Herramientas Computacionales para Científicos
29/03/2018
Ing. Nicolás Hoffmann

Introducción:

Los sistemas electroquímicos que serán usados como baterías son regularmente evaluados en tres parámetros principales. La primera es la carga que serán capaces de entregar en un circuito, la segunda es la eficiencia coulombiana, la porción de la energía que se le debe entregar para que entregue su carga máxima, y la tercera es la retención, que corresponde a la caída en la capacidad de descarga luego de varios ciclos de carga y descarga.

Para conocer los parámetros en cuestión, se deben ciclar los materiales, lo que corresponde a evaluar el voltaje a través del sistema a medida que se obliga a una corriente a pasar a través del mismo. Cuando la corriente eleva la diferencia de potencial que se mide, se trata de una carga; cuando la diferencia de potencial disminuye con la corriente, se trata de una descarga. Los rangos de diferencia de potencial permisibles son conocidos para la mayoría de los sistemas, y corresponden a una función de la composición química del sistema.

Objetivos:

1- Explicar el problema que se busca afrontar mediante el trabajo en cuestión.
2- Explicar el diseño del programa, incluyendo la selección de las librerías externas.
3- Explicar la implementación de las herramientas dentro del código del mismo.
4- Explicar el uso del programa diseñado.
5- Sugerir cambios posibles a futuro.

Planteo del problema:

Se busca automatizar la visualización de los cambios de potencial durante la carga y la descarga del sistema electroquímico bajo estudio, para poder facilitar la evaluación de los cambios de potencial entre dos o más sistemas.

Los archivos de base serán las salidas de un equipo Arbin, convertidas previamente a formato .xls, compatible con Microsoft Excel 2003.

Diseño del programa:

Para solucionar dicho problema, se elige utilizar Python, que es más sencillo de utilizar; Dado que se busca simplemente visualizar los datos de una tabla, no se espera que la carga de trabajo para el programa sea la suficiente como para justificar el uso de los lenguajes c++ o Fortran, que tienen un mejor rendimiento.

Para acceder a los archivos, se elige utilizar las librerías Pandas 0.22 y xlrd. Para permitir el uso de arrays, conjuntos multidimensionales de números, se utiliza la librería numpy 2.1. Para realizar las visualizaciones, se selecciona la librería matplotlib.pyplot. La interfase gráfica se controla con las herramientas de la librería PyQt(5.10).

El programa funcionará, primero, pidiendo al usuario que indique un archivo. El programa verificará que se trata de un archivo con las partes buscadas, y luego leerá sus contenidos para generar los datos necesarios para la visualización. Ese proceso consistirá en tres partes:

-Leer la masa de material activo del sistema, si está disponible.

-Leer la corriente medida por el equipo, y determinar a partir de ella si se trata de un sistema

de carga o descarga.

-Agrupar las lecturas de carga y descarga según el ciclo al que corresponden.

-Graficar cada ciclo separadamente.

Implementación del código:

El código utiliza las siguientes importaciones.

```
from PyQt5.QWidgets import (QMainWindow, QTextEdit, QAction, QFileDialog, QApplication)
```

Esto importa características básicas de PyQt para poder utilizar una interfase gráfica, incluyendo la ventana de trabajo (QMainWindow) y la ventana de selección de archivos (QFileDialog).

```
from PyQt5.QtGui import QIcon
import sys
```

Estas herramientas sirven para darle un ícono personalizado al sistema, y para permitir que el programa sea capaz de cerrarse a sí mismo.

```
import pandas as pd
import numpy as np
import xlrd
import matplotlib.pyplot as pl
from matplotlib.collections import LineCollection
```

La librería pandas tiene las herramientas necesarias para trabajar con los archivos de excel, y tiene como dependencia, para poder abrirlos, la librería xlrd. Numpy se importa para poder trabajar con arrays, pyplot se importa para poder visualizar la información, y LineCollection permite generar un número de gráficas al mismo tiempo sin conocer de antemano el número de líneas.

Se empezó el trabajo usando un script, sin envolver en una interfase gráfica. Luego, se dividió en varias tareas para poder usarlas independientemente. Dichas tareas están en formas de funciones definidas dentro de definiciones.py.

La primera tarea que se encuentra en Definiciones.py es `abrir_archivo()`, que devuelve `pd.ExcelFile()`. Dicha tarea no es demasiado compleja, y se considera que no requiere más explicaciones. Básicamente, la función es un alias para `pd.excelfile()`.

La siguiente función es `buscar_masa()`. La misma busca el archivo antes abierto (fle_open), que se le entrega como parámetro. Dentro de ese archivo, busca si hay una hoja llamada "Info". De haberla, busca en los comentarios del autor si hay algún patrón que corresponde al peso de material activo del sistema. Realiza esto mediante la función `str1.find(str2)`, donde se busca `str2` en `str1`. De ser encontrada una string que corresponde a la masa del material activo, se asume que el usuario ha registrado en el archivo la masa de material activo del sistema.

Hay dos clases de strings reconocidas por buscar_masa. La primera son los prefijos, como `"la masa de material activo es de "`, `"Material activo: "` o incluso: `"mg de MA: "` que indican que la masa se registra a continuación. Los sufijos indican que la masa de material activo se registra antes de la string que se busca igualar.

Se hace la suposición de que la masa de material activo no puede superar 4 caracteres, lo que corresponde a una masa de entre 0.1 y 9.9 mg de material activo. La mayoría de las masas no superan los 3 mg, y las precisiones de medición son invariablemente de 0.1 mg: se considera que esto es un límite razonable.

La masa se lee utilizando la función `float(string)`, leyendo primero 1 caracter, luego 2, luego 3, hasta llegar a 4. Si se lee un número de caracteres. Como la función devuelve un error de no poder convertir, debe estar precedida por la palabra clave `try`.

Cuando se usa `try`, Python evalúa si se generan errores durante la evaluación. En caso de que se encuentre un error, se cumplen las condiciones evaluadas en `except`. Si alguno de los try devuelve un valor float, sin efectuar except, se considera que existe un valor float para leer. Ni bien el programa incluya suficientes caracteres como para generar un except, se considera que el último valor devuelto por el loop es el valor de material activo.

Si tras todas sus evaluaciones, Python no puede encontrar un patrón de sufijo ni de prefijo en la casilla correspondiente, o si no logra evaluar la masa de material activo, entonces se pide al usuario ingresar manualmente la cantidad. La función devuelve finalmente esa cantidad.

La siguiente función es `buscar_hoja_ciclados()`, que busca en un archivo .xls abierto la hoja correspondiente al ciclado. Devuelve un único número entero, correspondiente a la hoja que se busca leer.

La siguiente función es `generar_hoja()`, que, leyendo un archivo de excel abierto, devuelve una Pandas Dataframe (esencialmente, una planilla de Microsoft Excel) que tiene únicamente las columnas buscadas. Esto permite un trabajo más agil.

Las salidas de buscar_masa y de generar_hoja se utilizan como entradas en la función `separar_ciclos()`. Dicha función es la más compleja del programa, y la más importante.

La función `separar_ciclos()` empieza leyendo la dataframe devuelta por `generar_hoja()`, y generando una lista con los nombres de las columnas. Luego, lee en una variable la posición de la columna de cada nombre buscado para separar los ciclos.

Para generar el arreglo buscado, se empieza generando un array, `arr_append`, de 2x1 casillas. Se utilizan como valores de control dentro de los loops los siguientes:

`arr_existe`: Este booleano registra si el ciclo actual ya ha empezado a ser registrado. Su valor original es False.
`arr_archivo`: Este booleano registra si existe un archivo de los ciclos anteriores. Su valor original es False.
`filas_interes`: El número de filas en la pandas dataframe que debe ser evaluada.
`int_ultimo_paso`: Este entero registra el último paso que hemos procesado. Su valor original es -1.
`int_primer_paso`: Este booleano registra si este es nuestro primer ciclo. Su valor original es True.
`arr_ciclos_tiempo`: Un array que registra el tiempo que pasa en cada ciclo.
`arr_ciclos_tipo`: Un array que registra el tiempo que toma cada ciclo.
`arr_ciclos_limite`: Un array que registra el número de lecturas de cada ciclo.
`jj`: Un contador instantáneo de la última lectura antes de la anterior.
`flt_corriente`: La corriente instantánea medida.
`flt_paso`: El número de paso en el que se está en cada momento.

El loop trabaja iterando un número de veces igual a `filas_interes`. Para cada fila, si la corriente es no nula (es decir, si estamos en un momento donde se está cargando o descargando, no en los descansos entre pasos), registra la diferencia de potencial instantánea y la carga o descarga acumuladas, dividiéndolas por la masa de material activo y multiplicándolas por un factor para determinar el valor en miliamperios hora. Los dos valores (Voltaje, carga) se guardan en un array, `arr_append`.

Luego se establece un loop para separar los ciclos. Lo primero que se hace es comparar el valor del último paso, `int_ultimo_paso`, con el del paso representado en la fila actual.

Cuando estamos dentro del mismo paso, se registra si existe un arreglo registrando el ciclo actual, usando el valor booleano de `arr_existe`. De ser falso, se genera un array, `arr_graficar`, así nombrado por razones históricas, copiando `arr_append`. Luego de copiar el array, se invierte el valor a positivo. Si el valor ya es positivo, `arr_append` se coloca en `arr_graficar`, usando la función `append` de la librería numpy Se debe aclarar `axis=0` para que se incluya verticalmente.

De estar tratando con un paso distinto al último, se actualiza el número de paso, y se indica que deben iniciarse los registros del ciclo actual mediante `arr_existe`. Se cambia el valor del tiempo para el último ciclo, igualándolo al del tiempo registrado en la fila registrada antes de la última: el valor de `jj` se registra por este propósito. Se añaden entradas a `arr_ciclos_tiempo` y `arr_ciclos_limite`, para de esa manera empezar a registrar el nuevo ciclo. Se registra el tipo de ciclo en `arr_ciclos_tipo`, y se añade un espacio vacío al mismo para poder registrar el próximo ciclo. Finalmente, se registra si se han registrado ciclos anteriores, usando el valor de `arr_archivo`. Si el mismo es falso, se lo torna verdadero, y se genera `arr_archivar`, un array, copiando el valor actual de `arr_graficar`.

En todos los casos, luego de evaluar una fila, se debe actualizar el valor de `jj`, la última fila evaluada, para igualarlo al iterador, y se debe avanzar en 1 el valor entero de la última posición de `arr_ciclos_limite`.

Luego de iterar todos los valores, se debe cerrar la función, repitiendo algunas de las funciones internas al loop fuera del mismo. Finalmente, la función debe devolver `arr_archivar` (un arreglo bidimensional con varias columnas de X e Y), `arr_ciclos_tiempo` (un arreglo que registra el tiempo que ha durado cada ciclo), `arr_ciclos_tipo` (un arreglo que registra si los ciclos son de carga o descarga), y `arr_ciclos_limite` (un arreglo que registra cuántas filas corresponden a cada ciclo).

Las salidas de esta función central se usan como entradas para las últimas funciones. La primera de ellas es `hacer_graficas`, que toma todas las salidas de `separar_ciclos`.

La función `hacer_graficas` lee las columnas de `arr_archivar` en sentido inverso (desde el final), tomando pares de las mismas como X e Y. Genera un arreglo de tantas filas como tenga un ciclo, y las llena con esos pares de datos. Cada grilla es luego usada para llenar una lista, `arr_lineas`, que contiene los arrays generados.

Mientras hace esto, genera cuatro valores límites: `x_min`, `x_max`, `y_min`, y_max, a partir de los valores de las columnas que se usarán como x e y, es decir, capacidades (X), y voltajes (Y).

Luego, genera dos listas de datos de ciclo, que incluyen la eficiencia coulombiana (la relación entre una descarga y la carga anterior), y la descarga total.

Devuelve tres listas: `arr_lineas`, `arr_descargas`, `arr_eficiencias`, y 4 valores flotantes: `x_min`, `x_max`, `y_min`, `y_max`.

La función `hacer_graficas_ciclos` es un duplicado de la función anterior, pero saltea la generación de la lista de eficiencias y de descargas, para generar únicamente la lista de `arr_lineas`.

Finalmente, la función `dibujar_capacidades` toma parte de las salidas de hacer_graficas, en particular la lista de arrays `arr_lineas`, y usando `LineCollection`, genera dinámicamente un grupo de líneas a partir de la misma. Tomando los valores `x_min`, `x_max`, `y_min`, e `y_max`, elige los límites del gráfico, de tal manera que no quede espacio muerto ni líneas continuando fuera del mismo.

La función `dibujar_eficiencias` es más sencilla. La misma invierte las listas de eficiencias y de descargas, ya que fueron creadas en sentido inverso, y luego grafica los valores de descarga y de eficiencia. El valor de eficiencia no debe ser graficado en el primer ciclo: como el material activo se sintetiza en el estado cargado, evaluar su eficiencia genera resultados sin ningún sentido.

La interfase gráfica que envuelve todo lo demás funciona con elementos de la librería PyQt5. De la misma se usan las funciones QMainWindow, para generar una ventana, QTextEdit, para generar la ventana de texto del centro, QAction, para separar las acciones de manera conveniente, QFileDialog, para abrir ventanas de selección de archivo, y QApplication, para generar una aplicación capaz de aceptar parámetros.

Se utiliza QIcon de la librería QtGui por razones históricas. Finalmente, se importa sys, para poder utilizar la función de cierre.

Para generar una aplicación, se crea la clase Example(QMainWindow), configurando que al iniciarse contenga una barra de estado y una barra de menú. En la barra de menú, se crea el menú archivo, con las opciones de abrir el archivo, y de generar los gráficos a partir del archivo generado. Se añade también un cierre de programa, para completar el menú.

Para generar los gráficos separadamente de la apertura del archivo, se crea una clase que almacena los valores extraídos de la lectura. En este caso, la clase se conoce como "Preparado". La clase preparado almacena los datos de los ciclos a graficar, y una tabla de valores de voltaje contra capacidad.

Uso del Programa:

Para usar el programa, el procedimiento es sencillo.

1. Correr RUNME.py
2. Abrir un archivo .xls.
3. Elegir una de las dos opciones: mostrar eficiencias, o mostrar ciclos, para generar el gráfico buscado.

Mejoras Posibles a Futuro:

Por el momento, el programa no graba archivos. El programa debería ser capaz de escribir archivos propios, para de esa manera no necesitar abrir un archivo varias veces: el proceso es pesado y trabajoso.

La función `numpy.append` es trabajosa. Tendría sentido leer de antemano el número de entradas que necesitaría un paso, y de esa manera trabajar siempre bajo un mismo array.

Los arrays en este momento almacenan datos heterogéneos, lo que es una mala práctica. Deberían usarse dos arrays, uno para valores de X, y uno para valores de Y. Lo que es más, el array `arr_ciclos_limite` demuestra que se está dejando mucho espacio vacío en la grilla producida por `separar_ciclos`. En lugar de trabajar con un array, tendría sentido trabajar con listas y tuplas anidadas en otras listas y tuplas. Se ha trabajado con arrays por hábito del estudiante, y no eran, en este caso, la mejor herramienta para lo que se buscaba hacer.

Los archivos de salida del equipo, de donde se leen los datos, también contienen datos sobre la eficiencia y la energía de descarga de cada ciclo. Leer individualmente cada momento del ciclado, y generar una dataframe para poder calcularlo, no tiene sentido si se busca sólo la energía de descarga: podría implementarse un mecanismo para leerlos directamente.

A veces, no todos los ciclos de carga y descarga practicados sobre un sistema están en un mismo archivo. Debería ser posible abrir varios archivos a la vez, y devolver una dataframe que los concentre en uno solo, ordenados por fecha, para su separación en ciclos.

Por defecto se muestran todos los ciclos. De a diez no se trata de un problema, pero hay sistemas que se han ciclado miles de veces. Una selección distribuida logarítmicamente (1, 2, 5, 10, 20...) o lineal (1, 1000, 2000...) puede dar una mejor visualización para números de ciclos muy altos. Además, los ciclos deberían representarse con colores distintos, para evitar la confusión.