echo “Sesión 5. ¡¡Bash scripting rules!!” \
>>$HOME/action.finally.now

Elaborado por: Emmanuel Vargas Blanco (uk3xd8@gmail.com)

Martes 3 de Julio, 2012

Como obtener datos de un archivo de MS-Excel (xls) y manipular su contenido.

Ahora que sabemos movernos en el sistema y conocemos algunos comandos básicos (ni tan básicos :) es el momento perfecto para empezar a llevar esas grandiosas ideas a la práctica.

Descripción del problema 1:

Supongamos que tenemos una serie de archivos en formato .xls de los cuales queremos sacar los datos de cada uno y hacer una gráfica con ellos. Los archivos se encuentran en una carpeta llamada problema1.

Cada archivo posee un registro de las mediciones realizadas por 3 sensores. En la primera columna se encuentra la fecha (YYYYmmdd), en la segunda la

date	time	Sensor 1	Sensor 2	Sensor 3
20110205	0004	0.6950955475	1.2042047456	36.4930087142
20110205	0005	0.9245306323	15.4372036341	24.025199879
20110205	0006	0.0421946528	20.8277918631	22.374657914
20110205	0007	0.1629746594	17.8794272011	35.9766245447
20110205	0008	0.9870903958	21.5978813241	37.5145295635
20110205	0009	0.3372332421	24.103271612	16.9998289086
20110205	0010	0.8656292744	4.2304735514	29.3865369074
20110205	0011	0.8352478091	16.1459700437	3.6318259872
20110205	0012	0.1529324106	18.0511950515	29.6567706205
...	...	...	...	...

Cuadro 1: Datos de cada archivo. Problema 1

hora (HHMM) y entre las columnas 3 y 5 se encuentran las mediciones de los tres sensores. El programa debe recorrer cada archivo y convertirlo a un formato csv (coma-separated values) como el que vimos en la sesión 3, con este archivo es que trabajaremos. Luego de tenerlos todos convertidos hay que recorrerlos y graficar la medición de cada sensor en todo el tiempo que posee datos, osea; todo el registro, así obtendremos 5 gráficos, uno de cada sensor con respecto al tiempo.

Solución:

Para convertir un archivo de excel a cvs podemos utilizar el comando xls2csv[1] de V.B.Wagner, para instalarlo sigan los siguientes pasos:

$ sudo rpm -Uhv http://packages.sw.be/rpmforge-release/rpmforge-release-0.5.2-2.el6.rf.x86_64.rpm

$ sudo yum update

$ sudo yum install catdoc

Se estarán preguntando, ¿Por qué estará instalando catdoc si se acaba de pegar una hablada acerca de un tal xls2csv (ver nota al pie # 1), el asunto es que este paquete (catdoc) viene con una serie de programas para modificar archivos de MS-Office.

Ahora, para modificar el archivo utilizo el comando de la siguiente forma:

$ xls2csv archivo1.xls >>archivo1.csv

Para nuestro caso, valla a la carpeta "Problema1" ejecute el comando anterior con alguno de los archivos. Como pueden ver (less archivo.csv) este es de texto plano y posee un formato especial, las columnas están separadas por una coma y los contenidos de cada celda están entre "<celda(fila,columna)>" justo como el archivo de la sesión 3. Ahora hay que hacerlo para todos los archivos[2] con un script (puede realizarse desde la línea de comandos, pero es engorroso y difícil de entender si no se tiene suficiente experiencia). Dentro de la carpeta "problema 1" cree un archivo nuevo con vim llamado problema1.sh y escriba el siguiente código:

```
1 #!/bin/bash
2
3 DATA=/home/centos/Documents/problema1/hojasDatos
4 mkdir $DATA/datos_csv
5
```

[1] Este comando tiene dos variantes, una programada en C por V.B.Wagner, bastante bueno pero tiene problemas con las fechas tipo 03/31/85 o 31-Mar-1985 pues al convertir el archivo las convierte en un número. Hay otra versión escrita por Ken Prows pero solo funciona bajo ciertas condiciones. Ver http://vinayhacks.blogspot.com/2010/04/converting-xls-to-csv-on-linux.html

[2] Por comodidad y tiempo utilizaremos unos pocos archivos, pero este programa funciona para cualquier cantidad.

```
 6 OUT_DATA=$DATA/archivos_csv
 7
 8 m=0
 9
10 for i in `find $DATA -name '*.xls' `
11 do
12         echo "Procesando archivo $i"
13
14         xls2csv $i > $OUT_DATA/data-$m.csv
15         let m=m+1
16 done 2> error1.log
```

Vamos paso a paso, la variable DATA contiene el PATH que apunta a la ubicación de los archivos que vamos a trabajar, luego creamos en esa carpeta otra que se llame datos_csv que es donde se guardarán los archivos convertidos. Dentro del ciclo for viene la primera "galanada" que vamos a hacer. Primero se ejecuta el comando find que actúa sobre la carpeta que contiene los archivos .xls, este buscará cada archivo y pasará el PATH absoluto de cada archivo que encuentre a la variable i, osea; se realiza el ciclo for sobre la lista de direcciones que entregue el comando find. Dentro del bucle se hace un echo para llevar el control de cada movimiento y seguidamente se ejecuta xls2csv para convertir cada archivo que le pasa find a la variable i. Con el comando let incrementamos la variable m que nos sirve de contador para ir nombrando cada archivo que convertimos.

Es importante notar el uso de las variables DATA, OUT_DATA pues nos ayudan a tener un código más sencillo de leer y modificar, así solo modificamos esas variables y no tenemos que ir por todo el programa haciendo cambios.

Finalmente se desvía el STD_ERROR a un archivo que se llame error1.log para poder revisarlo después de que corra el programa, esto nos ayudará incluso a llevar un registro de que sale mal, lo cual se aprecia enormemente cuando trabajamos con cantidades monumentales de datos.

Luego de escribirlo, guarde el archivo y asigne permisos de ejecución de manera que usted lo pueda correr de la forma ./problema1.sh

Revisando el directorio, podemos constatar que el programa se ejecutó de la manera correcta.

```
$ tree
.
├── Datos
├── error1.log
├── hojasDatos
│   ├── archivos_csv
│   │   ├── data-0.csv
│   │   ├── data-1.csv
│   │   ├── data-2.csv
│   │   ├── data-3.csv
│   │   └── data-4.csv
```

```
│   ├── datos1.xls
│   ├── datos2.xls
│   ├── datos3.xls
│   ├── datos4.xls
│   └── datos5.xls
└── problema1.sh
```

Ahora, revise uno de los archivos generados, por ejemplo data-0.csv; copárelo con el archivo Datos. Ejecute el comando file para ver la descripción de ambos. Como puede observar, el archivo Datos es un texto codificado en formato ISO-8859, por esta razón podrán notar que hay ciertos símbolos que no aparecen adecuadamente, el que generamos está en formato ASCII, esto facilita las cosas enormemente ya que permite trabajar todo el texto sin errores con caracteres. Para convertir el archivo Datos podemos utilizar el comando iconv:

```
$iconv -f iso-8859-1 -t utf-8 Datos >> datos.utf-8.out
```

El formato utf-8 es el preferido pues posee más símbolos. Ejecute de nuevo el programa file para determinar el formato del nuevo archivo.

Volviendo a nuestro script, ahora necesitamos acomodar los archivos *.csv para graficar los datos, la idea es que queden separados en columnas. Utilizando el comando awk el trabajo es bastante sencillo, esta herramienta lee del STD_Input cada línea y la separa por campos, como pueden observar en el archivo data-0.csv cada dato está separado por "," así que lo utilizamos como parámetro delimitador:

```
$ cat hojasDatos/archivos_csv/data-0.csv | awk -F "\",\""
'{print $1 " " $2 " " $3 " " $4 " " $5}' | head
```

Primero "tiramos" a la salida estandar el contenido del archivo data-0.csv y se lo "pasamos" al comando awk a través del pipe (|) al comando awk, este utiliza como delimitador (-F) a "," pero tenemos que agregar un \ para indicarle al comando que las comillas serán parte del delimitador, sino el comando las ignorará (código de color rojo), seguidamente le decimos a awk que imprima por cada línea que reciba en la entrada estandar (que viene de la salida de cat) los campos $1, $2, $3, $3 y $5 pero que los separe con un espacio "_" (código color azul), finalmente toda la salida la pasamos a head para que nos imprima solo las primeras 10 líneas.

```
$ cat hojasDatos/archivos_csv/data-0.csv | awk -F "\",\""
'{print $1 " " $2 " " $3 " " $4 " " $5}' | head
"date time Sensor 1 Sensor 2 Sensor 3"
"20110208 0004 0.696742416359484 3.40072046965361 18.1405823864043"
"20110208 0005 0.634325528517365 9.42674912512302 7.33233276754618"
"20110208 0006 0.0267701172269881 17.0649452600628 10.168843306601"
```

```
"20110208 0007 0.242750237695873 1.1048931744881 18.6560449190438"
"20110208 0008 0.263989919796586 12.0491339475848 25.1282352395356"
"20110208 0009 0.632156780920923 17.2953208209947 30.7881533540785"
"20110208 0010 0.653346463572234 19.1443510586396 37.9349328763783"
"20110208 0011 0.655594043433666 6.59481465118006 15.3494665212929"
"20110208 0012 0.667921626009047 19.6568421786651 37.0986516401172"
```

Ahora eliminamos las comillas que están al inicio de la fecha con sed reemplazandolo por nada:

```
$ cat hojasDatos/archivos_csv/data-0.csv | awk -F "\",\""
'{print $1 " " $2 " " $3 " " $4 " " $5}' |sed '1,$ s/"//g'| head
```

Lo que dice ese último pedazo de código es que substituya desde la línea 1 hasta el final del archivo ($) el símbolo “ por nada (//), la g es para que haga los cambios hasta el final de cada línea.

```
$ cat hojasDatos/archivos_csv/data-0.csv | awk -F "\",\""
'{print $1 " " $2 " " $3 " " $4 " " $5}' |sed '1,$ s/"//g'| head
date time Sensor 1 Sensor 2 Sensor 3
20110208 0004 0.696742416359484 3.40072046965361 18.1405823864043
20110208 0005 0.634325528517365 9.42674912512302 7.33233276754618
20110208 0006 0.0267701172269881 17.0649452600628 10.168843306601
20110208 0007 0.242750237695873 1.1048931744881 18.6560449190438
20110208 0008 0.263989919796586 12.0491339475848 25.1282352395356
20110208 0009 0.632156780920923 17.2953208209947 30.7881533540785
20110208 0010 0.653346463572234 19.1443510586396 37.9349328763783
20110208 0011 0.655594043433666 6.59481465118006 15.3494665212929
20110208 0012 0.667921626009047 19.6568421786651 37.0986516401172
```

¡Ya van tomando forma nuestros datos!, ahora hay que eliminar o comentar la primera línea para que a la hora de graficar no interfiera y sea vista como un comentario, para esto podemos utilizar de nuevo sed y sustituir date por #date:

```
$ cat hojasDatos/archivos_csv/data-0.csv | awk -F "\",\""
'{print $1 " " $2 " " $3 " " $4 " " $5}' |sed '1,$ s/"//g'
| sed '1 s/date/#date/g' | head
#date time Sensor 1 Sensor 2 Sensor 3
20110208 0004 0.696742416359484 3.40072046965361 18.1405823864043
20110208 0005 0.634325528517365 9.42674912512302 7.33233276754618
20110208 0006 0.0267701172269881 17.0649452600628 10.168843306601
20110208 0007 0.242750237695873 1.1048931744881 18.6560449190438
20110208 0008 0.263989919796586 12.0491339475848 25.1282352395356
20110208 0009 0.632156780920923 17.2953208209947 30.7881533540785
```

```
20110208 0010 0.653346463572234 19.1443510586396 37.9349328763783
20110208 0011 0.655594043433666 6.59481465118006 15.3494665212929
20110208 0012 0.667921626009047 19.6568421786651 37.0986516401172
```

¡Listo! Ya que logramos definir como queremos que quede cada archivo, lo agregamos a nuestro script para que realice la tarea sobre todos los archivos de datos csv. Para no tener que escribirlo de nuevo lo podemos agregar al archivo ya existente con echo "todo el comando" >>programa.sh y lo editamos para que recorra toda la carpeta.

```
#!/bin/bash

DATA=/media/DATOS/RespDatos/TALLERES/ICE/sesion5/2012/problema1/hojasDatos
OUT_DATA=$DATA/archivos_csv
GRAF_DATA=$DATA/datos_graf

mkdir $DATA/archivos_csv
mkdir $GRAF_DATA

m=0

for i in `find $DATA -name '*.xls' `
do
        echo "Procesando archivo $i"
        xls2csv $i > $OUT_DATA/data-$m.csv
        let m=m+1
done 2> error1.log

m=0

for e in `find $OUT_DATA -name "*.csv"`
do
  echo "Dando formato de datos para graficar el archivo $e"
  cat $e | awk -F "\",\"" '{print $1 " " $2 " " $3 " "
  $4 " " $5}'| sed '1,$ s/"//g' | sed '1 s/date/#date/g'
  > $GRAF_DATA/graf-$m.dat
        let m=m+1
done 2> error2.log


```

Ya tenemos un grupo de archivos decente con el que podemos graficar.

```
.
├── Datos
├── error1.log
├── error2.log
```

```
├── hojasDatos
│   ├── archivos_csv
│   │   ├── data-0.csv
│   │   ├── data-1.csv
│   │   ├── data-2.csv
│   │   ├── data-3.csv
│   │   └── data-4.csv
│   ├── datos1.xls
│   ├── datos2.xls
│   ├── datos3.xls
│   ├── datos4.xls
│   ├── datos5.xls
│   └── datos_graf
│       ├── graf-0.dat
│       ├── graf-1.dat
│       ├── graf-2.dat
│       ├── graf-3.dat
│       └── graf-4.dat
└── problema1.sh
```

gnuplot

Es el momento de graficar, para esto vamos a utilizar el programa gnuplot, primero vamos a crear una carpeta llama plot que contendrá nuestros scripts experimentales. Gnuplot se puede utilizar en modo batch e interactivo, primero trataremos de graficar un archivo de datos en modo interactivo y luego crearemos el script que haga todo.

```
[uk@bakur sesion5]$ gnuplot
G N U P L O T
Version 4.2 patchlevel 6
last modified Sep 2009
System: Linux 2.6.32-220.23.1.el6.x86_64
Copyright (C) 1986 - 1993, 1998, 2004, 2007 - 2009
Thomas Williams, Colin Kelley and many others
Type ‘help‘ to access the on-line reference manual.
The gnuplot FAQ is available from http://www.gnuplot.info/faq/
Send bug reports and suggestions to <http://sourceforge.net/projects/gnuplot>
Terminal type set to 'x11'
gnuplot>
```

Este programa es bastante sencillo de utilizar y una buena forma entrar en contacto es viendo el código:

```
Crear un titulo:              > set title "Titulo"
Etiqueta eje X:               > set xlabel "Tiempo (ms)"
Etiqueta eje Y:               > set ylabel "Corriente (A)"
```

```
Cambia rango eje X:            > set xrange [0.001:0.005]
Cambia rango eje X:            > set yrange [20:500]
Gnuplot determina los rangos:  > set autoscale
Pone etiqueta en gráfica:      > set label "etiqueta 1" at 0.003, 260
Borra todas las etiquetas:     > unset label
Ejes logaritmicos:             > set logscale
Eje Y logaritmico:             > unset logscale; set logscale y
```

Para ver todas las opciones de gnuplot pueden consultar el sitio web www.gnuplot.info

Ahora, copie uno de los archivos generados para graficar (digamos que el archivo graf-0.dat) a la carpeta plot y escriba el comando gnuplot con las siguientes instrucciones:

```
gnuplot> set xdata time
gnuplot> set timefmt "%Y%m%d%H%M"
gnuplot> set xrange  ["20110206 0000" : "20110206 0200"]
gnuplot> set format x "%H:%M"
gnuplot> plot "graf-0.dat" using 1:3 with lines title "sensor1",
"graf-0.dat" using 1:4 with linespoints title "sensor2"
```

Como podemos ver, se genera una imagen que podemos guardar agregando los siguientes comandos:

```
gnuplot> set terminal png
gnuplot> set output 'fig1.png'
```

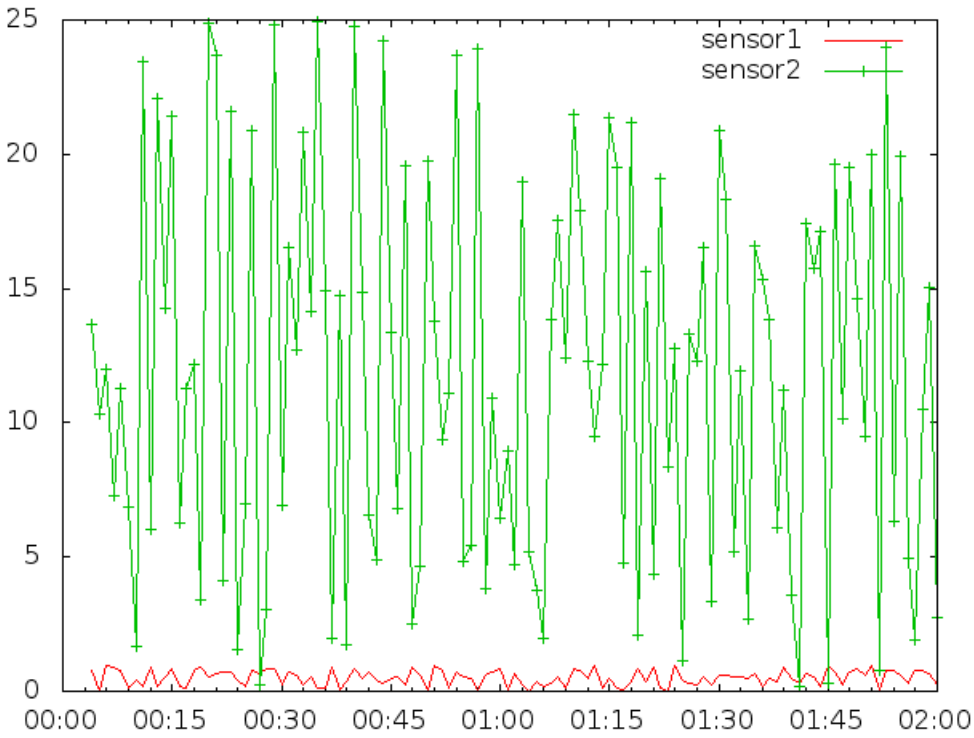

Figura 1: Gráfico 1

Ahora, esta serie de comandos las podemos agregar a un script para mayor control y modificación:

```
1 #!/bin/bash
2
3 FMT_BEGIN=20110206 0000
4 FMT_END=20110206 0200
5 FMT_X_SHOW=%m/%d
6
7
8 graficar()
9 {
10          gnuplot << EOF  2> error.log
11          set xdata time
12          set timefmt "%Y%m%d%H%M"
13          set xrange ["$FMT_BEGIN" : "$FMT_END"]
14          set format x "$FMT_X_SHOW"
```

```
15         set terminal png
16         set output 'fig1.png'
17         plot "graf-0.dat" using 1:3 with lines title "sensor1",
     "graf-0.dat" using 1:4 with linespoints title "sensor2"
18 EOF
19 }
20
21 graficar
```

Como se puede ver, con este script mejoro el manejo de los parámetros a graficar, de forma que puedo especificar hasta el rango en el que se trabajará y todo implementado en una función llamada graficar. Guardamos este programa en un archivo llamado plot1.sh cambiamos permisos y se corre (recordar que el archivo graf-0.dat debe estar en el mismo directorio (plot) que donde se corre el script) podemos observar como aparece la imagen fig1.png.

Ahora, puedo crear otra sección de mi código en el programa problema1.sh para eliminar la primera línea y juntar todos los archivos en uno solo para hacer un solo gráfico:

```
m=0
for k in ‘find $GRAF_DATA -name "*.dat"‘
do
        sed '1d' $k >> $FULL_DATA/full.dat
        let m=m+1
done 2> error3.log
```

Con todas estas herramientas listas puedo hacer mi “Super Script” :

```
 1 #!/bin/bash
 2
 3 DATA=/media/DATOS/RespDatos/TALLERES/ICE/sesion5/2012/problema1/hojasDatos
 4 OUT_DATA=$DATA/archivos_csv
 5 GRAF_DATA=$DATA/datos_graf
 6 FULL_DATA=$DATA/full_datos
 7
 8 mkdir $DATA/archivos_csv
 9 mkdir $GRAF_DATA
10 mkdir $FULL_DATA
11 m=0
12
13 for i in ‘find $DATA -name '*.xls'‘
14 do
15         echo "Procesando archivo $i"
16         xls2csv $i > $OUT_DATA/data-$m.csv
17         let m=m+1
18 done 2> error1.log
19
```

```
20 m=0
21
22 for e in `find $OUT_DATA -name "*.csv"`
23 do
24         echo "Dando formato de datos para graficar al archivo $e"
25         cat $e | awk -F "\",\"" '{print $1 " " $2 " " $3 " " $4 " " $5}'
   | sed '1,$ s/"//g' | sed '1 s/date/#date/g' > $GR    AF_DATA/graf-$m.dat
26         let m=m+1
27 done 2> error2.log
28
29
30 # Este condicional elimina el archivo full.dat ya que si corre varias veces
31 # entonces se agregaran mas datos al archivo en lugar de crearlo con los
32 # datos generados. Osea se agregan por cada corrida un duplicado de los mismos
33 # datos.
34
35 if [ -a $FULL_DATA/full.dat ]
36 then
37         rm $FULL_DATA/full.dat
38         echo "Archivo full.dat borrado"
39 fi 2> errorIf.log
40
41
42 for k in `find $GRAF_DATA -name "*.dat"`
43 do
44         sed '1d' $k >> $FULL_DATA/full.dat
45         echo "Procesando archivo $k"
46 done 2> error3.log
47
48
49 FMT_BEGIN='20110206 0000'
50 FMT_END='20110206 0200'
51 FMT_X_SHOW=%H:%M
52 DATA_DONE=$FULL_DATA/full.dat
53
54 ## La linea set xrange que esta comentada deja en manos de gnuplot el
55 ## la seleccion del mejor rango en el eje x de forma que aparescan todos los
56 ## datos. Si la descomentan entonces pueden manejar el despliegue de estos
57 ## a traves de las variables FMT_BEGIN y FMT_END. En este caso apareceran
58 ## todos los datos. Ver fig1.png donde aparecen todos los datos y en fig.png
59 ## solo aparecen los datos del 6 de febrero como lo establecen las variables.
60
61
62 graficar()
63 {
64        gnuplot << EOF  2> error.log
```

```
65          set xdata time
66          set timefmt "%Y%m%d%H%M"
67 #          set xrange ["$FMT_BEGIN" : "$FMT_END"]
68          set format x "$FMT_X_SHOW"
69          set terminal png
70          set output 'fig1.png'
71          plot "$DATA_DONE" using 1:3 with lines title "sensor1"
            ,"$DATA_DONE" using 1:4 with linespoints title "sensor2"
72 EOF
73
74 }
75
76 graficar
77
```

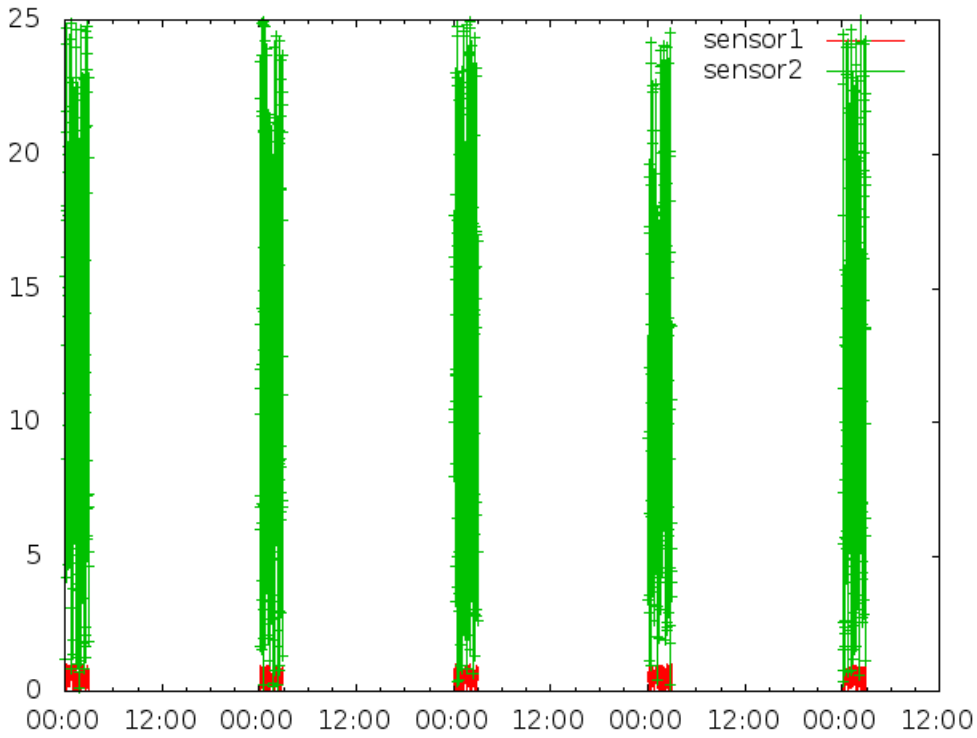

Figura 2: Figura con todos los datos.

En las carpetas problema 2 y ejercicio 3 hay una serie de archivos que terminan en .xls. El ejercicio 2 posee un registro de 6 meses de las cuentas en una

casa de estudiantes, entre las celdas B3 y B10 se encuentran los montos totales de los servicios básicos del "Chante". Obtenga un gráfico donde se muestre la variación del consumo eléctrico en los primeros 3 meses del año 2012, además grafique el consumo de agua desde enero hasta junio.

El ejercicio 3 contiene el famoso archivo Datos de la sesión 3, grafique el índice de radiación solar (KW/m^2) mínimo y máximo del archivo Datos. Cuando tenga armado el "SuperScript" utilice el comando time a la hora de ejecutarlo para determinar cuanto dura ejecutándose la tarea. ¿Cuanto cree que le habría llevado procesar usted solo este archivo? ciertamente, contando el tiempo de programación, habría sido mejor hacerlo en una hoja de calculo... pero, ¿que tal si no es solo un archivo de datos?

Información de Contacto:

- Gustavo Garbanzo Salas: ggarbanzos@gmail.com / 8713-7839
- Emmanuel de Jesús Vargas Blanco: uk3xd8@gmail.com / 8896-9189
- Esteban Pérez Hidalgo : stbn182@gmail.com / 8866-1647