Applying PhysioNet tools to manage neurophysiological signals

Jesus Olivan Palacios

version 0.00, December 2002

Contents

Chapter 1

Introduction: Using PhysioNet tools to read and analyze neurophysiological files

Nowadays, neurophysiological equipment is, in most cases, a set of closed systems designed to perform specific processing very efficiently. Tasks such as displaying signals, measuring times or amplitudes, quantifying sleep and many others are usually done properly. Why would one be interested in doing similar things with tools that are more difficult to use? I will try to show that they are not exactly *similar* things.

Let's begin with an example: When you take photographs with a simple camera, the result is good in a sunny day, when the object is not moving and if its distance to the camera is appropriate. More complex cameras allow taking photographs in more difficult circumstances at the price of having to make decisions and, consequently, at the price of having to make much more effort to understand the mechanics of the device. In almost any kind of tool there is a balance between versatility and simplicity. What is not always appreciated is that delegating decisions to automatic devices by hiding the details does not always produce the best results.

In the present tutorial we are going to face the problem of the analysis of the heart rate in the context of a neurophysiological recording. Usually, electrocardiographic signals are systematically recorded in sleep recordings and routine EEGs. We will describe the use of the tools contained in PhysioNet[1] in this context.

The analysis of heart rate in a neurophysiological environment is more and more important. Heart rate is a window to the autonomous nervous system, showing changes associated with sleep and wakefulness, providing a useful means for detecting arousals, and showing specific changes in association with apneas or seizures. As a matter of fact, heart rate is another *signal* similar to EEG, EOG or EMG. It is a signal that contains a lot of information to link with the remaining signals of our recordings.

The procedure is very similar to the procedure described in Sciteam[2] (also in our site) but in this case we are not going to center on the use of Scilab[3] but in the use of the command interface present in Windows and Unix systems. Since I think that Linux is the best choice we will center on this operating system, although in most cases the tools have been adapted to Windows and you can apply the same concepts with minor modifications.

PhysioNet[4] contains a lot of biological signals coming from different fields. Signals are stored in WFDB signal file formats and they can be very precisely annotated, analyzed and displayed. To do this, PhysioNet also provides free tools to handle the signals. Taken together, the huge collections of data contained in PhysioNet, the fully specified formats and the tools to handle them form an unbeatable team. The goal of

[1]http://www.physionet.org
[2]http://www.neurotraces.com/scilab
[3]http://www-rocq.inria.fr/scilab
[4]http://www.physionet.org

this tutorial is to present some of these tools to clinical neurophysiologists.

In Clinical Neurophysiology, most equipment does not include a *save as* option in formats directly compatible with PhysioNet tools. ASCII (text) files and EDF format are the most widely available options to store data in non-native format for neurophysiological devices. In this tutorial we will use ASCII files as the input of the procedure.

And let us begin. To follow the tutorial, all the tools that you need can be downloaded from the Internet:

1. The WFDB Software Package. It can be downloaded from the PhysioToolkit[5] section of PhysioNet. I recommend that you download it in RPM format because it is easier to install, although it can be easily compiled (it is an open source package under the GPL license) . It includes an excellent viewer: *wave* .

2. An example of a short segment of polygraphic recording (the file *base*) that can be downloaded from Neurotraces[6](anonymous ftp).

3. Scilab. A very versatile program for processing signals. It can be downloaded from the Scilab site. A tutorial about its use can be found in Neurotraces[7].

[5]http://www.physionet.org/physiotools/wfdb.shtml
[6]ftp://ftp.neurotraces.com/pub/ECG/
[7]http://www.neurotraces.com/scilab

Chapter 2

Reading an ASCII file produced by commercial equipment and marking the QRS complexes

We begin with the file *base* that can be downloaded from Neurotraces[1]. The directory contains the file in compressed form (done with gzip) and uncompressed forms; the uncompressed file is about six times as large as the compressed one. To avoid conflict with other files, we will create a directory called *Code*, where we are going to locate our files. We assume that WFDB has been installed previously.

2.1 The *base* file

The file was created by a Nihon Kohden EEG-1100. Similar files can be created with many types of modern neurophysiological equipment. The file is part of a polysomnographic recording, and one of its signals is an electrocardiographic signal. By viewing the recording we select a part and store the result as an ASCII file.

Now the first step is taking a glance into the content of the file:

```
[j@localhost Code]$ cat base | more
TimePoints=3400 Channels=19 BeginSweep[ms]=0.00 Sampli
ngInterval[ms]=5.000 Bins/uV=1.000
C3-A2 C4-A1 O1-A2 O2-A1 T1-A1 T2-A1 PG1-PG2 T5-P3 P4-T
6 X1-X2 X3-X4 X5-X6 X7-X8 E-X9 E-X10 E-X11 DC01 DC02 D
C03
    9.56    21.32    -1.47    11.76    -8.82    -8.09
  -26.47     1.47    13.97   -66.18    -6.62   -12.50
  -44.12    33.09   -17.65    44.85 -923529.41 -2205.8
8 -376470.59
   -7.35   -35.29    -7.35     1.47    -7.35    37.50
  -22.06     1.47    19.12     0.00    -3.68   -10.29
  -22.06    31.62   -18.38    43.38 -924264.71 -2941.1
8 -375735.29
   -4.41    -7.35   -12.50     4.41    -2.94    18.38
   -2.94     1.47   -42.65     0.00    -1.47    -2.94
```

[1] ftp://ftp.neurotraces.com/pub/ECG/

```
  -55.15    31.62   -19.12    43.38 -924264.71 -2941.1
8 -375000.00
    2.94    -0.74    -2.21     8.09     -4.41     4.41
   32.35     1.47   -63.97   -22.06      0.00     3.68
 -121.32    31.62   -16.91    44.12 -924264.71 -3676.4
7 -376470.59
   -1.47    -8.09    -4.41     8.82      1.47     9.56
--More--
```

By inspecting the content of the file we can see that it includes 3400 samples with 19 channels. Since the sampling interval is 5 ms (equivalent to a sampling rate of 200 Hz) we have 17 seconds of recording (3400 samples / 200 samples/second = 17 seconds). We also know that the values are expressed in μV. The first lines describe the recording as well as the signals included in the recording. In our case, the electrocardiographic signal is included in the channel whose label is *X1-X2*. Each line has been folded to adapt its length to the window size. The first line begins with *TimePoints...*, the second one begins with *C3-A2...* the third one begins with *9.56...* and the fourth one begins with *-7.35...*.

2.2 ASCII files as glue between applications

Since it is a key point in exchanging information, I would like to discuss a little bit the use of ASCII files to share neurophysiological recordings:

A recording is represented in ASCII files as a matrix of values. Usually, each column is a different signal, and the file contains as many columns as signals are stored; each row represents the samples of these signals at the same time.

- Unless we introduce a signal as *time* we do not know the sampling rate. This is an inconvenience because time is not stored in a standard location and then we have to introduce it by hand for further processing. Moreover, it is possible that we may not remember the sampling rate when we use the recording.

- The introduction of a header mixes the description of the signals with the samples (the digitized values of the signals) themselves. In our file, the sample values (from the third line to the end of the file) and the signal descriptions (the first two lines) are mixed. Since the format is not standardized, these lines are different in files obtained using other manufacturers' equipment.

- When we store polygraphic recordings in ASCII files, the size of the file is an inconvenience too. Let us consider what would happen if we decided to store the same information in binary format (3400 samples of 19 channels stored as two-byte integers). Each sample uses 38 bytes. A recording of 17 seconds sampled at 200 Hz as *base* uses 3400 samples, i.e. 129,200 bytes. Our file uses 598,203 bytes, five times more !! The size of the files is less and less important nowadays with massive storing disks but even so, some reduction in size would be nice. We can improve the efficiency of ASCII files by compressing them. A compressed form of *base* can be downloaded from Neurotraces[2] being its size about 100,000 bytes, less than the size used when we store the signal in binary format. It has been compressed with *gzip*; if we use *bzip2*, the size is about 60,000 bytes, half of the size of the binary file. Of course, the compression of binary files also increases the efficiency of copying, transmitting, and (sometimes) reading them.

Considering these inconveniences, let us say something about their benefits.

- ASCII files, even being so simple, are not a very bad format when they are compressed (binary files are frequently downloaded in a non-compressed form) or to store short signals.

[2]ftp://ftp.neurotraces.com/pub/ECG/

- Since we are not limited by any format, each data has arbitrary precision.
- The main benefit, however is: *They can be understood by almost any program, from spreadsheets to sophisticated digital signal processing packages*.

In summary, the conversion from and to ASCII files is an important feature of any format.

Of course, representing a recording as a matrix of rows and columns does not readily allow a different sampling rate for each signal (to do this, we might define a code to indicate that a signal was *not sampled* at the time corresponding to a specific row), but even so I can foresee that ASCII files are going to be used for a long time (unless XML[3] is quickly and universally adopted).

2.3 Creating a WFDB file

Our first task is to create a WFDB signal file from an ASCII file. To do this, we have a very easy command: *wrsamp* (something like *write samples*). Most WFDB applications show a short summary of how they are used if we type the name of the program (only) as a command; *wrsamp* shows us this description of itself:

```
[j@localhost Code]$ wrsamp
usage: wrsamp [OPTIONS ...] COLUMN [COLUMN ...]
where COLUMN selects a field to be copied (leftmost field is column 0),
and OPTIONS may include:
 -c          check that each input line contains the same number of fields
 -f N        start copying with line N (default: 0)
 -F FREQ     specify frequency to be written to header file (default: 250)
 -G GAIN     specify gain to be written to header file (default: 200)
 -h          print this usage summary
 -i FILE     read input from FILE (default: standard input)
 -l LEN      read up to LEN characters in each line (default: 1024)
 -o RECORD   save output in RECORD.dat, and generate a header file for
              RECORD (default: write to standard output in format 16, do
              not generate a header file)
 -r RSEP     interpret RSEP as the input line separator (default: \n)
 -s FSEP     interpret FSEP as the input field separator (default: space
              or tab)
 -t N        stop copying at line N (default: end of input file)
 -x SCALE    multiply all inputs by SCALE (default: 1)
```

A lot of interesting options. We have to indicate the sampling frequency (200 Hz); otherwise the program will assume that it is sampled at 250 Hz.

Electroencephalography or Electromyography amplitudes are usually expressed in μV, so one of the options deserves more comment. Here is a more detailed description of this option from the *WFDB Applications Guide*:

```
-G n:

Specify the gain (in A/D units per millivolt) for the output
signals (default: 200). This option is useful only in
conjunction with -o, since it affects the output header
file only. This option has no effect on the output signal
```

[3] http://www.w3.org/XML

```
file. If you wish to rescale samples in the signal file, use -x.
```

Our ASCII file contains sample values in μV, so there are 1000 A/D units per millivolt, and we should therefore specify a *gain* of 1000. If we do not consider this point, we will obtain a signal five times bigger (the default is 200). Another interesting option is -x, which directly modifies the input. It is an important option when our file contains values smaller than 1 (it is not the case in our signal).

We know that the ECG is contained in column 9 of our ASCII file, *base*. (WFDB numbers the columns beginning at 0). But will *wrsamp* be able to detect that the first two lines of the file are not data? Let's see.

```
[j@localhost Code]$ wrsamp -i base -F 200 -G 1000 -o ecg 9
wrsamp: line 0, column 9 missing
wrsamp: line 1, column 9 improperly formatted
```

Wrsamp detected that the first two lines were not properly formatted and emitted a message. We are impatient to see the result

```
[j@localhost Code]$ ls ecg*
ecg.dat  ecg.hea
[j@localhost Code]$ cat ecg.hea
ecg 1 200 3402
ecg.dat 16 1000 12 0 0 -25694 0 base, column 9
```

We created two files: a binary signal file, *ecg.dat*, that stores the digitized samples of the ECG signal, and a short text header file, *ecg.hea*, that contains information that will be needed by any WFDB application that reads the signal file.

A typical WFDB application reads a *record*, which is a collection of files that are all related to the same recording. It is important to understand that the name of the record we have just created is *ecg*, and not the name of either of the files that belong to this record. When we read these files later on, we will refer to them by the record name, *ecg*, and not by the names of the individual files.

We were lucky that *wrsamp* rejected the first two lines of *base*. If we had chosen a different column number, one or both of these lines might have been accepted, and our signal file would have a spurious sample or two at its beginning. Looking back at *wrsamp*'s options, we can see that -f allows us to tell *wrsamp* where to begin; so in the future, if we know that there are two header lines in our input file, we will add “-f 2” to our *wrsamp* command.

2.4 Analyzing the files

At this moment we have a WFDB record containing the ECG of our recording. We are interested in detecting the heart rate of the signal. We are going to use the command *sqrs*

```
[j@localhost Code]$ sqrs -r ecg
[j@localhost Code]$ ls ecg*
ecg.dat  ecg.hea  ecg.qrs
```

A new file (*ecg.qrs*) has been added to the *ecg* record. It is an annotation file that contains the positions of the QRS complexes. We can read the annotations

```
[j@localhost Code]$ rdann -r ecg -a qrs | more
    0:00.110        22      N     0     0     0
    0:00.785       157      N     0     0     0
    0:01.450       290      N     0     0     0
    0:02.115       423      N     0     0     0
    0:02.790       558      N     0     0     0
    0:03.450       690      N     0     0     0
    0:04.110       822      N     0     0     0
    0:04.775       955      N     0     0     0
    0:05.445      1089...
```

Each line is a QRS complex that has been detected.

2.5 In summary

Let us recapitulate what we did in this section:

- We had an ASCII file containing a segment of a polygraphic recording. We created a WFDB file with the content of one of the signals (by using *wrsamp*).
- Then we created an annotation file with the positions at which the QRS complexes are located (by using *sqrs*)

But how can we be confident of the result? WFDB has a very nice tool to view and edit the result: *wave*. In the next section we are going to edit the result by using it.

The WFDB Software Package includes two QRS detectors, named *sqrs* and *wqrs*, and PhysioToolkit offers another one, named *ecgpuwave*. All of them are used in a similar way, and all of them create an annotation file containing the times of the QRS complexes that they detect. Each has advantages for some types of studies; you can read more about them in the *WFDB Applications Guide*[4].

[4]http://www.physionet.org/physiotools/wag/wag.htm

Chapter 3

Editing the result

In the previous chapter we were able to detect the QRS complexes of a recording created by a commercial equipment by using WFDB tools. Now we want to edit the result.

3.1 The whole picture

To do this, we need a viewer that allows deleting or adding QRS detections as well as reposition them if necessary. First of all, we are going to show the result, then we will explain the choices taken, as well as the process of installation. Once installed, the command is direct:

```
[j@localhost Code]$ wave -r ecg -a qrs
```

We indicate the record name (*ecg*) as well the annotation file *qrs* and a nice window appears. The result (sligthly modified) can be seen in the figure 3.1

The result of the analysis seems very good. Let me stress that the signal is not good at all and that it was not even an WFDB signal in origin. Probably, no editing is necessary. In any case, editing the annotations with *wave* is a pleasure. Many subtle features are included (from defining your own set of annotations to measuring signal amplitudes, from acting as an interface to other programs to synchronizing several windows, from directly accessing recordings on the PhysioNet web server to annotating a recording from scratch). It is a wonderful program. You can follow an excellent tutorial at PhysioNet[1] with the details. *Wave* is installed as a part of WFDB Software Package in Linux systems.

3.2 Differences between Gtkwave and Wave

At this point, given the importance of *wave*, I am going to dedicate some time to installation topics. Firstly, I will emphasize the differences between *wave* and *gtkwave*.

To develop a graphic program, you usually use a graphic library. A graphic library is a set of graphic functions that do things such as creating a window, drawing a line etc. Wave uses XView. Since it is not easy to use it in Windows, there is an alternative. A similar program (*Gtkwave*) also developed by the authors of WFDB uses Gtk. Since both programs are more or less similar, I will stress the differences:

- Gtkwave is easier to install than Wave in Linux
- Gtkwave can be used in Windows. Wave can not be used in Windows

If you want to run WFDB in Windows, then you can download the Self-extracting-installer[2]. It installs

[1]http://www.physionet.org/physiotools/wug/wug.htm
[2]http://www.physionet.org/physiotools/beta/gtkwave/#DOWNLOADS

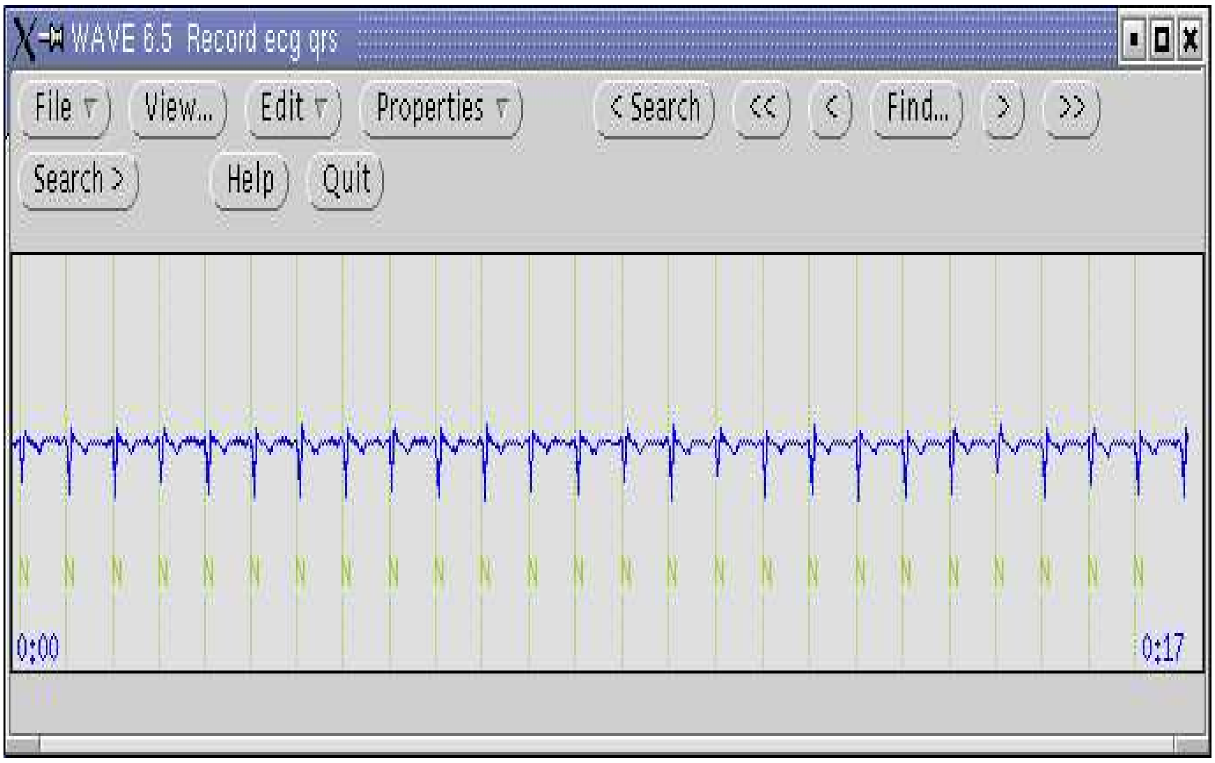

Figure 3.1: *Wave* showing the result of *sqrs* on our file

a set of WFDB tools as well as Gtkwave. The installation is very easy and you can immediately use the program with the recording *100s* that is included in the package.

If you are using Linux, I strongly recommend the use of Wave (instead of Gtkwave). Let me detail some reasons:

- The documentation is more precise
- There are some very nice features such as the “scope window” not implemented in gtkwave
- At least in Windows 98 and Windows NT, to mark the signal you often need to use the key F2. Sometimes this key does not act and the control is lost. This often happens when you use the mouse to navigate through the record. (If you want to do it and find this problem, try to navigate using *Page Up Key* and *Page Down Key*).
- There seems to be a bug when you select viewing only some signals in *gtkwave* (e.g., calling it with option -s and selecting *listed signals only*) drawing always the first signal when you do not select *drawing all signals*.
- In the *annotation template window*, the list of *type* usually exceeds the size of the screen, making difficult the selection of some options
- Wave is in general more stable

I would not like to be misunderstood. Gtkwave is an excellent program. Wave is even better. If you are running Linux, I recommend that you make the aditional effort of installing *wave*.

In any case, if you are running Windows and you feel frustrated by not being able to use *wave*, turn a lemon into lemonade: install Linux.

3.3 Installation in Mandrake 9.0

Since the main inconvenience encountered with *Wave* is the difficulty in the installation, and the details in PhysioNet are directed to Red Hat, I will try to detail the procedure in Mandrake 9.0. Everything you need is included in the distribution except:

- WFDB package. It can be downloaded in RPM form from here[3] in PhysioNet. You can download the *binary WFDB RPM* by clicking in the paragraph entitled *Installing the WFDB Software Package from a binary RPM*.
- XView library. On PhysioNet, there are different libraries for several versions of Red Hat. With Mandrake 9.0 the version of Red Hat 7.0 functions properly. You can download these RPMs here[4].

And now the procedure:

- Installation of the *W3C libwww libraries*.You can install them from Mandrake disks. You need to install the packages *w3c-libwww* and *w3c-libwww-devel*. You can do it by accessing *Mandrake Control Center Software Management* and *Install Software*. In this way every dependency will be solved.
- Installation of *XView libraries*. Install in this order *xview-3.2p1.4-16.i386.rpm*, *xview-devel-3.2p1.4-16.i386.rpm* and *xview-clients-3.2p1.4-16.i386.rpm*. An easy way to do this is opening the file with *kpackage* (by clicking on the icon of the file, once you downloaded it to your disk) and following the instructions to install the package. If the order is not the appropriate, *kpackage* will require you to install the other needed packages

[3]http://www.physionet.org/physiotools/binaries/intel-linux/
[4]http://www.physionet.org/physiotools/xview/i386-7.x/

- Installation the *WFDB package*. You can install *wfdb-10.3.0-1.i386.rpm* by using kpackage in the same way that in the previous point.

XView uses an editor called *textedit* (included in xview-clients). It is installed in */usr/openwin/bin/textedit*. To call it from *Wave*, it is necessary to define it as *EDITOR*

```
[j@localhost Code]$ export EDITOR=/usr/openwin/bin/textedit
[j@localhost Code]$ wave -r ecg -a qrs
```

Some alternatives are to add */usr/openwin/bin* to your PATH, or to copy *textedit* into a directory in your PATH, or to make a link from */usr/openwin/bin/textedit* to a directory in your PATH; if you do any of these it will not be necessary to set EDITOR each time you plan to use 'textedit' within 'wave'.

Once made this arrangements, when you push the button *Analyze* and then *Edit Menu* you are asked to copy *wavemenu* in your own directory. Once you click on *copy* the menu (a very useful resource) appears edited with *textedit*. You can define any other editor (not necessarily textedit) to edit the menu.

Finally, when you press F1 in *wave*, nothing happens. To get help when you press F1, you can use the next command.

```
[j@localhost Code]$ xmodmap  -e "keysym F1 = Help"
[j@localhost Code]$ wave -r ecg -a qrs
```

Now, when you press F1 a help window appears.

At this moment we have installed *Wave*. It is time to enjoy it. If you are connected to the Internet and you type the command

```
[j@localhost Code]$ wave -r slpdb/slp01a -a st
```

then you have access to the record *slp01a* of the database *slpdb* with the annotation *st*, as if it had been downloaded to your computer. It is a sleep recording with a quantification of stages in epochs of 30 seconds. Believe or not, you can navigate through it from your own computer without downloading the recording.

Let's do the same with our recording:

```
[j@localhost Code]$ wave -r ecg -a qrs
```

And we can begin to practice with our own data.

3.4 In summary

It has been a long and challenging session. At this moment we have installed *wave*. We can follow the tutorial and edit our results. We can add or eliminate annotations, we can change the type of annotation or assign them to some specific signal. Once finished we will export the results to continue our analysis.

In the next chapter we are going to use Scilab to analyze the data.

Chapter 4

Interacting with Scilab

In the previous chapter we were able to edit an annotation file, modifying the QRS complexes detected by WFDB software. Now we want to export the results to be used with other software.

As you probably know, Scilab[1] is a Matlab-like scientific software package for numerical computation that can be very useful in managing neurophysiological signals. In this section we are going to try to link signals and annotations between WFDB tools and Scilab.

In the last chapters we were able to convert ASCII files containing matrices of data into WFDB files. Matrices are the core of Scilab. Managing matrices (even very long ones) is extremely easy in Scilab. Since a recording can be seen as a matrix of values (the columns being signals), ASCII files, once more, can be used as the glue to connect Scilab and WFDB.

4.1 Reading the results of processing with Scilab

Our first task is to create a signal containing the heart rate. We would like for this signal to be sampled at the same rate as the ECG signal. WFDB has an application named *tach* that can make a heart rate tachogram from an annotation file. We can use *tach* to make either a WFDB-format tachogram that can be studied with *wave*, or a text-format tachogram that we can study with Scilab. We are going to indicate that we want the heart rate signal sampled at 200 Hz (the sampling rate of the original signal), and we send the result to the file *tac*.

```
[j@localhost Code]$ tach -r ecg -a qrs -F 200 > tac
```

Let us see how easy is to read the result with Scilab. We call Scilab from the same directory. Now we will read the file *base* containing the original signals and the file *tac* that contains the result of our analysis

```
[j@localhost Code]$ scilab
                           ===========
                           S c i l a b
                           ===========


                          scilab-2.6
                   Copyright (C) 1989-2001 INRIA
```

[1]http://www-rocq.inria.fr/scilab

```
Startup execution:
  loading initial environment

-->tac = fscanfMat("tac");

-->size(tac)
 ans  =

!   3241.    1. !

-->base = fscanfMat("base");

-->size(base)
 ans  =

!   3400.    19. !
```

Now we have two variables in Scilab:

- *base*: This is a matrix with 19 signals (columns). In the column 10 (9 for WFDB since WFDB counts from 0) we have the ECG that was analyzed.
- *tac*: This is a column of values containing the tachogram.

We can plot it:

```
-->time1 = (1:3241)/200;

-->signal1 = tac;

-->time2 = (1:3400)/200;

-->signal2 = base(:,10);

-->plot2d(time1,signal1)

-->plot2d(time2,signal2/5)
```

The result can be seen in figure 4.1.

We have corrected the amplitude of the electrocardiographic signal. There is a good coincidence between the ECG and the heart rate. You can check the result by eliminating in *wave* some QRS detections and by observing the changes with Scilab.

WFDB is very well equipped to analyze electrocardiographic signals. We are going to make some analysis not implemented in WFDB. We want to detect the respiratory rate. We have the airway signal in column 14 (13 in WFDB) of our original recording (*base*).

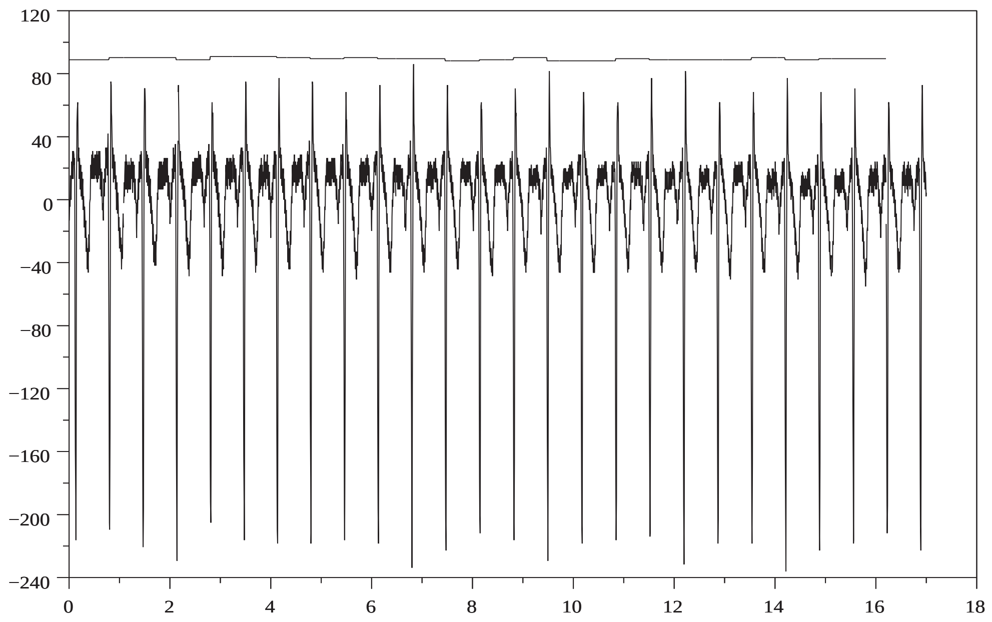

Figure 4.1: The tachogram plotted together with the ECG in Scilab

4.2 Creating some analysis tools with Scilab

To analyze the respiratory rate with Scilab we will use a very simple approach. Do not pay to much attention to the details since we are using it only as an example of an alternative analysis made with Scilab.

The approach includes the following steps:

- We read the file *base* into Scilab's variable *base*
- We extract the column containing the respiratory signal to the Scilab's variable *res*
- We detect the points that exceed an amplitude of 60 (our signal fluctuates from *-140* to *140*). Now we have a vector (*over60*) that contains the value *1* at the points that are over 60 and a value *0* in each other point
- We detect the transitions from *0* to *1* by subtracting from each point the previous one.
- We store the transitions as an index vector (called *ndx*). This index stores the point where a trigger detecting the level of 60 in increasing direction would be placed.

Let us see the code (note that Scilab code is less verbose than the explanation):

```
-->base = fscanfMat("base");             // reading the file

-->res = base(:,14);                     // extracting column 14

-->over60 = 1*(res>60);                  // values over 60

-->changes = over60(2:$)-over60(1:$-1);  // changes can be -1, 0 ,1

-->ndx = find( changes >0 )              // detection of values 1
 ndx  =

!   342.    884.    1435.    1936.    2445.    2957.
```

Let us check the result by plotting the detection with the respiratory signal

```
-->plot(res)                              // the respiratory signal

-->plot2d(ndx,res(ndx),-3)                // plotting ndx as points
```

The result can be seen in figure 4.2.

As we can see, the marked points act as a trigger. But if we want to see the result in *wave*, it is not an easy task: we have to create a WFDB annotation file.

4.3 Trying to create an annotation file with the result

To create an annotation file, our first thought is using *wrann*. Given our success by using *wrsamp* we could try this approach.

Let's see the information about *wrann*:

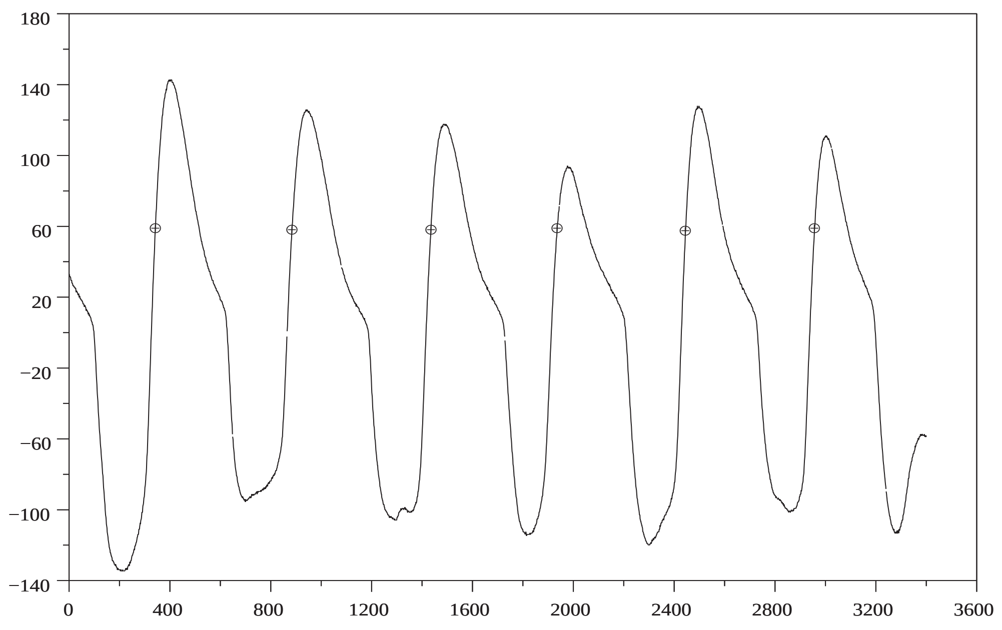

Figure 4.2: The respiration signal marked with the detection

```
The usual application for wrann is as an aid to annotation file
editing: an annotation file may be translated into ASCII format
using rdann, edited using a text editor, and then translated back
into annotation file format using wrann.
```

So *wrann* has been conceived to allow editing with a text editor. It seems a dead end. We would like to have a tool to create annotations easily from Scilab. We could follow the following approaches:

- We can understand the annotation format (it is completely documented) and program an application from scratch, even using Scilab. It would not be very convenient. It is a difficult task and we try to use as much previous code as possible.
- We could program an application by using the WFDB library. It is a very professional approach. It is difficult but the result is the best. We could even send our code to PhysioNet and, eventually, it could be considered for wider diffusion
- We could modify *wrann* to be adapted to our needs. It is much simpler. If we need to create a lot of annotations, probably it is the best choice but we will have a non-standard application
- We could create some output that mimics the output of *rdann* by using an editor. It is the simplest approach.
- We could try to understand the machinery of *wrann* to create some file that can be understood by *wrann*

Having a lot of possibilities increases the probability of error (Do you remember the simple camera of the introduction?). WFDB is open source software. Let us explore the code of *wrann*. After some searching, we reach to the core of the interface with the external files (lines 111 to 113):

```
    while (fgets(line, sizeof(line), stdin) != NULL) {
p = line+9;
if (line[0] == ’[’)
    while (*p != ’]’)
p++;
while (*p != ’ ’)
    p++;
(void)sscanf(p+1, "%ld%s%d%d%d", &tm, annstr, &sub, &ch, &nm);
annot.anntyp = strann(annstr);
annot.time = tm; annot.subtyp = sub; annot.chan = ch; annot.num = nm;
   ...
```

So, it seems that a line is read and unless the first character of the line is an square bracket it skips the first 9 characters and uses the C function *sscanf* to read a long integer (the position of the annotation), a string (the type) and three integers (the subtype, the *chan* and the *num* field). Probably, knowing the sample, the remainder of the annotation can be reconstructed. We could try to write some code to emulate the input format of *wrann* from Scilab.To do this, we create a string with the format. Let us see the code (the first lines have been repeated for convenience):

```
-->base = fscanfMat("base");
```

```
-->res = base(:,14);

-->over60 = 1*(res>60);

-->changes = over60(2:$)-over60(1:$-1);

-->ndx = find( changes >0 )
 ndx  =

!   342.    884.    1435.    1936.    2445.    2957. !

-->format_ann = "**dummy** %13.0f [15] 0 0 0";   // our discovery

-->fprintfMat("dummy_file",ndx',format_ann);     // store to a file
```

Notice that ***dummy*** has exactly 9 characters. We choose *[15]* because it is an *unassigned annotation type*. Our creature begins to live. Now we have to check the result (we return to the terminal):

```
[j@localhost Code]$ cat dummy_file
**dummy**           342 [15] 0 0 0
**dummy**           884 [15] 0 0 0
**dummy**          1435 [15] 0 0 0
**dummy**          1936 [15] 0 0 0
**dummy**          2445 [15] 0 0 0
**dummy**          2957 [15] 0 0 0
[j@localhost Code]$ cat dummy_file | wrann -r ecg -a res
[j@localhost Code]$ rdann -r ecg -a res
    0:01.710      342  [15]    0    0    0
    0:04.420      884  [15]    0    0    0
    0:07.175     1435  [15]    0    0    0
    0:09.680     1936  [15]    0    0    0
    0:12.225     2445  [15]    0    0    0
    0:14.785     2957  [15]    0    0    0
```

Isn't it incredible? We introduce garbage and we receive annotations properly formatted. Let's check that our method functions:

```
[j@localhost Code]$ wave -r ecg -a res
```

We select the record *ecg.dat* with the annotation file *ecg.res*.

The result can be seen in figure 4.3.

After some arrangements, we were able to introduce the result of the analysis. We have the ECG with the position of the respiratory cycle marked in the same screen. If we had plotted the respiratory signal (a trivial task), we could begin a new cycle of editing and exporting results. Our analysis was trivial (a trigger to detect the respiration cycle), but by using the same approach you can mark anything in the file of your commercial equipment (if it exports ASCII files) and edit the result in *wave*. Let us mention some things that you can mark:

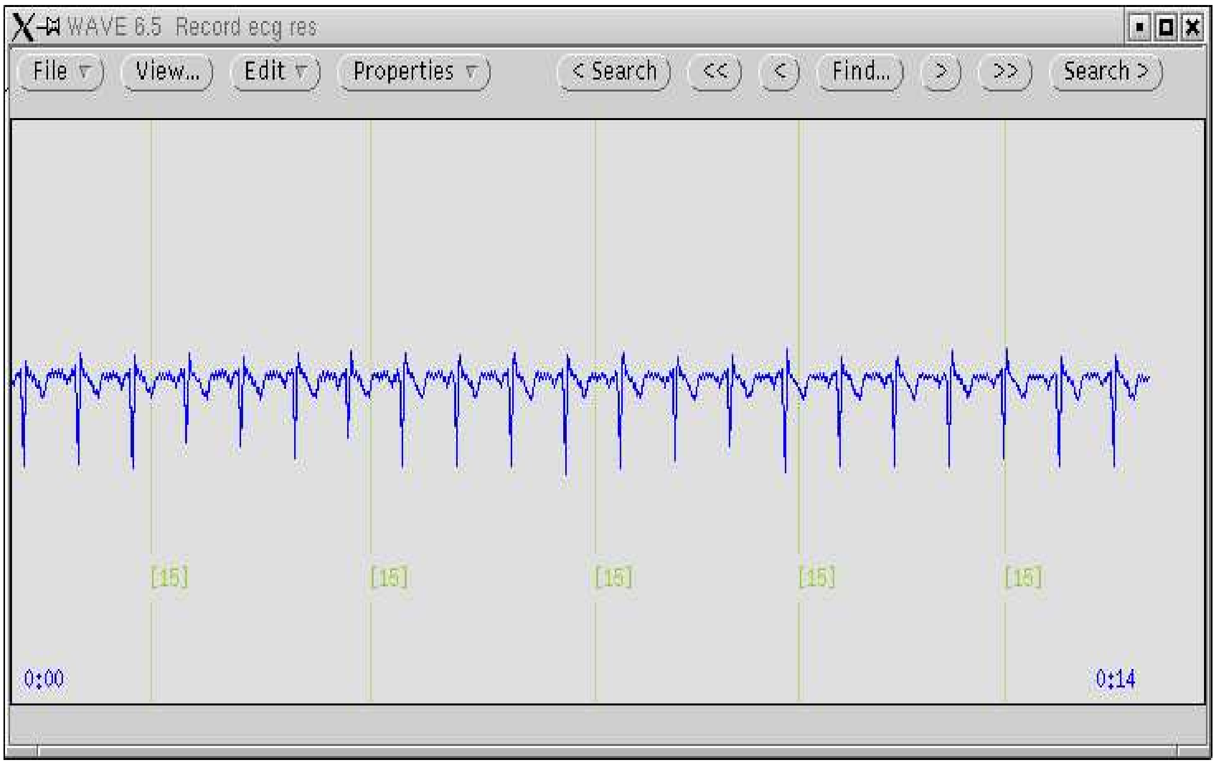

Figure 4.3: In this ocasion the ECG is plotted together with the marks of the respiratory cycle

- arousals
- K-complexes
- spindles
- apneas
- external stimuli...

And many more. Would you not like to see whether the temporal pattern of spindles is modified by drugs? What is the temporal relation between K-complexes and spindles? Do really spindles decrease in slow wave sleep or are they masked by the EEG? Does the auditory evoked potential relate with the sleep stages? These are questions that you can explore with WFDB software.

4.4 Using Scilab as a tool from wave

Wave uses a very ingenious method for further processing. Let us adapt it to use Scilab. We define the editor (any other editor can be used) and call *wave*:

```
[j@localhost Code]$ export EDITOR=/usr/openwin/bin/textedit
[j@localhost Code]$ wave -r 100s
```

Then you click on *File ¿ Analyze* and a *Notice* appears indicating that the file *wavemenu* will be copied in your directory. Once you choose *Copy*, you can begin to edit the menu. At the end you add the following lines:

```
...
# list contains three signals).  Adjust the -rx and -rz options to obtain the
# desired viewpoint.

# Add additional entries below.
Hello Scilab    echo "key= x_message (\"You are reading record $RECORD\" ) " \
                > com.sce; \
echo "exit()" >> com.sce; \
scilab -nw -f com.sce
```

The main features of the connection are illustrated here:

- Scilab is called with the option *-nw*, so it does not initiate a new window.
- Scilab is called with the option *-f*, so it reads the file indicated (*com.sce*).
- We create the file by echoing strings from Wave; notice that the variables from Wave can be passed to Scilab.
- We could indicate to Scilab that any other file (with complex processing commands) must be loaded and executed.

Now you save the file and click *Reread menu* in the *Analyze window*. A new button called *Hello Scilab* appears. If you click it, a small window with a message indicating the file that you are processing appears.

Do not pay too much atention to this subsection. A lot of things can fail with this approach: Scilab must be installed, you have to be in the proper directory, you have to be able to write the file (*com.sce*) and above all the synthax is very difficult. It is much better to call scilab from the command line. This point was included only to show the versatility of *Wave*.

4.5 In summary

In this section we were able to mix the analysis made by external programs with some elementary analysis made by Scilab. Scilab is a general mathematical package, Wave is a friendly viewer with a lot of powerful possibilities, WFDB applications are a set of very well conceived and checked programs for signal and annotation processing and analysis, including many designed for electrocardiographic analysis. In this chapter we made them work together by acting on ASCII files.

Chapter 5

Final considerations

Probably, programmers are not the best-equipped persons to make advances in Clinical Neurophysiology. In our time, a neurophysiological recording is no longer a drawing on paper but a set of numbers in a file. Details about how to handle the content of these files is not a collateral aspect of Clinical Neurophysiology. It is the heart of our activity. Details are important for evaluating results and are even more important when developing new ideas. Nowadays (and probably even more for a foreseeable future), in most fields of activity, knowledge equals software. Some related ideas about formats in Clinical Neurophysiology can also be read in Neurotraces[1]

In the tradition of Windows, a program is a frame with a lot of options. Most of them can be reached by clicking buttons. You do not have to leave the program to obtain the result. Data are represented in some unknown format making them hardly compatible with other programs and the code of the application is not known. In summary, the programmer and the user are different persons. The result is that the user does not know anything about the details of the program.

The classical approach in UNIX-Linux is completely different: small programs (like *sqrs*, *tach* or *wrann*) can be chained to get the result. There can be several tools to do the same task with subtle differences that adapt to our requirements. In certain sense, programs behave like functions, files behave like variables and the resources of the whole computer (or even the network) are accessible and can be tailored to reach our goals. We should not be excessively concerned with the look and feel of the application but with the versatility and compatibility of the application.

Imagine that we create a program (a script file) called *summary* with the following contents:

```
wrsamp -F 200 -G 1000 -i base  -o ecg 9
sqrs -r ecg
wave -r ecg -a qrs
tach -r ecg -a qrs -F 200 > tac
touch prog.sce
echo "base = fscanfMat(\"base\")" > prog.sce
echo "tac = fscanfMat(\"tac\")" >> prog.sce
echo "plot2d((1:size(base,1))/200,base(:,10)/5,1)" >> prog.sce
echo "plot2d((1:size(tac,1))/200,tac,1)" >> prog.sce
scilab -f prog.sce
```

When you execute *summary*, the program

- converts *base* to WFDB format

[1]http://www.neurotraces.com/views

- detects the QRS complex
- calls *wave* to allow review and correction of QRS detections
- extracts the tachogram sampled at 200 Hz once the annotations have been edited
- creates a set of commands that will be called by Scilab
- calls *scilab* and plots the result

In summary, we have a program especially well adapted to our needs, a program in which we control all the intermediate steps, know all the formats used and whose machinery can be easily modified.

But besides the propaganda of UNIX-Linux, the aim of this tutorial was to show the basis of how to adapt WFDB software and Scilab to explore neurophysiological recordings obtained from commercial equipment.

One last comment. When you read something, perhaps you do not appreciate to which extent your opinion is important to the author. I am extremely interested in knowing from you that you arrived to the last lines of this hard text getting over the difficulties installing *wave*, visiting the code of *wrann* and understanding the intricacies of the matrix notation of *scilab*. Please drop me a mail. You can reach me at olivan@neurotraces.com[2]. Be certain that your mail will have been very useful for me.

[2]mailto:olivan@neurotraces.com