01']
        i += 1
    if wisel.item(1).isSelected():
        dfm['ch02'] = df['ch02']
        i += 1
    if wisel.item(2).isSelected():
        dfm['ch03'] = df['ch03']
        i += 1
    if wisel.item(3).isSelected():
        dfm['ch04'] = df['ch04']
        i += 1
    if wisel.item(4).isSelected():
        dfm['ch05'] = df['ch05']
        i += 1
    if wisel.item(5).isSelected():
        dfm['ch06'] = df['ch06']
        i += 1
    if wisel.item(6).isSelected():
        dfm['ch07'] = df['ch07']
        i += 1
    if wisel.item(7).isSelected():
        dfm['ch08'] = df['ch08']
        i += 1
    if wisel.item(8).isSelected():
        dfm['ch09'] = df['ch09']
        i += 1
    if wisel.item(9).isSelected():
        dfm['ch10'] = df['ch10']
        i += 1
    if wisel.item(10).isSelected():
        dfm['ch11'] = df['ch11']
        i += 1
    if wisel.item(11).isSelected():
        dfm['ch12'] = df['ch12']
        i += 1
    if wisel.item(12).isSelected():
        dfm['ch13'] = df['ch13']
        i += 1
    if wisel.item(13).isSelected():
        dfm['ch14'] = df['ch14']
        i += 1
    if wisel.item(14).isSelected():
        dfm['ch15'] = df['ch15']
        i += 1
    if wisel.item(15).isSelected():
        dfm['ch16'] = df['ch16']
        i += 1
    if wisel.item(16).isSelected():
        dfm['ch17'] = df['ch17']
        i += 1
```

```
    if wisel.item(17).isSelected():
        dfm['ch18'] = df['ch18']
        i += 1
    # calculamos la mediana para un promedio de 1
    if promedio is '1T':
        # multiplicamos por el numero de tubos en el caso de que no
        # sea mediana por tubo o %
        if meditubo or rel:
            mediana = dfm.loc[DIni:DFin].T.median()
        else:
            mediana = dfm.loc[DIni:DFin].T.median() * i
        ax.plot(dfm.loc[DIni:DFin].index,
                mediana)
        # calculamos para promediado
    else:
        if meditubo or rel:
            mediana = dfm.loc[DIni:DFin].dropna().resample(promedio,
                                                           how='mean').T.median()
        else:
            mediana = dfm.loc[DIni:DFin].dropna().resample(promedio,
                                                           how='mean').T.median() * i
        ax.plot(mediana)
        # formateamos la grafica y la leyenda
    ax.xaxis.set_major_formatter(mdates.DateFormatter('%d-%m-%Y %-j'))
    fontP = FontProperties()
    fontP.set_size('small')
    ax.legend(bbox_to_anchor=(1, 1), loc=2, borderaxespad=0.,
              prop=fontP, frameon=False)
    if rel:
        ax.set_ylabel('Variation rate %')
    else:
        ax.set_ylabel('median count/min')
    ax.set_xlabel('date-doy')
    return mediana
```

Histograma:

```
from matplotlib.font_manager import FontProperties
import pandas as pd
import numpy as np
from scipy.stats import norm


# Aqui se guarda la función del histograma
# Se selecciona cada canal, se localiza inicio y fin
# y se traza su histograma
def histograma(wisel, ax, df, DIni, DFin, promedio):
        # Bin = (1000-100)/4 debido a que asi se hacia en calmante
        # Asignamos tambien un color por canal
        union = []
        titulo = []
        if promedio is '1T':
            if wisel.item(0).isSelected():
                (mu, sigma) = norm.fit(df['ch01'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                ax.hist(df['ch01'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch1', color='#ff0000')
                            # Calculamos la media y la sigma
                res = "μ1="+str(round(mu, 2))+" σ1="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch01')
            if wisel.item(1).isSelected():
                (mu, sigma) = norm.fit(df['ch02'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                ax.hist(df['ch02'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch2', color='#009eff')
                res = "μ2="+str(round(mu, 2))+" σ2="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch02')
            if wisel.item(2).isSelected():
                ax.hist(df['ch03'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch3', color='#ff00bd')
```

```
                (mu, sigma) = norm.fit(df['ch03'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ3="+str(round(mu, 2))+" σ3="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch03')
            if wisel.item(3).isSelected():
                ax.hist(df['ch04'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch4', color='#ffd800')
                (mu, sigma) = norm.fit(df['ch04'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ4="+str(round(mu, 2))+" σ4="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch04')
            if wisel.item(4).isSelected():
                ax.hist(df['ch05'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch5', color='#00ff3a')
                (mu, sigma) = norm.fit(df['ch05'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ5="+str(round(mu, 2))+" σ5="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch05')
            if wisel.item(5).isSelected():
                ax.hist(df['ch06'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch6', color='#4c764e')
                (mu, sigma) = norm.fit(df['ch06'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ6="+str(round(mu, 2))+" σ6="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch06')
            if wisel.item(6).isSelected():
                ax.hist(df['ch07'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch7', color='#2b3b63')
                (mu, sigma) = norm.fit(df['ch07'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ7="+str(round(mu, 2))+" σ7="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch07')
            if wisel.item(7).isSelected():
                ax.hist(df['ch08'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch8', color='#4e3815')
                (mu, sigma) = norm.fit(df['ch08'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ8="+str(round(mu, 2))+" σ8="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch08')
            if wisel.item(8).isSelected():
                ax.hist(df['ch09'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch9', color='#ff9d00')
                (mu, sigma) = norm.fit(df['ch09'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ9="+str(round(mu, 2))+" σ9="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch09')
            if wisel.item(9).isSelected():
                ax.hist(df['ch10'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch10', color='#00ffe8')
                (mu, sigma) = norm.fit(df['ch10'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ10="+str(round(mu, 2))+" σ10="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch10')
            if wisel.item(10).isSelected():
                ax.hist(df['ch11'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch11', color='#fabada')
                (mu, sigma) = norm.fit(df['ch11'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ11="+str(round(mu, 2))+" σ11="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch11')
            if wisel.item(11).isSelected():
                ax.hist(df['ch12'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch12', color='#6e60a0')
                (mu, sigma) = norm.fit(df['ch12'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ12="+str(round(mu, 2))+" σ12="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch12')
```

```
            if wisel.item(12).isSelected():
                ax.hist(df['ch13'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch13', color='#77eb44')
                (mu, sigma) = norm.fit(df['ch13'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ13="+str(round(mu, 2))+" σ13="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch13')
            if wisel.item(13).isSelected():
                ax.hist(df['ch14'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch14', color='#a53030')
                (mu, sigma) = norm.fit(df['ch14'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ14="+str(round(mu, 2))+" σ14="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch14')
            if wisel.item(14).isSelected():
                ax.hist(df['ch15'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch15', color='#734061')
                (mu, sigma) = norm.fit(df['ch15'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ15="+str(round(mu, 2))+" σ15="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch15')
            if wisel.item(15).isSelected():
                ax.hist(df['ch16'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch16', color='#345c06')
                (mu, sigma) = norm.fit(df['ch16'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ16="+str(round(mu, 2))+" σ16="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch16')
            if wisel.item(16).isSelected():
                ax.hist(df['ch17'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch17', color='#4c4b50')
                (mu, sigma) = norm.fit(df['ch17'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ17="+str(round(mu, 2))+" σ17="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch17')
            if wisel.item(17).isSelected():
                ax.hist(df['ch18'].loc[DIni:DFin].dropna().values,
                        bins=225, histtype='step', label='Ch18', color='#1bff00')
                (mu, sigma) = norm.fit(df['ch18'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ18="+str(round(mu, 2))+" σ18="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch18')
        else:
            if wisel.item(0).isSelected():
                ax.hist(df['ch01'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch1', color='#ff0000')
                (mu, sigma) = norm.fit(df['ch01'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ1="+str(round(mu, 2))+" σ1="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch01')
            if wisel.item(1).isSelected():
                ax.hist(df['ch02'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch2', color='#009eff')
                (mu, sigma) = norm.fit(df['ch02'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ2="+str(round(mu, 2))+" σ2="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch02')
            if wisel.item(2).isSelected():
                ax.hist(df['ch03'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch3', color='#ff00bd')
                (mu, sigma) = norm.fit(df['ch03'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ3="+str(round(mu, 2))+" σ3="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch03')
            if wisel.item(3).isSelected():
```

```
                ax.hist(df['ch04'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch4', color='#ffd800')
                (mu, sigma) = norm.fit(df['ch04'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ4="+str(round(mu, 2))+" σ4="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch04')
            if wisel.item(4).isSelected():
                ax.hist(df['ch05'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch5', color='#00ff3a')
                (mu, sigma) = norm.fit(df['ch05'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ5="+str(round(mu, 2))+" σ5="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch05')
            if wisel.item(5).isSelected():
                ax.hist(df['ch06'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch6', color='#4c764e')
                (mu, sigma) = norm.fit(df['ch06'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ6="+str(round(mu, 2))+" σ6="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch06')
            if wisel.item(6).isSelected():
                ax.hist(df['ch07'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch7', color='#2b3b63')
                (mu, sigma) = norm.fit(df['ch07'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ7="+str(round(mu, 2))+" σ7="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch07')
            if wisel.item(7).isSelected():
                ax.hist(df['ch08'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch8', color='#4e3815')
                (mu, sigma) = norm.fit(df['ch08'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ8="+str(round(mu, 2))+" σ8="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch08')
            if wisel.item(8).isSelected():
                ax.hist(df['ch09'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch9', color='#ff9d00')
                (mu, sigma) = norm.fit(df['ch09'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ9="+str(round(mu, 2))+" σ9="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch09')
            if wisel.item(9).isSelected():
                ax.hist(df['ch10'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch10', color='#00ffe8')
                (mu, sigma) = norm.fit(df['ch10'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ10="+str(round(mu, 2))+" σ10="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch10')
            if wisel.item(10).isSelected():
                ax.hist(df['ch11'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch11', color='#fabada')
                (mu, sigma) = norm.fit(df['ch10'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ10="+str(round(mu, 2))+" σ10="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch11')
            if wisel.item(11).isSelected():
                ax.hist(df['ch12'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch12', color='#6e60a0')
                (mu, sigma) = norm.fit(df['ch12'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
```

```
                res = "μ12="+str(round(mu, 2))+" σ12="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch12')
            if wisel.item(12).isSelected():
                ax.hist(df['ch13'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch13', color='#77eb44')
                (mu, sigma) = norm.fit(df['ch13'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ13="+str(round(mu, 2))+" σ13="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch13')
            if wisel.item(13).isSelected():
                ax.hist(df['ch14'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch14', color='#a53030')
                (mu, sigma) = norm.fit(df['ch14'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ14="+str(round(mu, 2))+" σ14="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch14')
            if wisel.item(14).isSelected():
                ax.hist(df['ch15'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch15', color='#734061')
                (mu, sigma) = norm.fit(df['ch15'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ15="+str(round(mu, 2))+" σ15="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch15')
            if wisel.item(15).isSelected():
                ax.hist(df['ch16'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch16', color='#345c06')
                (mu, sigma) = norm.fit(df['ch16'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ16="+str(round(mu, 2))+" σ16="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch16')
            if wisel.item(16).isSelected():
                ax.hist(df['ch17'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch17', color='#4c4b50')
                (mu, sigma) = norm.fit(df['ch17'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ17="+str(round(mu, 2))+" σ17="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch17')
            if wisel.item(17).isSelected():
                ax.hist(df['ch18'].loc[DIni:DFin].resample(promedio,
how='mean').dropna(),
                        bins=225, histtype='step', label='Ch18', color='#1bff00')
                (mu, sigma) = norm.fit(df['ch18'].loc[DIni:DFin].resample(promedio,
how='mean').dropna().values)
                res = "μ18="+str(round(mu, 2))+" σ18="+str(round(sigma, 2))
                titulo.append(res)
                union.append('ch18')
        # Formateamos la grafica y la leyenda
        fontP = FontProperties()
        fontP.set_size('small')
        ax.legend(bbox_to_anchor=(1, 1), loc=2, borderaxespad=0.,
                  prop=fontP, frameon=False)
        ax.set_ylabel('number of events')
        ax.set_xlabel('counts/min')
               #introducimos la mediana y sigma en el grafico
        ax.text(0, 0, '\n'.join(titulo),
                horizontalalignment='left',
                verticalalignment='bottom',
                transform=ax.transAxes, fontsize=7)
        # Returns:
        # hist : array The values of the histogram.
        # bin edges : array of dtype float Return
        # the bin edges (length(hist)+1)
        if promedio is '1T':
            return(df[union].dropna().apply(np.histogram, bins=225))
        else:
```

```
            return(df[union].resample(promedio, how='mean').dropna().apply(np.histogram,
bins=225))
```

Histograma centrado:

```
from matplotlib.font_manager import FontProperties
import numpy as np
import pandas as pd
from scipy.stats import norm


def histogramacentrado(wisel, ax, df, DIni, DFin, promedio):

        # Funcion similar a la de histograma
        # pero centrado restando mean()
        calculo = []
        i = 0
        for i in range(0, 18):
            (mu, sigma) = norm.fit(df['ch' +
str(i+1).zfill(2)].loc[DIni:DFin].resample(promedio, how='mean').dropna().values)
            calculo.append(df['ch' +
str(i+1).zfill(2)].loc[DIni:DFin].resample(promedio, how='mean').dropna() - mu)
        dfcentrado = pd.DataFrame()
        union = []
        if wisel.item(0).isSelected():
            ax.hist(calculo[0],
                    bins=225, histtype='step', label='Ch1', color='#ff0000')
            dfcentrado["ch01"] = calculo[0]
            union.append('ch01')
        if wisel.item(1).isSelected():
            ax.hist(calculo[1],
                    bins=225, histtype='step', label='Ch2', color='#009eff')
            dfcentrado["ch02"] = calculo[1]
            union.append('ch02')
        if wisel.item(2).isSelected():
            ax.hist(calculo[2],
                    bins=225, histtype='step', label='Ch3', color='#ff00bd')
            dfcentrado["ch03"] = calculo[2]
            union.append('ch03')
        if wisel.item(3).isSelected():
            ax.hist(calculo[3],
                    bins=225, histtype='step', label='Ch4', color='#ffd800')
            dfcentrado["ch04"] = calculo[3]
            union.append('ch04')
        if wisel.item(4).isSelected():
            ax.hist(calculo[4],
                    bins=225, histtype='step', label='Ch5', color='#00ff3a')
            dfcentrado["ch05"] = calculo[4]
        if wisel.item(5).isSelected():
            ax.hist(calculo[5],
                    bins=225, histtype='step', label='Ch6', color='#4c764e')
            dfcentrado["ch06"] = calculo[5]
            union.append('ch06')
        if wisel.item(6).isSelected():
            ax.hist(calculo[6],
                    bins=225, histtype='step', label='Ch7', color='#2b3b63')
            dfcentrado["ch07"] = calculo[6]
            union.append('ch07')
        if wisel.item(7).isSelected():
            ax.hist(calculo[7],
                    bins=225, histtype='step', label='Ch8', color='#4e3815')
            dfcentrado["ch08"] = calculo[7]
            union.append('ch08')
        if wisel.item(8).isSelected():
            ax.hist(calculo[8],
                    bins=225, histtype='step', label='Ch9', color='#ff9d00')
            dfcentrado["ch09"] = calculo[8]
            union.append('ch09')
        if wisel.item(9).isSelected():
            ax.hist(calculo[9],
                    bins=225, histtype='step', label='Ch10', color='#00ffe8')
            dfcentrado["ch10"] = calculo[9]
            union.append('ch10')
        if wisel.item(10).isSelected():
```

```
            ax.hist(calculo[10],
                    bins=225, histtype='step', label='Ch11', color='#fabada')
            dfcentrado["ch11"] = calculo[10]
            union.append('ch11')
        if wisel.item(11).isSelected():
            ax.hist(calculo[11],
                    bins=225, histtype='step', label='Ch12', color='#6e60a0')
            dfcentrado["ch12"] = calculo[11]
            union.append('ch12')
        if wisel.item(12).isSelected():
            ax.hist(calculo[12],
                    bins=225, histtype='step', label='Ch13', color='#77eb44')
            dfcentrado["ch13"] = calculo[12]
            union.append('ch13')
        if wisel.item(13).isSelected():
            ax.hist(calculo[13],
                    bins=225, histtype='step', label='Ch14', color='#a53030')
            dfcentrado["ch14"] = calculo[13]
            union.append('ch14')
        if wisel.item(14).isSelected():
            ax.hist(calculo[14],
                    bins=225, histtype='step', label='Ch15', color='#734061')
            dfcentrado["ch15"] = calculo[14]
            union.append('ch15')
        if wisel.item(15).isSelected():
            ax.hist(calculo[15],
                    bins=225, histtype='step', label='Ch16', color='#345c06')
            dfcentrado["ch16"] = calculo[15]
            union.append('ch16')
        if wisel.item(16).isSelected():
            ax.hist(calculo[16],
                bins=225, histtype='step', label='Ch17', color='#4c4b50')
            dfcentrado["ch17"] = calculo[16]
            union.append('ch17')
        if wisel.item(17).isSelected():
            ax.hist(calculo[17],
                    bins=225, histtype='step', label='Ch18', color='#1bff00')
            dfcentrado["ch18"] = calculo[17]
            union.append('ch18')
        fontP = FontProperties()
        fontP.set_size('small')
        ax.legend(bbox_to_anchor=(1, 1), loc=2, borderaxespad=0.,
                  prop=fontP, frameon=False)
        ax.set_ylabel('number of events')
        ax.set_xlabel('counts/min-mean(counts/mean)')
        # Returns:
        # hist : array The values of the histogram.
        # bin_edges : array of dtype float Return
        # the bin edges (length(hist)+1)
        return(dfcentrado[union].dropna().apply(np.histogram, bins=225))
```

Ajuste de la presión:

```
from matplotlib.font_manager import FontProperties
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression


# Funcion diseñada para el pressure correction fit
# que primero calcula la mediana de los datos
def pf(wisel, ax, df, DIni,
        DFin, meditubo, rel, promedio):

    referencias = []
    pref = 93500
    i = 0
    dfm = pd.DataFrame(index=df.index)
    if wisel.item(0).isSelected():
        dfm['ch01'] = df['ch01']
        i += 1
        referencias.append(304.6)
    if wisel.item(1).isSelected():
```

```
        dfm['ch02'] = df['ch02']
        i += 1
        referencias.append(317.)
    if wisel.item(2).isSelected():
        dfm['ch03'] = df['ch03']
        i += 1
        referencias.append(289.7)
    if wisel.item(3).isSelected():
        dfm['ch04'] = df['ch04']
        i += 1
        referencias.append(300.1)
    if wisel.item(4).isSelected():
        dfm['ch05'] = df['ch05']
        i += 1
        referencias.append(300.)
    if wisel.item(5).isSelected():
        dfm['ch06'] = df['ch06']
        i += 1
        referencias.append(273.0)
    if wisel.item(6).isSelected():
        dfm['ch07'] = df['ch07']
        i += 1
        referencias.append(298.4)
    if wisel.item(7).isSelected():
        dfm['ch08'] = df['ch08']
        i += 1
        referencias.append(251.9)
    if wisel.item(8).isSelected():
        dfm['ch09'] = df['ch09']
        i += 1
        referencias.append(300.8)
    if wisel.item(9).isSelected():
        dfm['ch10'] = df['ch10']
        i += 1
        referencias.append(300.3)
    if wisel.item(10).isSelected():
        dfm['ch11'] = df['ch11']
        i += 1
        referencias.append(270.3)
    if wisel.item(11).isSelected():
        dfm['ch12'] = df['ch12']
        i += 1
        referencias.append(299.78)
    if wisel.item(12).isSelected():
        dfm['ch13'] = df['ch13']
        i += 1
        referencias.append(296.6)
    if wisel.item(13).isSelected():
        dfm['ch14'] = df['ch14']
        i += 1
        referencias.append(325.96)
    if wisel.item(14).isSelected():
        dfm['ch15'] = df['ch15']
        i += 1
        referencias.append(296.8)
    if wisel.item(15).isSelected():
        dfm['ch16'] = df['ch16']
        i += 1
        referencias.append(252.5)
    if wisel.item(16).isSelected():
        dfm['ch17'] = df['ch17']
        i += 1
        referencias.append(291.1)
    if wisel.item(17).isSelected():
        dfm['ch18'] = df['ch18']
        i += 1
        referencias.append(270.7)
    X = ((df['pressure'].loc[DIni:DFin] - pref) / 100).values.reshape(-1, 1)
    y = np.log(dfm.dropna().T.median().loc[DIni:DFin] /
               np.mean(referencias)).replace([np.inf, -np.inf], 0)
    regre = LinearRegression()
    regre.fit(X, y)
    ax.scatter(X, y, color='blue')
    ax.plot(X, regre.predict(X), color='red')
    m = regre.coef_[0]
    b = regre.intercept_
    fontP = FontProperties()
```

```
    fontP.set_size('small')
    ax.legend(bbox_to_anchor=(1, 1), loc=2, borderaxespad=0.,
              prop=fontP, frameon=False)
    ax.set_ylabel('ln(N/Nref)')
    ax.set_xlabel('P-PRef')
    ax.title.set_text("Beta = %s " %(m))

    return(dfm)
```

Corrección de la presión:

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression


# Correccion de la presión mediante datos del canal
# Hacemos una regresión lineal para hallar las betas en tiempo real
def corre(df):

    pref = 93500
    referencias = [304.6, 317., 289.7, 300.1,
                   300., 273.0, 298.4, 251.9, 300.8, 300.3,
                   270.3, 299.78, 296.6, 325.96, 296.8, 252.5,
                   291.1, 270.7]
    dfm = pd.DataFrame(index=df.index)
    y = np.log(df['ch01'].dropna() / referencias[0]).replace([np.inf, -np.inf], 0)
    x = (pref - df['pressure']).values.reshape(-1, 1)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta1 = ajust.coef_[0]
    y = np.log(df['ch02'].dropna() / referencias[1]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta2 = ajust.coef_[0]
    y = np.log(df['ch03'].dropna() / referencias[2]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta3 = ajust.coef_[0]
    y = np.log(df['ch04'].dropna() / referencias[3]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta4 = ajust.coef_[0]
    y = np.log(df['ch05'].dropna() / referencias[4]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta5 = ajust.coef_[0]
    y = np.log(df['ch06'].dropna() / referencias[5]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta6 = ajust.coef_[0]
    y = np.log(df['ch07'].dropna() / referencias[6]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta7 = ajust.coef [0]
    y = np.log(df['ch08'].dropna() / referencias[7]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta8 = ajust.coef_[0]
    y = np.log(df['ch09'].dropna() / referencias[8]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta9 = ajust.coef_[0]
    y = np.log(df['ch10'].dropna() / referencias[9]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta10 = ajust.coef_[0]
    y = np.log(df['ch11'].dropna() / referencias[10]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta11 = ajust.coef_[0]
    y = np.log(df['ch12'].dropna() / referencias[11]).replace([np.inf, -np.inf], 0)
```

```
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta12 = ajust.coef_[0]
    y = np.log(df['ch13'].dropna() / referencias[12]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta13 = ajust.coef_[0]
    y = np.log(df['ch14'].dropna() / referencias[13]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta14 = ajust.coef_[0]
    y = np.log(df['ch15'].dropna() / referencias[14]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta15 = ajust.coef_[0]
    y = np.log(df['ch16'].dropna() / referencias[15]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta16 = ajust.coef_[0]
    y = np.log(df['ch17'].dropna() / referencias[16]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta17 = ajust.coef_[0]
    y = np.log(df['ch18'].dropna() / referencias[17]).replace([np.inf, -np.inf], 0)
    ajust = LinearRegression()
    ajust.fit(x, y)
    beta18 = ajust.coef_[0]

    dfm['ch01'] = df['ch01'] * np.exp(-beta1 * (pref - df['pressure']))
    dfm['ch02'] = df['ch02'] * np.exp(-beta2 * (pref - df['pressure']))
    dfm['ch03'] = df['ch03'] * np.exp(-beta3 * (pref - df['pressure']))
    dfm['ch04'] = df['ch04'] * np.exp(-beta4 * (pref - df['pressure']))
    dfm['ch05'] = df['ch05'] * np.exp(-beta5 * (pref - df['pressure']))
    dfm['ch06'] = df['ch06'] * np.exp(-beta6 * (pref - df['pressure']))
    dfm['ch07'] = df['ch07'] * np.exp(-beta7 * (pref - df['pressure']))
    dfm['ch08'] = df['ch08'] * np.exp(-beta8 * (pref - df['pressure']))
    dfm['ch09'] = df['ch09'] * np.exp(-beta9 * (pref - df['pressure']))
    dfm['ch10'] = df['ch10'] * np.exp(-beta10 * (pref - df['pressure']))
    dfm['ch11'] = df['ch11'] * np.exp(-beta11 * (pref - df['pressure']))
    dfm['ch12'] = df['ch12'] * np.exp(-beta12 * (pref - df['pressure']))
    dfm['ch13'] = df['ch13'] * np.exp(-beta13 * (pref - df['pressure']))
    dfm['ch14'] = df['ch14'] * np.exp(-beta14 * (pref - df['pressure']))
    dfm['ch15'] = df['ch15'] * np.exp(-beta15 * (pref - df['pressure']))
    dfm['ch16'] = df['ch16'] * np.exp(-beta16 * (pref - df['pressure']))
    dfm['ch17'] = df['ch17'] * np.exp(-beta17 * (pref - df['pressure']))
    dfm['ch18'] = df['ch18'] * np.exp(-beta18 * (pref - df['pressure']))
    dfm['pressure'] = df['pressure']
    return dfm


# Correción de la presión mediante beta fija por canal
def fija(df):
    # probar no dropna
    # Presion de la estacion
    pref = 93500
    dfm = pd.DataFrame(index=df.index)
    beta1 = -0.00651
    beta2 = -0.00653
    beta3 = 0.0
    beta4 = -0.00645
    beta5 = 0.0
    beta6 = 0.0
    beta7 = -0.00659
    beta8 = -0.00635
    beta9 = -0.00642
    beta10 = -0.00676
    beta11 = -0.00679
    beta12 = -0.00641
    beta13 = -0.00655
    beta14 = -0.00661
    beta15 = -0.00660
    beta16 = -0.00655
    beta17 = -0.00659
    beta18 = -0.00649

    dfm['ch01'] = df['ch01'] * np.exp(beta1 * (pref - df['pressure'])/100)
    dfm['ch02'] = df['ch02'] * np.exp(beta2 * (pref - df['pressure'])/100)
```

```
    dfm['ch03'] = df['ch03'] * np.exp(beta3 * (pref - df['pressure'])/100)
    dfm['ch04'] = df['ch04'] * np.exp(beta4 * (pref - df['pressure'])/100)
    dfm['ch05'] = df['ch05'] * np.exp(beta5 * (pref - df['pressure'])/100)
    dfm['ch06'] = df['ch06'] * np.exp(beta6 * (pref - df['pressure'])/100)
    dfm['ch07'] = df['ch07'] * np.exp(beta7 * (pref - df['pressure'])/100)
    dfm['ch08'] = df['ch08'] * np.exp(beta8 * (pref - df['pressure'])/100)
    dfm['ch09'] = df['ch09'] * np.exp(beta9 * (pref - df['pressure'])/100)
    dfm['ch10'] = df['ch10'] * np.exp(beta10 * (pref - df['pressure'])/100)
    dfm['ch11'] = df['ch11'] * np.exp(beta11 * (pref - df['pressure'])/100)
    dfm['ch12'] = df['ch12'] * np.exp(beta12 * (pref - df['pressure'])/100)
    dfm['ch13'] = df['ch13'] * np.exp(beta13 * (pref - df['pressure'])/100)
    dfm['ch14'] = df['ch14'] * np.exp(beta14 * (pref - df['pressure'])/100)
    dfm['ch15'] = df['ch15'] * np.exp(beta15 * (pref - df['pressure'])/100)
    dfm['ch16'] = df['ch16'] * np.exp(beta16 * (pref - df['pressure'])/100)
    dfm['ch17'] = df['ch17'] * np.exp(beta17 * (pref - df['pressure'])/100)
    dfm['ch18'] = df['ch18'] * np.exp(beta18 * (pref - df['pressure'])/100)
    dfm['pressure'] = df['pressure']
    return dfm
```

Eliminar datos obviamente malos:

```
def baddatasn(df):
    dfm = pd.DataFrame(index=df.index)
    dfm['ch01'] = df['ch01']
    dfm['ch02'] = df['ch02']
    dfm['ch03'] = df['ch03']
    dfm['ch04'] = df['ch04']
    dfm['ch05'] = df['ch05']
    dfm['ch06'] = df['ch06']
    dfm['ch07'] = df['ch07']
    dfm['ch08'] = df['ch08']
    dfm['ch09'] = df['ch09']
    dfm['ch10'] = df['ch10']
    dfm['ch11'] = df['ch11']
    dfm['ch12'] = df['ch12']
    dfm['ch13'] = df['ch13']
    dfm['ch14'] = df['ch14']
    dfm['ch15'] = df['ch15']
    dfm['ch16'] = df['ch16']
    dfm['ch17'] = df['ch17']
    dfm['ch18'] = df['ch18']

    # Creamos un dataframe
    # buscamos la media +-desviacion tipica * 3
    x = dfm.mean() + 3 * dfm.std()
    y = dfm.mean() - 3 * dfm.std()

    for j in range(1, 18):
        # los que no cumplan el rango se ponen NaN
        dfm['ch' + str(j).zfill(2)][dfm['ch' + str(j).zfill(2)] < y[j-1]] = np.nan
        dfm['ch' + str(j).zfill(2)][dfm['ch' + str(j).zfill(2)] > x[j-1]] = np.nan
    dfm['pressure'] = df['pressure']
    return dfm
```

Escala relativa:

```
import pandas as pd


# Funciones para representar en escala relativa
# Funcion "relativo" con la media actual y
# Funcion "fijo" con valores fijos de la estacion
def relativo(df):
    dfm = pd.DataFrame(index=df.index)
    dfm['ch01'] = ((df['ch01'] - df['ch01'].mean()) /
                   df['ch01'].mean()) * 100
    dfm['ch02'] = ((df['ch02'] - df['ch02'].mean()) /
                   df['ch02'].mean()) * 100
    dfm['ch03'] = ((df['ch03'] - df['ch03'].mean()) /
                   df['ch03'].mean()) * 100
    dfm['ch04'] = ((df['ch04'] - df['ch04'].mean()) /
```

```
                df['ch04'].mean()) * 100
    dfm['ch05'] = ((df['ch05'] - df['ch05'].mean()) /
                df['ch05'].mean()) * 100
    dfm['ch06'] = ((df['ch06'] - df['ch06'].mean()) /
                df['ch06'].mean()) * 100
    dfm['ch07'] = ((df['ch07'] - df['ch07'].mean()) /
                df['ch07'].mean()) * 100
    dfm['ch08'] = ((df['ch08'] - df['ch08'].mean()) /
                df['ch08'].mean()) * 100
    dfm['ch09'] = ((df['ch09'] - df['ch09'].mean()) /
                df['ch09'].mean()) * 100
    dfm['ch10'] = ((df['ch10'] - df['ch10'].mean()) /
                df['ch10'].mean()) * 100
    dfm['ch11'] = ((df['ch11'] - df['ch11'].mean()) /
                df['ch11'].mean()) * 100
    dfm['ch12'] = ((df['ch12'] - df['ch12'].mean()) /
                df['ch12'].mean()) * 100
    dfm['ch13'] = ((df['ch13'] - df['ch13'].mean()) /
                df['ch13'].mean()) * 100
    dfm['ch14'] = ((df['ch14'] - df['ch14'].mean()) /
                df['ch14'].mean()) * 100
    dfm['ch15'] = ((df['ch15'] - df['ch15'].mean()) /
                df['ch15'].mean()) * 100
    dfm['ch16'] = ((df['ch16'] - df['ch16'].mean()) /
                df['ch16'].mean()) * 100
    dfm['ch17'] = ((df['ch17'] - df['ch17'].mean()) /
                df['ch17'].mean()) * 100
    dfm['ch18'] = ((df['ch18'] - df['ch18'].mean()) /
                df['ch18'].mean()) * 100
    dfm['pressure'] = df['pressure']
    return dfm


def fijo(df):
    referencias = [304.6, 317., 289.7, 300.1,
                300., 273.0, 298.4, 251.9, 300.8, 300.3,
                270.3, 299.78, 296.6, 325.96, 296.8, 252.5,
                291.1, 270.7]
    dfm = pd.DataFrame(index=df.index)
    dfm['ch01'] = ((df['ch01'] - referencias[0]) /
                referencias[0]) * 100
    dfm['ch02'] = ((df['ch02'] - referencias[1]) /
                referencias[1]) * 100
    dfm['ch03'] = ((df['ch03'] - referencias[2]) /
                referencias[2]) * 100
    dfm['ch04'] = ((df['ch04'] - referencias[3]) /
                referencias[3]) * 100
    dfm['ch05'] = ((df['ch05'] - referencias[4]) /
                referencias[4]) * 100
    dfm['ch06'] = ((df['ch06'] - referencias[5]) /
                referencias[5]) * 100
    dfm['ch07'] = ((df['ch07'] - referencias[6]) /
                referencias[6]) * 100
    dfm['ch08'] = ((df['ch08'] - referencias[7]) /
                referencias[7]) * 100
    dfm['ch09'] = ((df['ch09'] - referencias[8]) /
                referencias[8]) * 100
    dfm['ch10'] = ((df['ch10'] - referencias[9]) /
                referencias[9]) * 100
    dfm['ch11'] = ((df['ch11'] - referencias[10]) /
                referencias[10]) * 100
    dfm['ch12'] = ((df['ch12'] - referencias[11]) /
                referencias[11]) * 100
    dfm['ch13'] = ((df['ch13'] - referencias[12]) /
                referencias[12]) * 100
    dfm['ch14'] = ((df['ch14'] - referencias[13]) /
                referencias[13]) * 100
    dfm['ch15'] = ((df['ch15'] - referencias[14]) /
                referencias[14]) * 100
    dfm['ch16'] = ((df['ch16'] - referencias[15]) /
                referencias[15]) * 100
    dfm['ch17'] = ((df['ch17'] - referencias[16]) /
                referencias[16]) * 100
    dfm['ch18'] = ((df['ch18'] - referencias[17]) /
                referencias[17]) * 100
    dfm['pressure'] = df['pressure']
    return dfm
```

Descarga de los ficheros de datos del servidor:

```
import sys
from PyQt5.QtWidgets import QMainWindow, QApplication, QWidget, QPushButton, QAction,
QMenu, QDialog, QLabel, QLineEdit
from PyQt5.QtGui import QIcon, QPixmap
from PyQt5 import QtCore, QtGui, QtWidgets
import os
import urllib.request


# Menu para descargar los datos del CaLMa desde su servidor
class menufecha(QDialog):
    def __init__(self):
        super().__init__()
        self.title = 'Descargar'
        self.left = 10
        self.top = 10
        self.width = 200
        self.height = 200
        self.iniciar()

    def iniciar(self):
        self.setWindowTitle(self.title)
        self.setGeometry(self.left, self.top, self.width, self.height)
        self.setModal(True)

        etinicio = QLabel(self)
        etinicio.setText('Inicio')
        etinicio.setMinimumWidth(100)
        etinicio.move(50, 40)
        etifin = QLabel(self)
        etifin.setText('Fin')
        etifin.setMinimumWidth(100)
        etifin.move(50, 100)
        self.inicio = QtWidgets.QDateEdit(self)
        self.inicio.setGeometry(QtCore.QRect(50, 60, 110, 27))
        self.inicio.setMinimumDate(QtCore.QDate(2011, 10, 1))
        self.inicio.setDate(QtCore.QDate(2011, 10, 26))
        self.inicio.setObjectName("inicio")
        self.fin = QtWidgets.QDateEdit(self)
        self.fin.setGeometry(QtCore.QRect(50, 120, 110, 27))
        self.fin.setMinimumDate(QtCore.QDate(2011, 10, 1))
        self.fin.setDate(QtCore.QDate(2011, 10, 26))
        self.fin.setObjectName("fin")

        boton = QPushButton('Download', self)
        boton.clicked.connect(self.descarga)
        boton.resize(100, 32)
        boton.move(50, 160)
    # Esta función es un bucle anidado para la distinción de fechas
    # y manda una solicitud de descarga al servidor pasandole las cadenas
    # adecuadas segun el rango de fechas seleccionadas
    def descarga(self):
        if not os.path.exists('data'):
            os.makedirs('data')
        if self.inicio.date() <= self.fin.date():
            for i in range(self.inicio.date().year(),
                           self.fin.date().year() + 1):
                if self.inicio.date().year() == self.fin.date().year():
                    for j in range(self.inicio.date().month(),
                                   self.fin.date().month() + 1):


urllib.request.urlretrieve('http://www.servidor.loquesea/asciidata/'
                           + str(i) + str(j).zfill(2) + '_calma.dat', 'data/' + str(i) +
str(j).zfill(2) +
                           '_calma.dat')
                elif self.inicio.date().year() == i:
                    for j in range(self.inicio.date().month(), 12 + 1):
```

```
urllib.request.urlretrieve('http://www.servidor.loquesea/asciidata/'
                        + str(i) + str(j).zfill(2) + '_calma.dat', 'data/' + str(i) +
str(j).zfill(2) +
                        '_calma.dat')
                elif self.inicio.date().year() != i and i != self.fin.date().year():
                    for j in range(1, 12 + 1):


urllib.request.urlretrieve('http://www.servidor.loquesea/asciidata/'
                        + str(i) + str(j).zfill(2) + '_calma.dat', 'data/' + str(i) +
str(j).zfill(2) +
                        '_calma.dat')
                else:
                    for j in range(1, self.fin.date().month() + 1):


urllib.request.urlretrieve('http://www.servidor.loquesea/asciidata/'
                        + str(i) + str(j).zfill(2) + '_calma.dat', 'data/' + str(i) +
str(j).zfill(2) +
                        '_calma.dat')
                print('finalizado')
        else:
            import dialogo
            dialog = dialogo.Error()
            value = dialog.exec_()
            print ("fecha no valida")
            print (self.inicio.date())
            print (self.fin.date())


def llamada(self):
    menfe = menufecha()
    val = menfe.exec_()
```

Menú para solicitar datos al NMBD:

```
import descarganest as dn
import pandas as pd
import matplotlib.pyplot as plt
from PyQt5 import QtCore, QtGui, QtWidgets


# Menu para seleccionar estaciones del nmdb y representarlas
class Ui_Dialog(object):
    def setupUi(self, Dialog):
        Dialog.setObjectName("Dialog")
        Dialog.resize(444, 463)
        self.label = QtWidgets.QLabel(Dialog)
        self.label.setGeometry(QtCore.QRect(160, 30, 67, 17))
        self.label.setObjectName("label")
        self.dateTimestart = QtWidgets.QDateTimeEdit(Dialog)
        self.dateTimestart.setGeometry(QtCore.QRect(190, 250, 194, 27))
        self.dateTimestart.setMinimumDate(QtCore.QDate(1951, 1, 1))
        self.dateTimestart.setObjectName("dateTimestart")
        self.dateTimeend = QtWidgets.QDateTimeEdit(Dialog)
        self.dateTimeend.setGeometry(QtCore.QRect(190, 280, 194, 27))
        self.dateTimeend.setMinimumDate(QtCore.QDate(1951, 1, 1))
        self.dateTimeend.setObjectName("dateTimeend")
        self.label_2 = QtWidgets.QLabel(Dialog)
        self.label_2.setGeometry(QtCore.QRect(110, 250, 67, 17))
        self.label_2.setObjectName("label_2")
        self.label_3 = QtWidgets.QLabel(Dialog)
        self.label_3.setGeometry(QtCore.QRect(110, 290, 67, 17))
        self.label_3.setObjectName("label_3")
        self.datavarbox = QtWidgets.QComboBox(Dialog)
        self.datavarbox.setGeometry(QtCore.QRect(20, 370, 191, 27))
        self.datavarbox.setObjectName("datavarbox")
        self.datavarbox.addItem("")
        self.datavarbox.addItem("")
        self.datavarbox.addItem("")
        self.datavarbox.addItem("")
```

```
self.relscabutton = QtWidgets.QRadioButton(Dialog)
self.relscabutton.setGeometry(QtCore.QRect(290, 340, 121, 22))
self.relscabutton.setObjectName("relscabutton")
self.countbutton = QtWidgets.QRadioButton(Dialog)
self.countbutton.setGeometry(QtCore.QRect(290, 380, 117, 22))
self.countbutton.setObjectName("countbutton")
self.countbutton.setChecked(True)
self.checkLog = QtWidgets.QCheckBox(Dialog)
self.checkLog.setGeometry(QtCore.QRect(290, 420, 97, 22))
self.checkLog.setObjectName("checkLog")
self.label_4 = QtWidgets.QLabel(Dialog)
self.label_4.setGeometry(QtCore.QRect(20, 340, 101, 17))
self.label_4.setObjectName("label_4")
self.submitButton = QtWidgets.QPushButton(Dialog)
self.submitButton.setGeometry(QtCore.QRect(10, 410, 99, 27))
self.submitButton.setObjectName("submitButton")
self.ASCIIpushButton = QtWidgets.QPushButton(Dialog)
self.ASCIIpushButton.setEnabled(False)
self.ASCIIpushButton.setGeometry(QtCore.QRect(120, 410, 99, 27))
self.ASCIIpushButton.setObjectName("ASCIIpushButton")
self.listWidget = QtWidgets.QListWidget(Dialog)
self.listWidget.setGeometry(QtCore.QRect(90, 50, 256, 192))
self.listWidget.setSelectionMode(QtWidgets.QAbstractItemView.MultiSelection)
self.listWidget.setObjectName("listWidget")
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
self.listWidget.addItem(item)
item = QtWidgets.QListWidgetItem()
```

```
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)
        item = QtWidgets.QListWidgetItem()
        self.listWidget.addItem(item)

        self.retranslateUi(Dialog)
        self.submitButton.clicked.connect(self.submit)
        self.ASCIIpushButton.clicked.connect(self.asciif)
        QtCore.QMetaObject.connectSlotsByName(Dialog)
        Dialog.finished['int'].connect(self.cerrargraf)

    # Funcion para seleccionar los parametros que queremos de las estaciones
    def submit(self):
        if not self.listWidget.selectedItems():
            print("No hay estación seleccionada")
        else:
            items = self.listWidget.selectedItems()
            station = []
            for i in range(len(items)):
                station.append(str(self.listWidget.selectedItems()[i].text()))
            self.i = len(items)
            if self.datavarbox.currentIndex() == 0:
                tipo = "corr_for_efficiency"
            elif self.datavarbox.currentIndex() == 1:
                tipo = "corr_for_pressure"
            elif self.datavarbox.currentIndex() == 2:
                tipo = "uncorrected"
            elif self.datavarbox.currentIndex() == 3:
                tipo = "pressure_mbar"
            if self.countbutton.isChecked():
                unidad = str(0)
            elif self.relscabutton.isChecked():
```

```
                unidad = str(1)
            daystart = str(self.dateTimestart.date().day()).zfill(2)
            monthstart = str(self.dateTimestart.date().month()).zfill(2)
            yearstart = str(self.dateTimestart.date().year())
            hourstart = str(self.dateTimestart.time().hour()).zfill(2)
            minstart = str(self.dateTimestart.time().minute()).zfill(2)
            dayend = str(self.dateTimeend.date().day()).zfill(2)
            monthend = str(self.dateTimeend.date().month()).zfill(2)
            yearend = str(self.dateTimeend.date().year())
            hourend = str(self.dateTimeend.time().hour()).zfill(2)
            minend = str(self.dateTimeend.time().minute()).zfill(2)
            self.dfonline = dn.desc(station, tipo, unidad,
                                    daystart, monthstart, yearstart,
                                    hourstart, minstart, dayend, monthend, yearend,
                                    hourend, minend, self.i)
            self.ASCIIpushButton.setEnabled(True)

    # Fucion que pasa a ASCII los datos
    def asciif(self):
        fileName, _ = QtWidgets.QFileDialog.getSaveFileName()
        if fileName:
            j = 0
            for j in range(self.i-1):
                self.dfonline[j].to_csv(fileName + str(j) + ".dat", sep=";",
encoding='ascii')

    def cerrargraf(self):
        # Funcion para cerrar las graficas al cerrar el dialogo
        plt.close()

    def retranslateUi(self, Dialog):
        _translate = QtCore.QCoreApplication.translate
        Dialog.setWindowTitle(_translate("Dialog", "conexión estaciones"))
        self.label.setText(_translate("Dialog", "Stations"))
        self.label_2.setText(_translate("Dialog", "Start"))
        self.label_3.setText(_translate("Dialog", "End"))
        self.datavarbox.setItemText(0, _translate("Dialog", "Pressure & efficiency
corr."))
        self.datavarbox.setItemText(1, _translate("Dialog", "Pressure corrected"))
        self.datavarbox.setItemText(2, _translate("Dialog", "Uncorrected"))
        self.datavarbox.setItemText(3, _translate("Dialog", "Pressure"))
        self.relscabutton.setText(_translate("Dialog", "Relative scale"))
        self.countbutton.setText(_translate("Dialog", "Counts/s"))
        self.checkLog.setText(_translate("Dialog", "Log scale"))
        self.label_4.setText(_translate("Dialog", "Data variables"))
        self.submitButton.setText(_translate("Dialog", "Submit"))
        self.ASCIIpushButton.setText(_translate("Dialog", "Ascii"))
        __sortingEnabled = self.listWidget.isSortingEnabled()
        self.listWidget.setSortingEnabled(False)
        item = self.listWidget.item(0)
        item.setText(_translate("Dialog", "AATB"))
        item = self.listWidget.item(1)
        item.setText(_translate("Dialog", "APTY"))
        item = self.listWidget.item(2)
        item.setText(_translate("Dialog", "ARNM"))
        item = self.listWidget.item(3)
        item.setText(_translate("Dialog", "ATHN"))
        item = self.listWidget.item(4)
        item.setText(_translate("Dialog", "BKSN"))
        item = self.listWidget.item(5)
        item.setText(_translate("Dialog", "CALG"))
        item = self.listWidget.item(6)
        item.setText(_translate("Dialog", "CALM"))
        item = self.listWidget.item(7)
        item.setText(_translate("Dialog", "DOMB"))
        item = self.listWidget.item(8)
        item.setText(_translate("Dialog", "DOMC"))
        item = self.listWidget.item(9)
        item.setText(_translate("Dialog", "DRBS"))
        item = self.listWidget.item(10)
        item.setText(_translate("Dialog", "ESOI"))
        item = self.listWidget.item(11)
        item.setText(_translate("Dialog", "FSMT"))
        item = self.listWidget.item(12)
        item.setText(_translate("Dialog", "HRMS"))
        item = self.listWidget.item(13)
        item.setText(_translate("Dialog", "INVK"))
```

```
        item = self.listWidget.item(14)
        item.setText(_translate("Dialog", "IRK2"))
        item = self.listWidget.item(15)
        item.setText( translate("Dialog", "IRK3"))
        item = self.listWidget.item(16)
        item.setText(_translate("Dialog", "IRKT"))
        item = self.listWidget.item(17)
        item.setText(_translate("Dialog", "JBGO"))
        item = self.listWidget.item(18)
        item.setText( translate("Dialog", "JUNG"))
        item = self.listWidget.item(19)
        item.setText(_translate("Dialog", "JUNG1"))
        item = self.listWidget.item(20)
        item.setText(_translate("Dialog", "KERG"))
        item = self.listWidget.item(21)
        item.setText( translate("Dialog", "KIEL"))
        item = self.listWidget.item(22)
        item.setText(_translate("Dialog", "KIEL2"))
        item = self.listWidget.item(23)
        item.setText(_translate("Dialog", "LMKS"))
        item = self.listWidget.item(24)
        item.setText(_translate("Dialog", "MCRL"))
        item = self.listWidget.item(25)
        item.setText(_translate("Dialog", "MGDN"))
        item = self.listWidget.item(26)
        item.setText(_translate("Dialog", "MOSC"))
        item = self.listWidget.item(27)
        item.setText(_translate("Dialog", "MRNY"))
        item = self.listWidget.item(28)
        item.setText(_translate("Dialog", "MWSN"))
        item = self.listWidget.item(29)
        item.setText(_translate("Dialog", "MXCO"))
        item = self.listWidget.item(30)
        item.setText(_translate("Dialog", "NAIN"))
        item = self.listWidget.item(31)
        item.setText(_translate("Dialog", "NANM"))
        item = self.listWidget.item(32)
        item.setText(_translate("Dialog", "NEU3"))
        item = self.listWidget.item(33)
        item.setText(_translate("Dialog", "NEWK"))
        item = self.listWidget.item(34)
        item.setText(_translate("Dialog", "NRLK"))
        item = self.listWidget.item(35)
        item.setText(_translate("Dialog", "NVBK"))
        item = self.listWidget.item(36)
        item.setText(_translate("Dialog", "OULU"))
        item = self.listWidget.item(37)
        item.setText(_translate("Dialog", "PSNM"))
        item = self.listWidget.item(38)
        item.setText(_translate("Dialog", "PTFM"))
        item = self.listWidget.item(39)
        item.setText(_translate("Dialog", "PWNK"))
        item = self.listWidget.item(40)
        item.setText(_translate("Dialog", "ROME"))
        item = self.listWidget.item(41)
        item.setText(_translate("Dialog", "SANB"))
        item = self.listWidget.item(42)
        item.setText(_translate("Dialog", "SNAE"))
        item = self.listWidget.item(43)
        item.setText(_translate("Dialog", "SOPB"))
        item = self.listWidget.item(44)
        item.setText(_translate("Dialog", "SOPO"))
        item = self.listWidget.item(45)
        item.setText(_translate("Dialog", "TERA"))
        item = self.listWidget.item(46)
        item.setText( translate("Dialog", "THUL"))
        item = self.listWidget.item(47)
        item.setText(_translate("Dialog", "TIBT"))
        item = self.listWidget.item(48)
        item.setText(_translate("Dialog", "TXBY"))
        item = self.listWidget.item(49)
        item.setText( translate("Dialog", "YKTK"))
        self.listWidget.setSortingEnabled(__sortingEnabled)


if __name__ == "__main__":
    import sys
```

```
    app = QtWidgets.QApplication(sys.argv)
    Dialog = QtWidgets.QDialog()
    ui = Ui_Dialog()
    ui.setupUi(Dialog)
    Dialog.show()
    sys.exit(app.exec_())
```

Módulo de descarga y representación de datos del NMDB:

```
from bs4 import BeautifulSoup
import urllib.request
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from matplotlib.font_manager import FontProperties


# Funcion para descargar datos del nmdb
def desc(station, tipo, unidad, daystart, monthstart,
         yearstart, hourstart, minstart, dayend, monthend,
         yearend, hourend, minend, i):
    plt.close()
    dataframes = []
    j = 0
    fig, ax = plt.subplots(i)
    while j < i:
    # Hacemos la peticion a la url de nest con los datos que necesitamos
        response = urllib.request.urlopen('http://www.nmdb.eu/nest/' +
                                          'draw_graph.php?' +
                                          'formchk=1&stations[]=' +
                                           station[j] +
                                          '&tabchoice=revori&dtype=' +
                                           tipo +
                                           '&tresolution=60&yunits=' +
                                           unidad +
                                           '&date_choice=bydate&start_day=' +
                                           daystart +
                                           '&start_month=' +
                                           monthstart +
                                           '&start_year=' +
                                           yearstart +
                                          '&start_hour=' +
                                           hourstart +
                                          '&start_min=' +
                                          minstart +
                                          '&end_day=' +
                                          dayend +
                                          '&end_month=' +
                                          monthend +
                                          '&end_year=' +
                                          yearend +
                                          '&end_hour=' +
                                          hourend +
                                          '&end_min=' +
                                          minend +
                                          '&output=ascii')
        if tipo == "corr_for_efficiency":
            cadena = "  start_date_time   RCORR_E\n"
            name = "RCORR_E"
        elif tipo == "corr_for_pressure":
            cadena = "  start_date_time   RCORR_P\n"
            name = "RCORR_P"
        elif tipo == "uncorrected":
            cadena = "  start_date_time   RUNCORR\n"
            name = "RUNCORR"
        elif tipo == "pressure_mbar":
            cadena = "  start_date_time    RPRESS\n"
            name = "RPRESS"
        html = response.read()
        parsed_html = BeautifulSoup(html, 'html.parser')
        # Buscamos la etiqueta <code> donde se encuentran los datos
        texto = parsed_html.body.find('code')
```

```
        cad = ''.join(map(str, texto.contents))
        # Separamos en dos cadenas, los datos de estacion se ecuentran en la b
        a, b = str(cad).split(cadena)
        datos = pd.read_csv(pd.compat.StringIO(b), index_col=0, sep=';',
                            date_parser=lambda x: pd.
                            datetime.strptime(x, '%Y-%m-%d %H:%M:%S'),
                            header=None, names=["start_date_time", name])
        dataframes.append(datos)
        print(a)
        # Creamos el datagrama y lo dibujamos, retornamos los datos
        # Para su tratado posterior o importacion en ascii
        #fig, ax = plt.subplots(figsize=(10, 6))
        ax = plt.subplot(i,1,j+1)
        ax.xaxis.set_major_formatter(mdates.DateFormatter('%d-%m-%Y %-j'))
        ax.plot(datos.index, datos[name])
        fontP = FontProperties()
        fontP.set_size('small')
        if unidad == "0":
            ax.set_ylabel('count/s')
        elif unidad == "1":
            ax.set_ylabel('%')
        ax.set_xlabel('date-doy')
        fig.autofmt_xdate()
        plt.title(station[j])
        j += 1
    # ñapa de momento
    plt.subplots_adjust(hspace=0.6)
    plt.show(block=False)
    return dataframes
```

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR

Universidad
de Alcalá