TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: GPS data compression through an intensive study of data correlation

TITULACIÓ: Enginyeria Tècnica Aeronàutica, especialitat Aeronavegació

AUTOR: Patrícia Ruiz Rodríguez

DIRECTORS: Enrique García-Berro Montilla, Alberto González Villafranca

DATA: 15/09/2009

TÍTOL**:** GPS data compression through an intensive study of data correlation

AUTOR: Patrícia Ruiz Rodríguez

DIRECTORS: Enrique García-Berro Montilla, Alberto González Villafranca

DATA: 15/09/2009

Resum

El Sistema de Posicionament Global (GPS) és un sistema emprat a molts camps de la tecnologia actual. S'utilitza tant per aplicacions socials com en estudis científics. Precisament algunes d'aquestes aplicacions requereixen l'enregistrament de les dades per ser reenviades després a un altre receptor. En aquests cassos la quantitat de dades que s'han d'enviar és massa gran, el que fa realment útil la compressió d'aquestes dades. L'objectiu principal d'aquest projecte és realitzar un estudi teòric de les dades que guarden i/o envien els receptors de GPS per poder dissenyar un mètode o mètodes, pel moment també teòrics, que generin nous arxius que contenguin exactament les mateixes dades però que ocupin menys espai, es a dir, mètodes que siguin capaços de comprimir els arxius del GPS, anomenats RINEX.

Primerament s'ha estudiat el mètode de compressió Hatanaka, el qual s'aplica actualment als fitxers, per obtenir una idea del possible tractament de les dades. Després s'ha analitzat la relació entre les diferents dades que contenen els arxius RINEX. L'estudi de les dades s'ha realitzat mitjançant scripts de Matlab. Aquests scripts (o programes) generen diversos tipus de gràfiques, que mostren d'una manera més clara els valors de les variables i les relacions entre aquestes. Aquestes gràfiques s'han realitzat amb un gran nombre d'arxius RINEX molt variats per tal d'obtenir un mètode factible per al major nombre d'arxius de GPS possibles. Amb aquest estudi s'han obtingut 4 mètodes diferents de compressió segons el tipus de dades a comprimir i amb l'objectiu de millorar la compressió obtinguda pel mètode de Hatanaka.

De l'estudi fet amb Matlab s'han obtingut uns resultats molt positius que indiquen que la utilització dels nous mètodes de compressió dissenyats en aquest projecte poden millorar notablement la relació de compressió aconseguit pel compressor Hatanaka.

TITLE: GPS data compression through an intensive study of data correlation

AUTHOR: Patrícia Ruiz Rodríguez

DIRECTORS: Enrique García-Berro Montilla, Alberto González Villafranca

DATE: September 15th 2009

Overview

The Global Positioning System (GPS) is one of the more a system that is commonly used in current positioning technologies, due to its social benefits and to its versatility which makes possible to use it in scientific studies. Some of those scientific applications require saving the data to be sent again later to another receptor. In those cases, the amount of data to be sent can be rather high and, thus, compressing this data is sorely needed.

The main objective of this project is to perform a theoretical study of the data that are saved and sent by the GPS receptors in order to design a method, for the moment only theoretical, that generates files with exactly the same data but with a smaller file size.

First of all, to get an idea of how data can be treated, a study of the Hatanaka's method of compression has been done. This method is being used nowadays. The next process has been analyzing the different types of observation that the data can contain and study using scripts written in Matlab the correlations between those data. The scripts or programs generate several plots that show in an easier way the values of the data and the relations between the data of different types. The study has been done with a large number of RINEX files with the aim of obtaining a feasible method for the maximum number of GPS files as possible. After the study four methods of compression have been developed, one method or another will be used depending on the type of observation to compress.

The results obtained in this study are extremely encouraging, in the sense that better compression algorithms can be devised. Specifically, using the new methods developed here the compression ration obtained in this theoretical study is much larger than that obtained using Hatanaka's algorithm.

INDEX

INDEX OF FIGURES

1. INTRODUCTION

1.1 The GPS system

The Global Positioning System (GPS) is an U.S. space-based radionavigation system that provides reliable positioning, navigation, and timing services to civilian users on a continuous worldwide basis, freely available to all. This system was designed by the U.S military, which still operates it. The GPS is made of three segments: the space segment (satellites orbiting the Earth), the control segment (control and monitoring stations on Earth) and, finally, the user segment (GPS receivers owned by users). Although the GPS operational constellation consists of 24 satellites that orbit the Earth in 12 hours (in fact, there are often more than 24 satellites orbiting for redundancy reasons), only four GPS satellite signals are used to compute positions in three dimensions (latitude, longitude, and altitude) and the time offset in the receiver clock. Those 24 satellites stay in 6 orbital planes equally spaced (60 degrees), and inclined around fifty-five degrees with respect to the equatorial plane. Usually there are four satellites at each orbital plate. This system configuration provides the user with between five and eight satellites visible from any point on the Earth.

As of January 17th 1994, the complete constellation was in orbit and since April 29th 1995, GPS became fully operational. GPS receivers convert the signals that receive from the satellites into position, velocity, and time estimates. Satellites generate two microwave carrier signals: L1 frequency (1575.42 MHz) carries navigation message and the SPS (Standard Positioning Service) code signals. L2 frequency (1227.60 MHz) is used to measure the ionospheric delay by PPS (Precise Positioning Service) receivers. This system gives precise positioning with an approximate accuracy of a few millimeters and a few nanoseconds time (UTC). Nevertheless those values are typical for professional and military GPS receivers, not for civilian commonly used GPS receivers. Due to this high accuracy GPS is used at ground vehicles, ships and even aircrafts.

Although navigation is the most famous application of the GPS, there are many more applications for this system that are not so well known. Nevertheless they are as important (or even more) as navigation applications. For example, applications in public safety and security areas, geophysical studies or disaster relief and emergency services that depend upon GPS for location and timing capabilities in their life-saving missions. Another area where GPS is providing significant advances is geophysical exploration. These exploration methods require high accuracy positioning, that GPS can afford, especially those that require subsequent data reduction or processing. For example new systems that use GPS allow the study of the movement of glaciers with a high precision, but this just an example and many more can be envisaged.

Finally environmental studies are also being benefited by systems which use GPS. Accurate tracking of environmental disasters such as fires and oil spills makes easier fighting against those disasters. On the other hand, the monitoring and preservation of endangered species can be facilitated through GPS tracking and mapping. These are only a few of the many environmental applications of GPS.

1.2 GPS compression

The common use of GPS is the navigation system. It works perfectly due to the fact that the signals with which receivers work do not have to be saved, since they are just used to determine the position. However for geophysical applications, where the studies require a high number of receivers in a huge stretch of land, the signals to be stored and subsequently sent to another receiver become a problem as a large bandwidth is required to send all this data. That is the reason why a compression method for the GPS data was developed. This method was first developed by Yuki Hatanaka, therefore the name: Hatanaka's compression format. Although bandwidth keeps being quite small for some geophysics applications there are many other in which large amounts of data need to be transmitted to a remote location and no other compression method has been yet developed.

1.3 Motivation

Taking as a reference some advantages developed for the Gaia mission, we aim to improve the GPS data compression. Due to the fact that the Hatanaka compression scheme is not powerful enough for some studies, designing a new compression system to get a higher compression ratio is mandatory. The system developed will not have a direct application to most of the typical GPS applications as navigation or other environmental applications, but it is of relevance for many other areas. Specifically, the main use will be for processes for which a big amount of data has to be sent. More particularly, in the area of geophysics these kinds of studies are sorely needed. An example of this is a current project of the IEEC (Institut d'Estudis Espacials de Catalunya) that aims to study the movement of glaciers in Greenland. Within this project, several GPS receivers are placed in designated locations. These receivers send the data through the Iridium constellation (thus, using a satellite link). If this application becomes successful it could be used subsequently into other areas.

During the development of the project two different tools have been used. The first one used was the python programming language. Using it a script was created to split one GPS file (in RINEX format) into different files containing each a specific part of the data. The second tool is Matlab, which analyzes the values of the RINEX file, thus obtaining a large variety of graphics and histograms. They have enabled us to develop and polish our pre-compression method.

1.4 Structure of the work

First of all it has been necessary to make a short study about the Rinex file, its structure, the variables that appear, how the code determines which data is processed at each line and what are the codes used at the file. This is done in section 2. It follows section 3, where we study and describe the Hatanaka pre-compressor. Once the Hatanaka study has been completed, some new pre-compression methods have been developed trying to improve the ratio of

compression achieved using the Hatanaka method. All the new methods developed have been tested for all the different types of observations in a RINEX file and, after that, the best method for each one of the observations has been chosen. The way in which this is done is explained in chapters 4 and 5, which respectively describe the pre-compression strategies and the way in which some of the most relevant variables are correlated. Finally, in chapter 6 we summarize our main results and conclusions are drawn.

2. THE RINEX FILES

The Astronomical Institute of the University of Bern developed the first proposal for the Receiver Independent Exchange Format, more commonly know as RINEX, for the easy exchange of GPS data to be collected during the large European GPS campaign EUREF 98. The proposal involved more than 60 GPS receivers of 4 different manufacturers. The governing aspect during the development was the fact that most geodetic processing software for GPS data uses a well-defined set of observables:

- The carrier-phase measurement at one or both carriers (actually being a measurement on the beat frequency between the received carrier of the satellite signal and a receiver-generated reference frequency).
- The pseudo-range (code) measurement, equivalent to the difference of the time of reception (expressed in the time frame of the receiver) and the time of transmission (expressed in the time frame of the satellite) of a distinct satellite signal.
- The observation time being the reading of the receiver clock at the instant of validity of the carrier-phase and/or the code measurements.

Usually the software assumes that the observation time is valid for both the phase and the code measurements, and for all satellites observed. Consequently, all these programs do not need most of the information that is usually stored by the receivers. Specifically, they only need phase, code and time in the above mentioned definitions, and some station-related information like station name, antenna height, and a few others.

2.1 Format description

Currently the RINEX format consists of six ASCII file types:

- Observation Data File
- Navigation Message File
- Meteorological Data File
- GLONASS Navigation Message File
- GEO Navigation Message File
- Satellite and Receiver Clock Date File

Each of those files consists of two sections: the header and the data section. The header section contains global information for the entire file and is placed at the beginning of the file. The header section contains header labels in columns 61-80 for each line contained in the header section. These labels are mandatory and must appear with a specific format which is always the same for all RINEX files.

The format has been optimized for minimum space requirements, independently of the number of different observation types of a specific receiver, by indicating in the header the types of observations to be stored. In computer systems

allowing variable record lengths the observation records may be kept as short as possible. For instance, trailing blanks can be removed from the records. The maximum record length is 80 bytes per record. Each observation file basically contains the data from one site and one session.

Besides the fact that there are 6 different types of RINEX files only the observational data file is studied in this report. This is because this type of file includes the more important information and it is usually the larger in size. Hence, the entire document refers only to observational files. In Appendix A of this document, an example of an observational RINEX file can be found and also a table showing what every value in a RINEX file is.

As can be seen in Appendix A, every character or value in a RINEX file has a specific meaning. There are different types of values in a RINEX file, which are defined in the tables of this appendix. It is worth mentioning that the format of these files is adapted to the conventions of the FORTRAN programming language. All the values are listed below:

- F (real numbers): F9.2, a floating point number (a real number) with a total of 9 characters (including the point), and two of those are the decimals. For example: 123456.78

- A (ASCII characters): A3, a string with three characters (including spaces). For example: ABC

- 2A20: two strings of 20 characters each one.

- x (spaces): 11x, eleven spaces.

- I (integer): I5, an integer number (without point or decimals) with a total of 5 characters. For example: 87654

- D (exponential)

2.2 The observables

There are three main types of observations that must be known, those are time, phase and pseudo-range. We describe them one by one in the following subsections.

2.2.1 Time

The time of the measurement is the receiver time of the received signals. It is identical for the phase and range measurements and is identical for all satellites observed at that epoch. It is expressed in GPS time (not in Universal Time).

2.2.2 Pseudo-range

The pseudo-range (PR) is the distance from the receiver antenna to the satellite antenna including receiver and satellite clock offsets (and other biases, such as atmospheric delays). The PR can be defined as:

$$PR = d + c \times (rco - sco + ob) \quad (1.1)$$

where, d is the distance, c is the speed of light, rco is the receiver clock offset, sco is the satellite clock offset and ob are the other biases. Thus, the pseudo-range reflects the actual behavior of the receiver and satellite clocks, and it is stored in units of meters.

2.2.3 Phase

The phase is the carrier-phase measured in entire cycles at both L1 and L2 (see below). The half-cycles measured by squaring-type receivers must be converted to entire cycles and flagged by the wavelength factor in the header section. The phase changes in the same sense as the range (negative Doppler). The phase observations between epochs must be connected by including the integer number of cycles. The phase observations will not contain any systematic drifts from intentional offsets of the reference oscillators. The observables are not corrected for external effects like atmospheric refraction, satellite clock offsets, and others.

If the receiver or the converter software adjusts the measurements using the real-time-derived receiver clock offsets dT(r), the consistency of the 3 quantities phase, pseudo-range and epoch must be maintained, i.e. the receiver clock correction should be applied to all 3 observables:

$$\text{Time (corr)} = \text{Time(r)} - dT(r) \quad (2.2)$$
$$\text{PR (corr)} = \text{PR(r)} - dT(r) \times c \quad (2.3)$$
$$\text{Phase (corr)} = \text{phase(r)} - dT(r) \times f \quad (2.4)$$

being f the frequency.

2.2.4 Doppler

The sign of the Doppler shift as additional observable is defined as usual: positive for approaching satellites.

2.2.5 Abbreviations

All the types of observations in a RINEX file are abbreviated with the names L1, L2, C1, P1, P2, D1, D2, S1, S2, T1 and T2. The meaning of each one of these is:

- L1 and L2: phase measurements on L1 and L2
- C1: pseudo-range using C/A-code on L1
- P1 and P2: pseudo-range using P-code on L1 and L2
- D1 and D2: Doppler shift on L1 and L2
- T1 and T2: transit integrated Doppler on 150 (T1) and 400 MHz (T2)
- S1 and S2: raw signal strengths or SNR values as given by the receiver for the L1 and L2 phase observations

Although there are many types of observables not all files contain all of them. Nevertheless, there is one type of observation that appears in all RINEX files, the phase L1. Also they do not always appear in the same order: the position of appearance is always defined at the headers. It is important to note here that there are some types of observations that can appear only during certain periods. Usually this is due to a loss of signal.

2.2.6 Flags

Each observation data can contain two numbers after each value. Those numbers are called flags or LLI (Loss of Lock Indicators). Each flag has a range of values and a meaning depending on its value. Specifically, the first flag can have a value between 0 and 7. This number translated to a binary number consists of three bits. For example, if the first bit is the number 3, it will represent the binary 011. If the flag is a 7, the binary number will be 111. Thus, if any of the bits of the binary number is 0, it will mean that the data is fine or that is not known if there is loss of lock. If any of the bits is 1 it means:

- Bit 0 set to 1: lost of lock between the previous and the current observation, a cycle slip possible.
- Bit 1 set to 1: Opposite wavelength factor to the one defined for the satellite by a previous wavelength fact L1/2 line. Valid for the current epoch only.
- Bit 2 set to 1: Observation under anti-spoofing (may suffer from increased noise).

The second flag represents the signal strength projected into an interval from 1 to 9:

- 1: Minimum possible signal strength.
- 5: threshold for good S/N ratio.
- 9: Maximum possible signal strength.

If the second flag is 0 it means that the strength it is not known or it does not matter.

3. THE HATANAKA COMPRESSION METHOD

The Hatanaka method uses a third order filter to compress each one of the columns of numeric data. This filter consists in a differential method. Let

$$Y_i^0 \tag{3.1}$$

for i=1,…,n be a column of numeric data of a specific type of observation, where the superscript 0 represents the order of difference. The first order filter consists on the difference between the actual value minus the previous one:

$$Y_i^1 = Y_i^0 - Y_{i-1}^0 \tag{3.2}$$

So, an m-order filter would be:

$$Y_i^m = Y_i^{m-1} - Y_{i-1}^{m-1} \tag{3.3}$$

For i=2,…,n. This operation reduces considerably the size of the columns values and it is a lossless pre-compression method. This means that the original data can be recovered without any loss of data. The recovering process would be as follows:

$$\begin{aligned} Y_i^{m-1} &= Y_{i-1}^{m-1} + Y_i^m \\ &\cdots \\ Y_i^1 &= Y_{i-1}^1 + Y_i^2 \\ Y_i^0 &= Y_{i-1}^0 + Y_i^1 \end{aligned} \tag{3.4}$$

The compression process is schematically indicated by the red arrows in figure 3.1. The decompression process is also indicated in the scheme by the blue arrows.

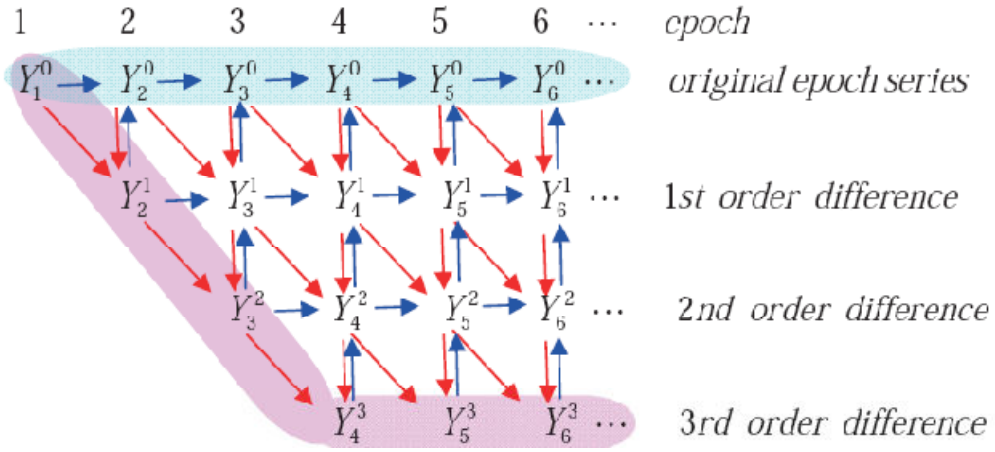

Fig. 3.1: Hatanaka's compression and decompression scheme.

Although this method remarkably reduces the size of the RINEX files it is not enough powerful for some applications. That is the main reason why a more elaborated compression method is really useful. Using as a starting point Hatanaka's compression method, an alternative method is developed in the next sections.

4. RINEX DATA STUDY

To get a better understanding on how to compress RINEX data, a thorough study of the techniques employed in the only current compressor in use, e.g., the Hatanaka compressor, has been done. We have used Matlab to study the correlations between the GPS data in a RINEX file. It is to be noted that to perform this study not all the information in a RINEX file is necessary. In fact, only the information in a specific part of the RINEX file is relevant. The reason for this that the largest data to be pre-compressed, an ultimately, compressed is the physical data itself. Consequently, we put our attention on all the observation data, and we disregard flags, headers and sub-headers. Additionally, we mention that in order to produce a satisfactory compression ratio it is necessary to have all the data from each satellite separated. In this way, the pre-compression rules are much simpler and the results obtained are better. This, in turn, requires that the pre-compression and compression stages in a RINEX file must be done separately for the data received from each satellite.

4.1 Preparing the RINEX files

To prepare the RINEX files for a convenient study of the cross-correlations present in the data a script using the python programming language has been devised and implemented. Python is an open source program and it is included in most of the Linux operating systems. Additionally, it can also be used in Windows. In addition, python is very versatile and works best with files in which strings are present. These are the reasons why chose this programming language. The script must be executed from a terminal, calling also the name of the RINEX file which is to be split either in Linux or Windows platforms. After executing the script, some new files are created in the working folder. Each of these new files will have the data of only one satellite, without any headers, sub-headers or comments, and the name of those files will be the same as the original RINEX file followed by the number of the corresponding satellite. Thus, if a RINEX file had N satellites, after executing the script there will be N different files. There will also be N additional files which include the flags of each of the satellites and, finally, there will be an additional file which includes all the headers and comments of the original RINEX file.

The files including the observational data are organized in columns, where each column contains one type of observation for one satellite. The order of the columns is L1, L2, C1, P1, P2, D1, D2, S1, S2, T1, and T2. Some RINEX files do not have values for some of the parameters. In this case, the columns in the new file in which some of the parameters are missing are arbitrarily replaced by a ‘0’. To avoid confusions this character can be set to ‘&’, and this is possibly the best option. However, within the scope of this document and for the purpose of studying the correlations between different data in a RINEX file a ‘0’ is sufficient. By construction, all the data in the same row belong to a given instant of time. So if the first data row happened at, say, time t the next data row occurs at time $t+\Delta t$, the next at $t+2\Delta t$ and so on.

Figure 4.1 shows the above described process for a specific RINES file called RINEX123.09o. As can be seen, the use of rinex2raw.py produces a set of new files: 1RINEX123.09o.sbh, NRINEX123.09o.flg, and NRINEX123.09o.obs, each containing, respectively, the headers and sub-headers, the flags and the observations for each one of the satellites in the original RINEX file (N). Once these data files have been created the study of the cross-correlations in the observational data using the Matlab scripts can be done. In passing, we mention that the listing of the python code can be found in Appendix B of this document.

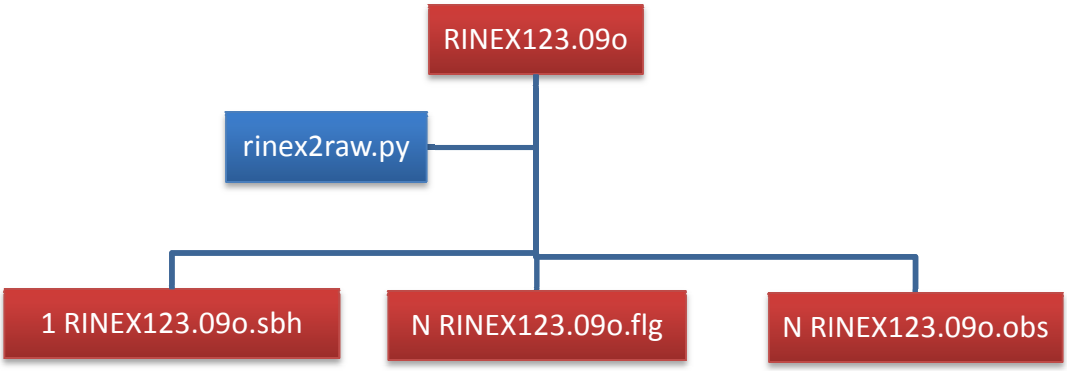

Fig. 4.1: Input-output files scheme for the script rinex2raw.py.

4.2 Matlab scripts

As already said, to analyze the correlations present in an arbitrary RINEX file we have devised and implemented three Matlab scripts, which read the RINEX file created as described in section 2.1 and produce a rich variety of plots. The complete listings of the three Matlab scripts can be found in Appendix D of this report. The function Rinex_hist(file, color, orden, divisiones, psoph, type, adim, lim1, lim2).m has been programmed to produce the histograms that reveal the correlations between the values of an observable and how many times this value is inside a file.

The function Rinex_hist produces different histograms depending on the order of the filter used for the pre-compression stage. To run this function it is thus necessary to define the order of the filter to be used for the pre-compression stage and how many columns of values will be represented. Additionally, it is also necessary to introduce which observable will be studied and if we want to produce a non-dimensional plot or not.

4.3 Matlab results

Figures 4.2 to 4.5 show the results obtained with specific values for the pseudo-range of a satellite using filters of different orders.

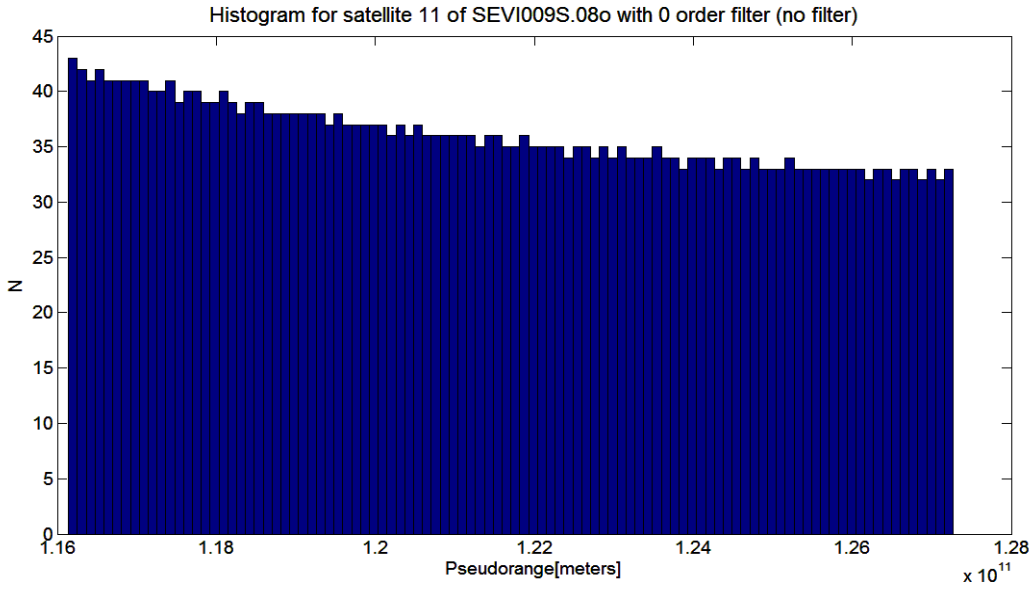

Fig. 4.2: Histogram for the satellite 11 of file SEVI009S.08o without filter.

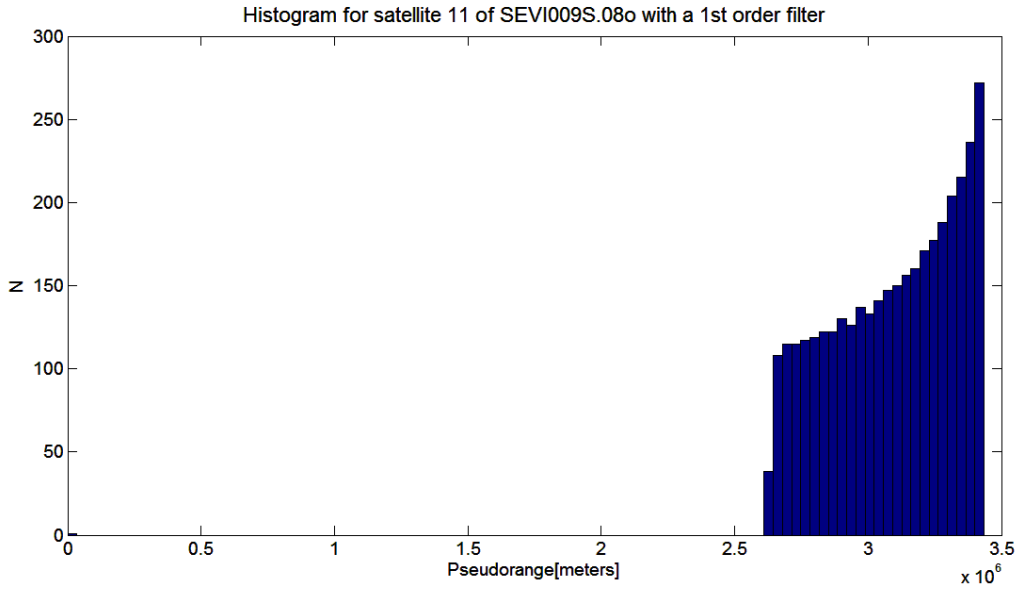

Fig. 4.3: Same as figure 4.2 with a first order filter.

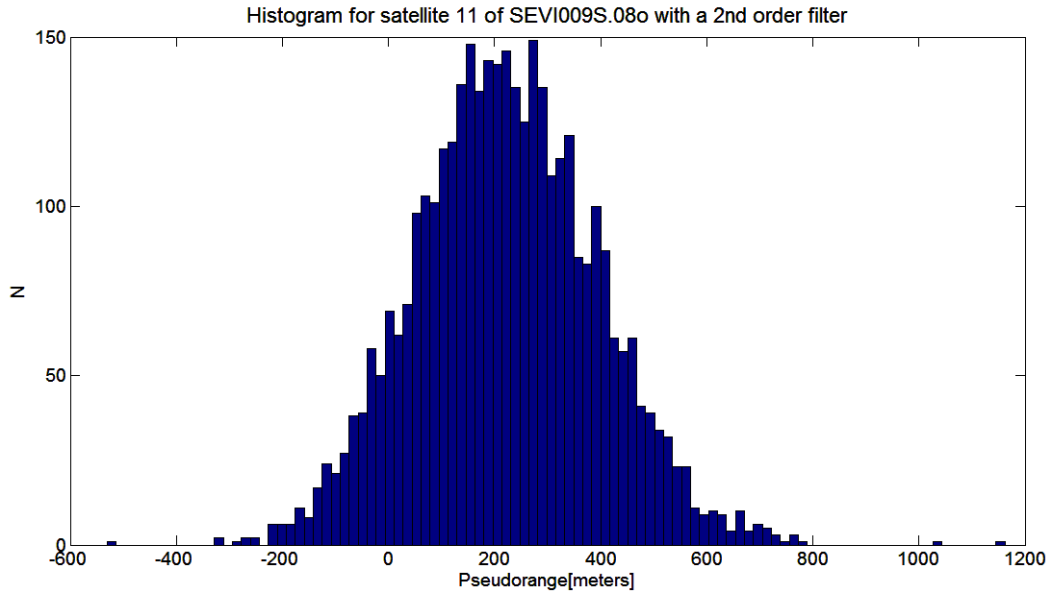

Fig. 4.4: Same as figure 4.2 with a second order filter.

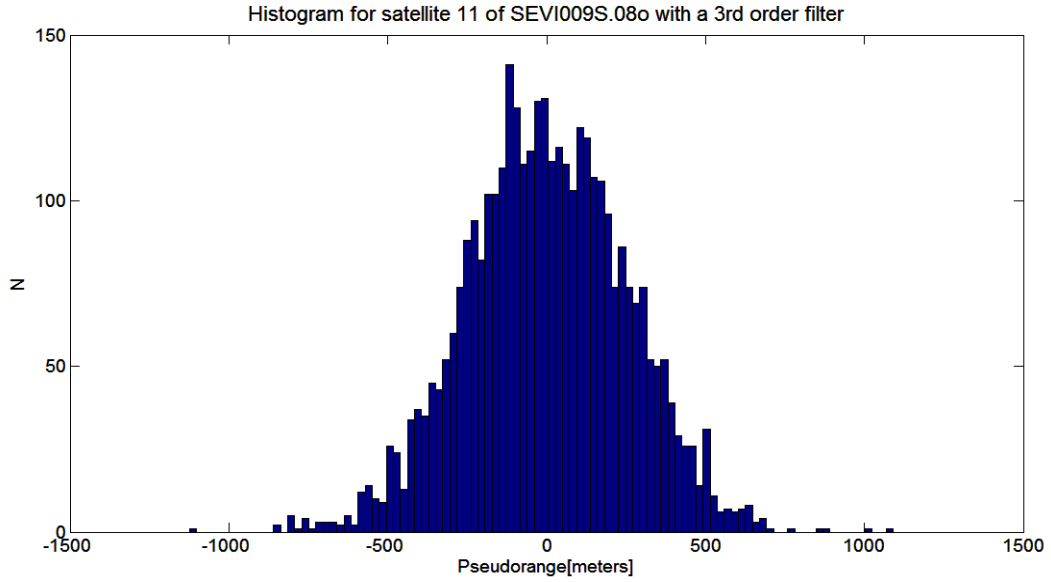

Fig. 4.5: Same as figure 4.2 with a third order filter.

From the histograms shown above (Fig. 4.2 to Fig. 4.5) it is quite evident how the Hatanaka pre-compression scheme works. For instance, figure 4.2 shows that the data in this file is smoothly varying. If we wanted to encode directly the values of the observable (in this case the pseudo-range in meters) a large number of bits would be needed. Applying a first order filter, see figure 4.3, the histogram of values for the pseudo-range is shifted to much smaller values, meaning that indeed the differences between consecutive values of the pseudo-range are rather small. In particular, in the original RINEX file the typical values of the pseudo-range were of the order of 10^{11}, whilst now we find values of the order of 10^{6}. Thus, a smaller number of bits is needed to encode the same

information. Using a second order filter reduces even more the amount of information to be coded, see figure 4.4. Now, when a second order filter is used, the typical values of the encoded pseudo-range are $\sim 10^2$. But the results obtained using a second order filter can still be improved. Applying a third order filter to the data (see figure 4.5) also results in typical values of 10^2, but the distribution of values turns out to have better statistical properties. In particular, when the second order filter is applied, the distribution of values is not centered on zero, whereas using the third order filter the distribution is almost symmetrically centered on zero. Thus, fewer bits will be needed to encode the same amount of information. Additionally, when the third order filter is used, the distribution of values is narrower. Consequently, the higher the order of the filter applied to the original data the better the pre-compression ratio should be. However, we will show in subsequent sections that this ratio can be improved analyzing the properties of the data.

4.4 Improving the Matlab study

Although the distributions shown in the previous section show that indeed satisfactory results can be obtained in the pre-compression stage, it is rather evident as well that they have been obtained for just one satellite, and thus these results may not be totally representative and cannot be extrapolated. Therefore, it turns out that is necessary to study data issued from more than one satellite. As the results from different satellites cannot be pained at the same histogram, because one histogram bar will cover the bar from the previous one, we use only a line for each satellite, instead of histograms.

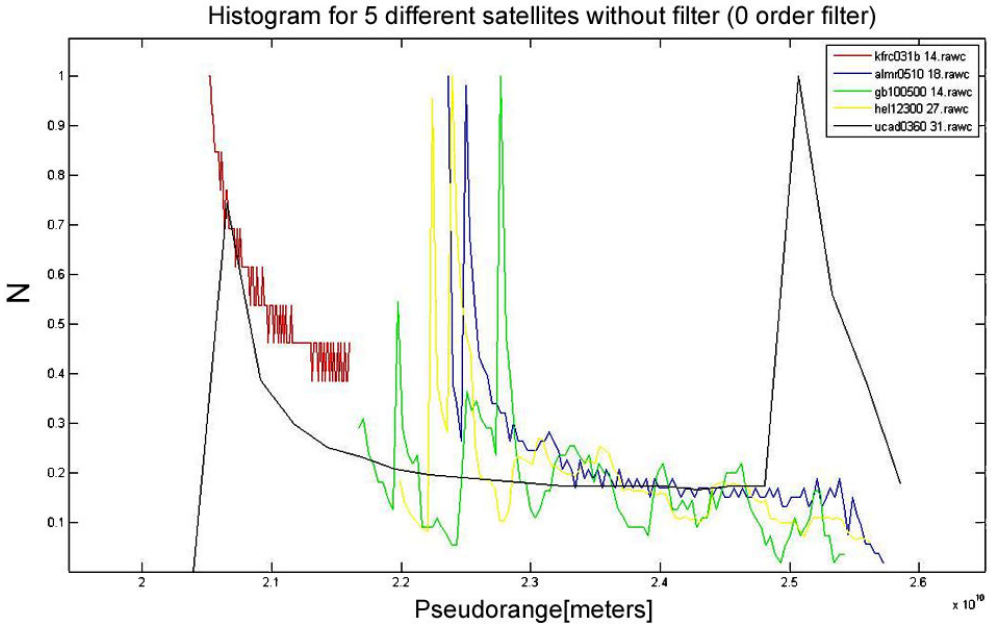

Fig. 4.6: Distribution for 5 different satellites without filter ($\times 10^{10}$).

Figures 4.6 to 4.9 show the data for 5 different satellites employing filters of different orders. Using our Matlab script it is possible to represent as many as wished.

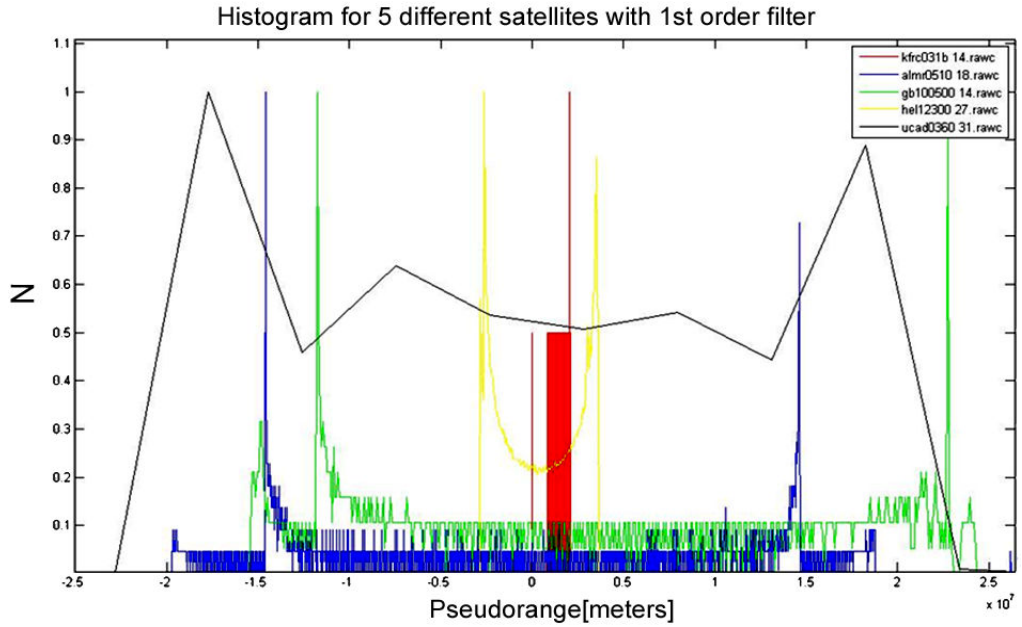

Fig. 4.7: Distribution for 5 different satellites with a first order filter ($\times 10^7$).

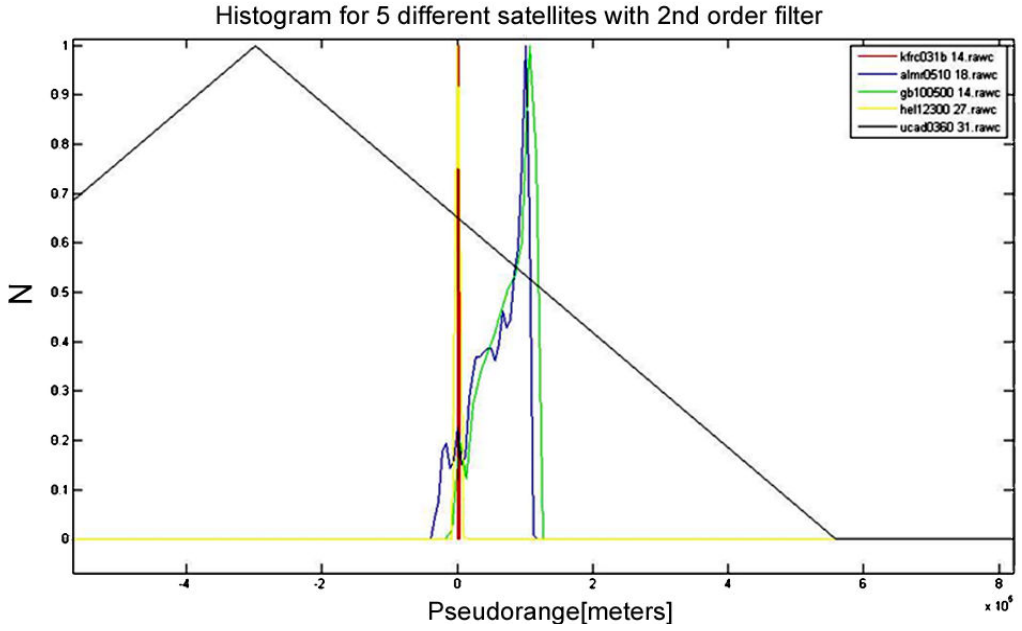

Fig. 4.8: Distribution for 5 different satellites with a second order filter ($\times 10^6$).

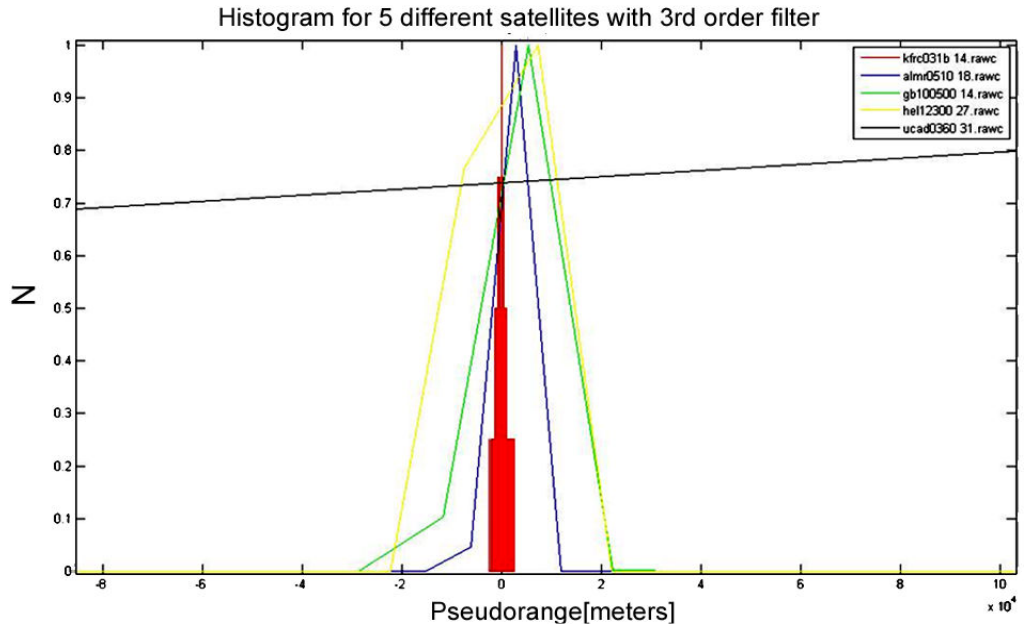

Fig. 4.9: Distribution for 5 different satellites with a third order filter ($\times 10^4$).

A first look at these plots at these graphics reveals that the range of the pseudo-range is reduced when filter of higher order are applied. For instance the range for the order zero order filter goes approximately from 2.0×10^{10} to 2.6×10^{10}, while for the third order goes from -8×10^4 to 10^5. Another visible fact is that also while the order increases the values themselves of the pseudo-range are smaller. Again, we see that increasing the order of the filter has a positive influence in the distribution of values and that many bits will be necessary to code the same information. Additionally, as it was the case in the previous section the third order filter gives data around 0 whereas the rest of filters of lower order do have always some offset. Hence, the third order filter keeps a mean value of 0 and it is for this reason it is the best one to use for the pre-compression stage, because at least there is no obvious redundancy present in the data (e.g. offsets).

Now an obvious question can be posed. Why we do not implement an even higher order filter? In fact, it is quite natural to think that next orders should give better results. We developed a fourth order filter to test this assumption. The result is shown in figure 4.10. As can be seen in this figure, the fourth order filter also provides values around 0 but the range of values of the pseudo-range is larger than that obtained using a third order filter. This undesirable characteristic of the fourth order filter does not allow to obtain better pre-compression ratios for the RINEX files. This result is striking and can be attribute to the fact fact that the values obtained using a third order filter do not have almost offset, so when the filter subtracts the results of the third filter the results are more spread along the x axis. Thus, within this study we will only use third order filters or filters of smaller order.

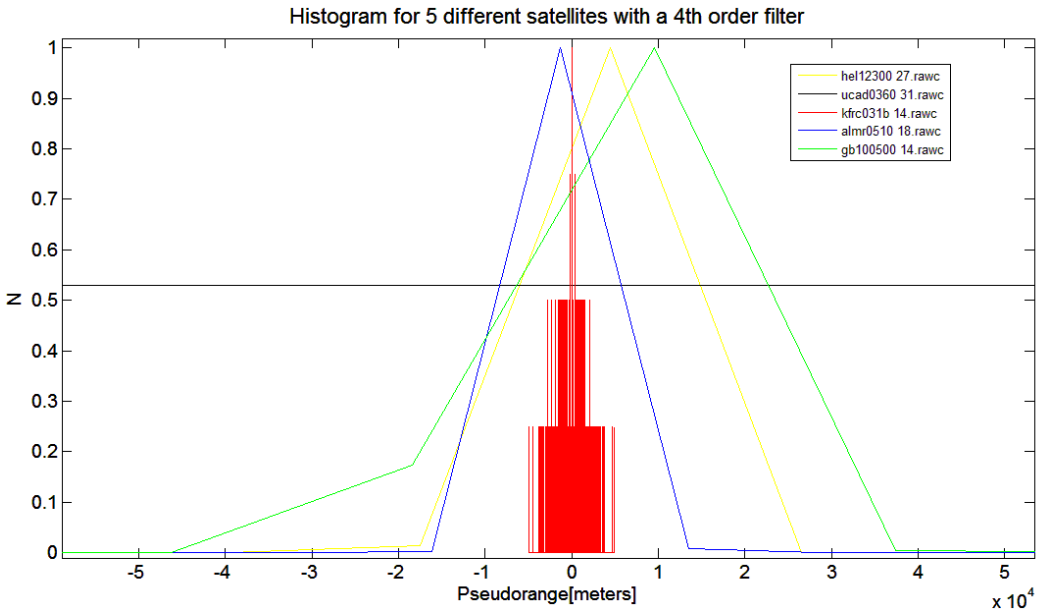

Fig. 4.10: Histogram for 5 different satellites with a fourth order filter.

Now that the distribution of values using filters of different orders has been studied and understood, we turn our attention to an alternative method. Hatanaka uses the third order filter to pre-compress all the values in a RINEX file. In the following section other methods will be developed and test. Some of these methods use the same filter than the Hatanaka pre-compression algorithm does, other methods use lower order filters and, finally, some other do not use filters at all.

5. COMPRESSING DATA USING A PREDICTION METHOD

In this section we present alternative methods to improve the ratio of compression of the Hatanaka pre-compression algorithm. We will use the fact that there are close correlations between several types of observables. This fact allows the prediction one observable, once the other is known, to a certain accuracy. Obviously, the prediction will not be exact and, consequently, we will have to store the corresponding error, but it turns out that if the prediction is sufficiently accurate the stored error needs fewer bits to be coded.

For this purpose another script has been developed. This Matlab script tries to find a correlation between the pseudo-range and the phase, for example, of a RINEX file, or another 2 different types of observations. Using this function the plots obtained will show how many times in a RINEX file the same value of the pseudo-range and the same value of phase appear. Using this information a two-dimensional histogram is plot. Figure 5.1 shows the results obtained with the implemented script.

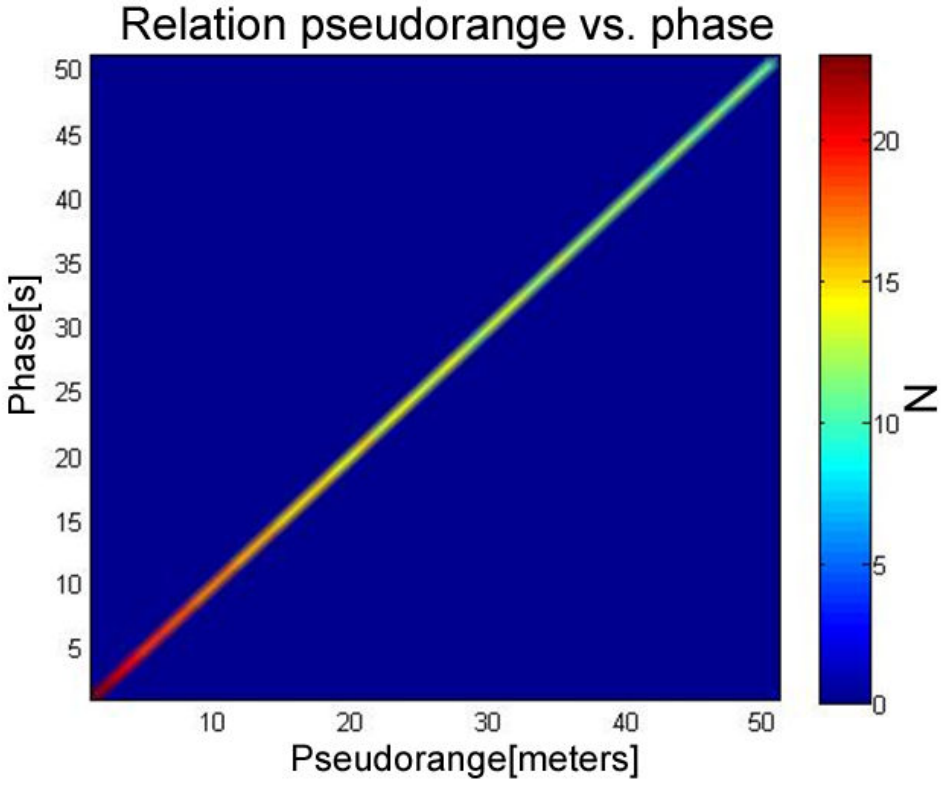

Fig. 5.1: Correlation between the pseudo-range and the phase.

As can be seen in figure 5.1, where the correlation between de phase and the pseudo-range is plotted, there is a very close correlation between those two variables. In fact the correlation is so tight that most of the observational points are on a straight line with a slope of ½. Moreover, most of the values of these

two observables lie in the lower left corner. This can be appreciated by considering the density of points on this plane. The color of a certain pixel in the figure reflects the density of points, according to the scale on the right. This very clear linear correlation between the phase and the pseudo-range will be used below to better pre-compress the data.

This means that knowing at a certain moment one either the pseudo-range or the phase, the other one can be predicted quite easily with a very small error. For example, if the next data is to be compressed:

L1	C1
12345678	34567
12356789	34589

From the first row a ratio can be extracted dividing L1 by C1. From now on this ratio will be called C. C will be calculated for each data to predict with the values of the previous data. In this case the value of C would be:

$$C = 12345678 / 34567 = 357 \quad (5.1)$$

To predict the value of C1 in the second row the operation to be done is:

$$L1 / C = abs(12356789 / 357) = 34598 \quad (5.2)$$

The absolute value is taken to avoid loosing data in the following steps. Finally the value which would be used as a compressed value would be the error of this prediction, which is:

$$34598 – 34589 = 9 \quad (5.3)$$

The value ‘9’ would be saved as a compressed value, which obviously is much smaller and needs fewer bits to be encoded.

One of the advantages of this method is that already in the second row of the file the full compression can be done, whereas when third order filter was used 4 rows were needed to reach the best compression ratio. The most important advantage is, however, that usually the ratio of compression reached using this method is better than the one achieved using the third order filter method. To prove this another Matlab script called comparativa(file,apredecir,orig) has been designed. This script plots 2 graphics together, one for the results obtained using the prediction method and the other one for the results obtained using the

third order filter. The code of this script can also be found in Appendix D. Figure 5.2 is an example of the output of this script.

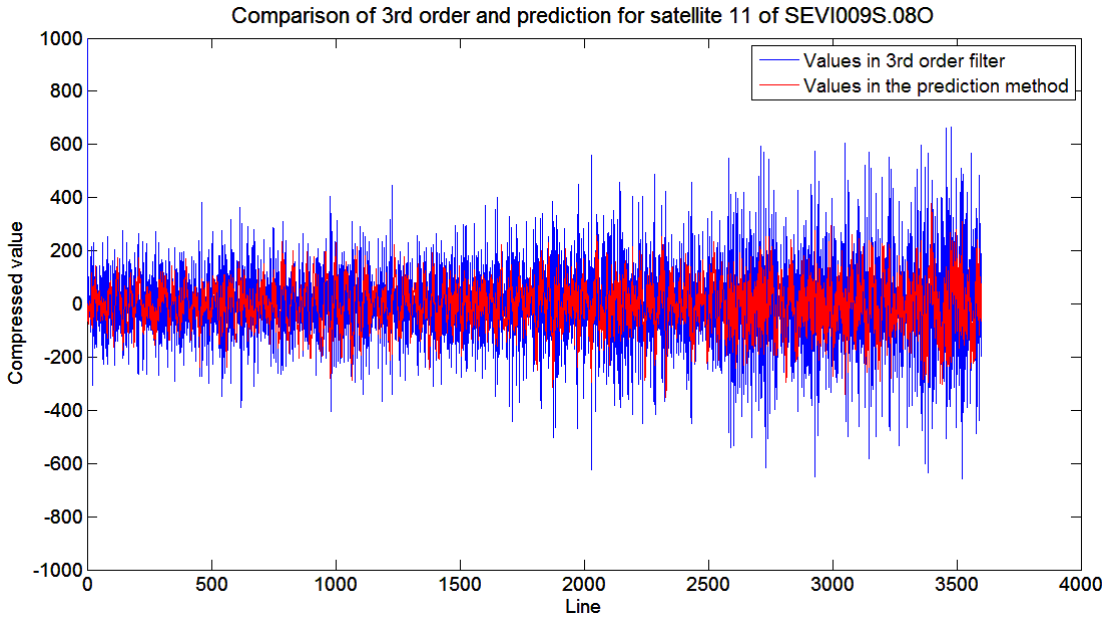

Fig. 5.2: Comparison of the third order filter and the prediction method.

As can be seen in figure 5.2, the values obtained using the prediction error method are usually smaller than the values obtained when the third order filter is used. To have a more quantitative assessment of the quality of our approach the Matlab function also calculates the mean value of all the absolute values obtained with the prediction method an it does also the mean value of all the absolute values obtained with the third order filter, and finally it divides those two values, obtaining a final ratio. This ratio indicates which of both pre-compression schemes performs better. If the value obtained is larger than one it means that the prediction method is better than third order filter.

Fig. 5.2 shows only the results for just one satellite of only one RINEX file. To have more physically sound results the value of the previous comparison of pre-compression schemes has been tabulated for several RINEX files obtained elsewhere. An example of this procedure is shown in table 5.1, where we show the ratio between the two pre-compression schemes for several RINEX files. Each one of the rows in this table corresponds to one RINEX file, and each column is one of the possible types of predictions. For example, if the column name is L2 ← L1 it means that L2 is being predicted using L1. A color code has been applied in such a way that cells in blue correspond to a better performance of the prediction error method, whereas cells in red correspond to a better performance of Hatanaka's method. Similar tables for a large variety of RINEX files can be found in Appendix C of this document.

	L2 ← L1	C1 ← L1	P1 ← L1	P2 ← L1
almr0510_18.rawc	575792,05	137128,47	-	95469,54
DPCA001K_21.rawc	-	314697,70	-	-
esqu0630_16.rawc	89,008	0,096	0,096	0,096
gb100500_14.rawc	-	4,712	-	-
gc082360_12.rawc	-	6,759	-	-
hel12300_27.rawc	1360154,51	18356,80	22397,57	19885,01

Table 5.1: Example of comparative table.

As can be seen in table 5.1, for the Rinex file esqu0630_16.rawc the prediction of the C1 from the L1 is bad, whereas for the same file, the prediction of L1 from L2 produces results which are somewhat better than those obtained using the third order filter.

As already mentioned, for the sake of conciseness we do not reproduce here the full results obtained using this procedure and we refer the interested reader to Appendix C. However, an analysis of the results presented in that appendix demonstrates that the best variables to compress using the prediction method are L1, L2, C1, P1 and P2. All these observables can be predicted from L1. To reduce even more the size of the resulting file, for L1 we use a third order filter. For D1, D2, T1 and T2 the prediction method performs also very well and those types of observation can be predicted from one or another. Specifically, D2 can be predicted from D1 and T2 from T1. For the types of observations S1 and S2 the prediction method is not the best one, so another method will be designed for those observations in the chapter 6.

After doing many compressions with the prediction method to several different RINEX files, we observed that sometimes the pre-compression ratio obtained is much worse than that obtained using Hatanaka's method (a third order filter). In most of the cases this is due to an offset which can be eliminated easily with a first order filter. It is for this reason that another compression method has been designed. This method consists in doing a preliminary pre-compression using the prediction method, and after that applying a first order filter. We have tested this method using the same RINEX files used previously and we have obtained that in most of the cases the pre-compression ratio obtained using this method is the larger than those obtained using either the prediction method and the order filter.

6. COMPRESSION OF S1 AND S2

In the previous chapter we stated that the prediction method is not the best one to compress the types of observations S1 and S2. For this reason another compression method has been developed for only those types of observations. The values of S1 and S2 are usually much smaller than those of the rest of observations. Thus, compressing them with a first order filter would already give a good compression ratio. However, after trying this procedure, it can be seen that the values obtained are always divisible by 25, so this division could be done after the first order filter. The values obtained after the division are so small that a high pre-compression ratio can be readily obtained. Figure 6.1 shows the the results of compressing the values of S1 using the method explained in this chapter.

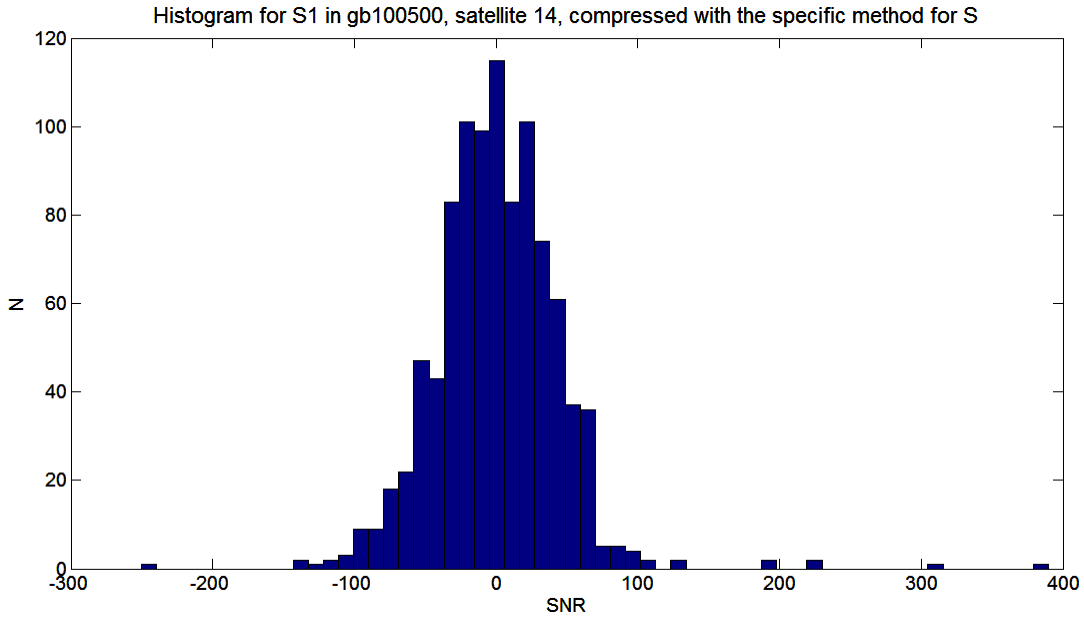

Fig. 6.1: Histogram of the pre-compressed values of S1.

On the other hand figure 6.2 shows the resulting values after applying a third order filter to the same RINEX file. As can be seen in this case using Hatanaka's method and using the method devised in this section produces almost indistinguishable results.

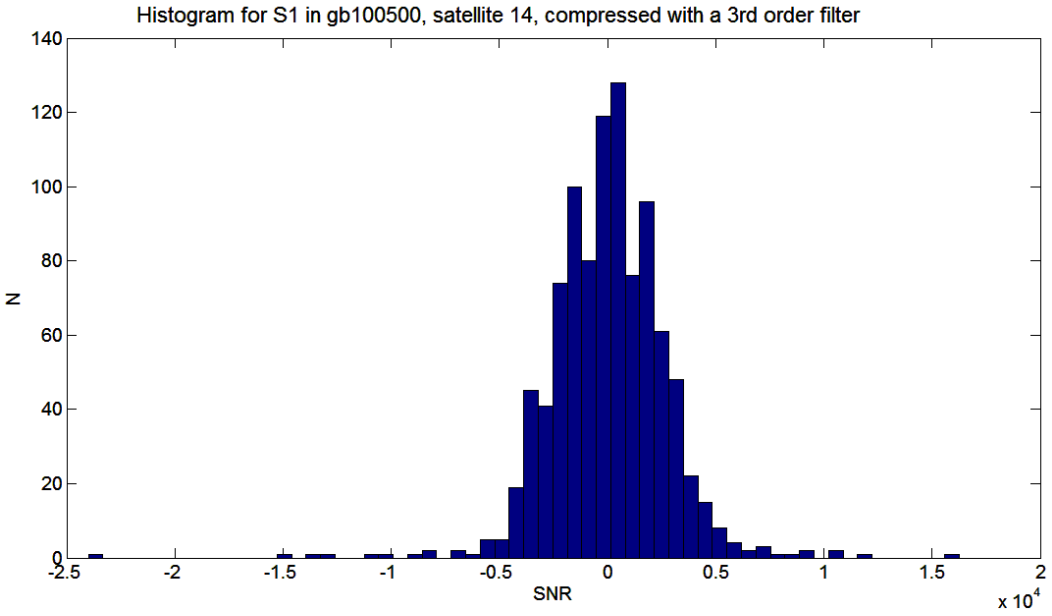

Fig. 6.2: Pre-compressed values of S1 using Hatanaka's method.

7. GLOBAL PRE-COMPRESSION RULES

7.1 Pre-compression strategy

In the previous sections four different methods for pre-compressing RINEX files have been tested and the best can be chosen depending on the scenarios. These methods are:

- Hatanaka's method (third order filter)
- Prediction methods
- Prediction method plus a first order filter
- First order filter plus a division by 25 for S1 and S2

It is obvious that the last one of these methods will only be used for S1 and S2 because it is the one which always works best for those variables (see section 6). Usually, the prediction method is the one which gives the best results, although sometimes there are values which are better compressed using one of the other two methods, either Hatanaka's method or the prediction method plus first order filter). The solution chosen in the project is to always use the best pre-compression method, selecting between the three types of compression methods every 20 lines of a RINEX file. Consequently, the pre-compression script reads the lines of the RINEX file and every 20 lines a compression of all the values is done using the three methods, and then, for the next 20 lines, the method which has given best results will be used.

7.2 Methods used for all types of observables

As explained in the chapter 1, a RINEX file can have up to 11 different types of observations. There are 4 different methods of compression, and each of those yields different results on each one of the observations. We have found that the best method for each one of the types of observations is:

- For L1 we find that is the best variable to be used for predicting some other observations. Because of this observation cannot be predicted and we always pre-compress it using Hatanaka's method.
- L2, C1, P1 and P2 usually have the best pre-compression ratios when they are predicted from L1. In some cases it turns out that using Hatanaka's method or the prediction method plus a first order filter we obtain better pre-compression ratios than when a pure prediction is done. For this reason every 20 lines the 3 methods have to be tried and the best one is chosen for the next 20 lines.
- For S1 and S2 we always use their own method, the first order filter plus a division by 25.
- T1 and D1 are always pre-compressed using Hatanaka's method because these observables are used to predict T2 and D2.
- T2 and D2 can be pre-compressed using one of the three different types of pre-compressors that are also used for L2, C1, P1 and P2, but with the

difference that once those are predicted, T2 is predicted from T1 and D2 is predicted from D1.

Sometimes there is missing data in some of the lines and that some other value cannot be pre-compressed because of this. In these cases, it seems that a good idea could be to use the data from previous lines instead of the immediately previous line. After some tests, it has been found that usually this method gives worst values than leaving the value without compression, so for those cases the value to pre-compress will be left as it was. Moreover, sometimes there are positions where there is no observational data. If one of these positions is also the position to check the method of compression the function will use the method used in the 20 previous lines.

7.3 Type of compression flags

At the end of each group of 20 lines, some flags will be included to indicate which kind of compression has been used. There are only 6 types of observations that can have the 3 different pre-compression possibilities. Thus, apart from the columns of observables, there will be another column that will contain 6 or less flags each 20 lines. Those flags belong to columns L2, C1, P1, P2 (those can be predicted from L1 or pre-compressed using Hatanaka's method), D2 and T2 (that can be predicted from D1 and T1 respectively or also pre-compressed using Hatanaka's method).

The 3 different flags that can be used are 0, 1 and 2, where:

- '0' means that Hatanaka's pre-compression method is being used.
- '1' means that the pre-compression is being done using the IEEC method (the prediction method).
- '2' means that the pre-compression is done using the prediction method plus a second order filter.

Those flags are data added to the files which also have to be pre-compressed. The method to pre-compress those flags is different from all the others. Firstly, the number composed by the different flags (which is in base 3, because it uses only the three numbers 0, 1 and 2) is converted into a number in base 10 (numbers from 0 to 9). To do it, each digit of the base 3 number must be multiplied by 3^x, where x is the index of the digit, starting with 0 on the right side. After having all the numbers multiplied, all those must be added and the result is the number in base 10. Finally, with the number obtained in this way, a differential filter (a first order filter) is used. For example:

Last flag already compressed = 35 (7.1)

Actual flag = 12012 (7.2)

Actual flag in decimal = 140 (7.3)

$$1^{st} \text{ order filter} = 140 - 35 \quad (7.4)$$

$$\text{Actual flag compressed} = 105 \quad (7.5)$$

7.4 Pre-compressing more data

Apart from pre-compressing all the types of observations some other data has also to be pre-compressed. Specifically the flags and the sub-headers which are saved in some separated files (one for the header and all the sub-headers and as many as satellites in the RINEX files for the flags).

To pre-compress the sub-headers file (the main header is not pre-compressed) and the flags files the same differential method is used. Each line read is saved into a matrix. To pre-compress a line, the current one is compared with the previous line. If a character of the line is the same that the character at the same position of the previous line, this character will be replaced by a space. In the case that there is no character in one position to avoid confusions an '&' will be introduced at that position, instead of the space. Finally if this position at the next line is once again empty, this time the position will be filed with a space and the same for the next lines while the position remains empty. Let us say that after this pre-compression, if a space is found it means that that space has the value which can be found some lines above, and if that value is an '&' it means that there is no value.

7.5 De-compression method

The de-compression method should follow the same steps previously described, but in a reverse order. We have not attempted to implement a de-compressor because the tight temporal bounds. However, future work should obviously include the design and implementation of the de-compression strategy. For doing this, the corresponding python scripts should be designed and implemented.

8. CONCLUSIONS

This report has been done only as a theoretical study, without a real application. To make sure that the methods developed were giving positive results a big number of Rinex file have been analyzed and some rough ratios were calculated during the project. Their purpose was simply to get some indications of the level of our improvements with respect to Hatanaka. Thus they helped us to decide whether we were on a good path or not, and we decided include them in the appendix 9.3. A more detailed explanation on the compression issue has been carried out in [2].

Nevertheless, a theoretical study could not be enough to prove that the methods work properly, as there could be some facts from Rinex files which we have not taken in account. That is why a second project [2] has been developed using as a base the methods exposed here, with the objective of creating a whole functional compressor for the Rinex files. Therefore the results of this project cannot be truly judged without the results of [2], because they have been developed at the same time and their results are completely dependent.

BIBLIOGRAPHY

[1] Werner, G., “RINEX-2.10 Format”, Astronomical Institute, University of Bern (2001) http://www.ngs.noaa.gov/CORS/Rinex2.html

[2] Mora, I., “GPS data compression using lossless compression algorithms”, UPC (2009).

[3] SOPAC. http://sopac.ucsd.edu/other/sopacDescription.html

APPENDIX A: A RINEX EXAMPLE

The next RINEX file is a short version of only 4 observations, taken each one every 5 seconds. It has 7 different types of observations, those are C!, P1, P2, L1, L2, D1 and D2, as indicated in the header line with the comment TYPES OF OBSRV. We remark that in the 4 observations recorded this file the information of 11 satellites is being saved, and as there are 7 types of observations, the data of only one satellite occupies two lines instead of just one.

```
     2.10           OBSERVATION DATA    G (GPS)             RINEX VERSION / TYPE
teqc  2002Mar14     Shfaqat Abbas Khan  20060904 11:13:05UTCPGM / RUN BY / DATE
Solaris 2.7|Ultra 2|cc SC5.0|=+-|*Sparc                     COMMENT
JPS2RIN 1.40        RUN BY              04-SEP-06 11:54     COMMENT
build Feb  6 2004 (c) Javad Navigation Systems              COMMENT
Use -p (profile) switch to override ANTENNA TYPE and DELTA  COMMENT
and other fields                                            COMMENT
GL210629q.jps                                               COMMENT
SE TPS 00000000                                             COMMENT
HEL1                                                        MARKER NAME
0021                                                        MARKER NUMBER
Shfaqat Ababs Khan  Danish National Space Center            OBSERVER / AGENCY
2012                JPS LEGACY          2.5 Feb,21,2005     REC # / TYPE / VERS
0205                JPSREGANT_DD_E                          ANT # / TYPE
  2011937.4648 -1584132.0150  5822389.6387                  APPROX POSITION XYZ
        0.0000        0.0000        0.0000                  ANTENNA: DELTA H/E/N
     1     1                                                WAVELENGTH FACT L1/2
     7    C1    P1    P2    L1    L2    D1    D2           # / TYPES OF OBSERV
     5.0000                                                 INTERVAL
teqc windowed: start @ 2006 Aug 19 00:00:00.000             COMMENT
teqc windowed: delta = 86400.000 sec                        COMMENT
  2006     6    29    16    23   15.0000000     GPS         TIME OF FIRST OBS
                                                            END OF HEADER
 06  8 19  0  0  0.0000000  0 11G 3G 8G11G15G18G19G22G26G27G28G29
  22447440.726    22447440.4144   22447443.0554  117962129.610 7  91918575.27044
     -3646.440       -2841.393
  22105385.529    22105384.9774   22105387.2684  116164610.564 7  90517894.91044
     -2127.458       -1657.785
  22309355.552    22309354.8574   22309356.0004  117236503.920 7  91353138.00144
      2747.722        2141.087
  24038194.794    24038193.6554   24038196.9394  126321603.970 6  98432445.02541
     -4011.235       -3125.665
  23504249.767    23504248.7654   23504250.0694  123515671.803 7  96246005.15143
     -1969.864       -1534.969
  20677668.316    20677667.0584   20677667.2244  108661919.221 8  84671653.52647
     -1638.805       -1277.000
  22586740.717    22586740.3894   22586740.1054  118694134.261 7  92488964.50644
       287.308         223.881
  23709269.786    23709269.8504   23709273.0164  124593061.402 5  97085524.16742
     -2058.402       -1603.935
  23608002.500    23608001.4494   23608003.8234  124060912.192 6  96670860.01642
     -3223.486       -2511.799
  22626417.864    22626417.2794   22626418.8424  118902647.124 7  92651444.08444
      2285.674        1781.045
  24177843.839    24177843.4494   24177847.2094  127055434.703 5  99004241.51242
     -2237.867       -1743.810
 06  8 19  0  0  5.0000000  0 11G 3G 8G11G15G18G19G22G26G27G28G29
  22450910.425    22450909.6284   22450912.4604  117980360.942 7  91932781.49544
     -3647.043       -2841.829
  22107410.584    22107410.1584   22107412.5354  116175253.479 7  90526188.08444
     -2130.077       -1659.776
  22306741.268    22306740.8384   22306741.6824  117222766.020 7  91342433.14444
      2746.926        2140.475
  24042010.680    24042010.0994   24042013.1054  126341658.426 5  98448071.89441
     -4011.164       -3125.514
  23506124.617    23506123.6704   23506124.9884  123525524.712 7  96253682.76143
     -1971.902       -1536.559
  20679228.127    20679226.9964   20679227.2134  108670117.028 9  84678041.42047
     -1640.638       -1278.403
  22586468.334    22586467.7914   22586467.6404  118692701.826 7  92487848.34244
```

```
      284.906         222.021
 23711229.547    23711229.3614   23711233.3534  124603358.691 6  97093547.96742
    -2061.029       -1605.934
 23611069.222    23611068.8694   23611071.3264  124077031.525 6  96683420.52842
    -3224.509       -2512.594
 22624243.116    22624242.9294   22624244.5734  118891220.559 7  92642540.25544
     2284.698        1780.290
 24179974.142    24179973.3184   24179976.6394  127066626.946 6  99012962.71042
    -2239.545       -1745.099
06  8 19  0  0 10.0000000  0 11G 3G 8G11G15G18G19G22G26G27G28G29
 22454381.853    22454380.8514   22454383.8134  117998601.765 7  91946995.12244
    -3648.396       -2842.932
 22109440.026    22109439.2124   22109441.6164  116185915.589 7  90534496.21444
    -2133.758       -1662.664
 22304129.275    22304128.7674   22304129.1234  117209039.269 7  91331736.98044
     2745.022        2138.956
 24045829.192    24045827.9464   24045831.4294  126361721.351 6  98463705.47341
    -4012.520       -3126.660
 23508002.659    23508001.9454   23508003.0244  123535394.915 7  96261373.80743
    -1975.083       -1539.066
 20680791.081    20680789.9824   20680790.2304  108678330.312 8  84684441.34946
    -1643.600       -1280.736
 22586198.814    22586198.7854   22586198.1144  118691288.161 7  92486746.81744
      281.450         219.291
 23713192.884    23713192.4354   23713196.4904  124613675.602 6  97101587.09242
    -2064.846       -1608.947
 23614139.120    23614138.2434   23614140.8264  124093161.846 6  96695989.61542
    -3226.441       -2514.107
 22622072.020    22622070.6204   22622072.5144  118879805.728 7  92633645.55344
     2282.406        1778.481
 24182107.715    24182106.9734   24182109.6204  127077835.399 6  99021696.57642
    -2242.974       -1747.826
06  8 19  0  0 15.0000000  0 11G 3G 8G11G15G18G19G22G26G27G28G29
 22457854.432    22457853.9224   22457856.6414  118016851.731 7  91961215.87944
    -3650.739       -2844.781
 22111472.434    22111471.5524   22111474.2414  116196596.379 7  90542818.91544
    -2137.928       -1665.938
 22301519.053    22301518.5084   22301519.3454  117195323.162 7  91321049.11544
     2742.311        2136.816
 24049648.030    24049647.4634   24049651.5654  126381791.894 6  98479344.94641
    -4014.733       -3128.370
 23509884.410    23509883.3964   23509884.7194  123545282.041 7  96269078.08243
    -1979.030       -1542.094
 20682356.797    20682355.7964   20682356.0374  108686558.608 8  84690852.96547
    -1647.038       -1283.419
 22585933.368    22585933.1804   22585933.1094  118689892.673 7  92485659.44144
      277.448         216.186
 23715159.362    23715159.7464   23715163.5454  124624011.524 6  97109641.05942
    -2069.023       -1612.239
 23617210.540    23617209.3584   23617212.5044  124109302.656 6  96708566.87142
    -3229.068       -2516.168
 22619901.122    22619900.4784   22619902.4934  118868402.199 7  92624759.70544
     2279.665        1776.360
 24184243.494    24184242.8964   24184245.3644  127089059.369 6  99030442.55542
    -2246.057       -1750.206
```

Tables A1 and A2 are extracted from the RINEX definition document [4] and shows the organization of the data inside the header of a RINEX file and also the organization inside the data.

TABLE A1
GPS OBSERVATION DATA FILE - HEADER SECTION DESCRIPTION

	HEADER LABEL (Columns 61-80)	DESCRIPTION	FORMAT
	RINEX VERSION / TYPE	- Format version (2.10) - File type ('O' for Observation Data) - Satellite System: blank or 'G': GPS 'R': GLONASS 'S': Geostationary signal payload 'T': NNSS Transit 'M': Mixed	F9.2,11X, A1,19X, A1,19X
	PGM / RUN BY / DATE	- Name of program creating current file - Name of agency creating current file - Date of file creation	A20, A20, A20
*	COMMENT	Comment line(s)	A60
	MARKER NAME	Name of antenna marker	A60
*	MARKER NUMBER	Number of antenna marker	A20
	OBSERVER / AGENCY	Name of observer / agency	A20,A40
	REC # / TYPE / VERS	Receiver number, type, and version (Version: e.g. Internal Software Version)	3A20
	ANT # / TYPE	Antenna number and type	2A20
	APPROX POSITION XYZ	Approximate marker position (WGS84)	3F14.4
	ANTENNA: DELTA H/E/N	- Antenna height: Height of bottom surface of antenna above marker - Eccentricities of antenna center relative to marker to the east and north (all units in meters)	3F14.4
	WAVELENGTH FACT L1/2	- Default wavelength factors for L1 and L2 1: Full cycle ambiguities 2: Half cycle ambiguities (squaring) 0 (in L2): Single frequency instrument - zero or blank The default wavelength factor line is required and must preceed satellite-specific lines.	2I6, I6
*	WAVELENGTH FACT L1/2	- Wavelength factors for L1 and L2 1: Full cycle ambiguities 2: Half cycle ambiguities (squaring) 0 (in L2): Single frequency instrument - Number of satellites to follow in list for which these factors are valid. - List of PRNs (satellite numbers with system identifier) These opional satellite specific lines may follow, if they identify a state different from the default values. Repeat record if necessary.	2I6, I6, 7(3X,A1,I2)
	# / TYPES OF OBSERV	- Number of different observation types stored in the file	I6,

	- Observation types	9(4X,A2)	
	If more than 9 observation types: Use continuation line(s)	6X,9(4X,A2)	
	The following observation types are defined in RINEX Version 2.10: L1, L2: Phase measurements on L1 and L2 C1 : Pseudorange using C/A-Code on L1 P1, P2: Pseudorange using P-Code on L1,L2 D1, D2: Doppler frequency on L1 and L2 T1, T2: Transit Integrated Doppler on 150 (T1) and 400 MHz (T2) S1, S2: Raw signal strengths or SNR values as given by the receiver for the L1,L2 phase observations Observations collected under Antispoofing are converted to "L2" or "P2" and flagged with bit 2 of loss of lock indicator (see Table A2). Units : Phase : full cycles Pseudorange : meters Doppler : Hz Transit : cycles SNR etc : receiver-dependent The sequence of the types in this record has to correspond to the sequence of the observations in the observation records		
*	INTERVAL	Observation interval in seconds	F10.3	*
	TIME OF FIRST OBS	- Time of first observation record (4-digit-year, month,day,hour,min,sec) - Time system: GPS (=GPS time system) GLO (=UTC time system) Compulsory in mixed GPS/GLONASS files Defaults: GPS for pure GPS files GLO for pure GLONASS files	5I6,F13.7, 5X,A3	
*	TIME OF LAST OBS	- Time of last observation record (4-digit-year, month,day,hour,min,sec) - Time system: Same value as in TIME OF FIRST OBS record	5I6,F13.7, 5X,A3	*
*	RCV CLOCK OFFS APPL	Epoch, code, and phase are corrected by applying the realtime-derived receiver clock offset: 1=yes, 0=no; default: 0=no Record required if clock offsets are reported in the EPOCH/SAT records	I6	*
*	LEAP SECONDS	Number of leap seconds since 6-Jan-1980 Recommended for mixed GPS/GLONASS files	I6	*
*	# OF SATELLITES	Number of satellites, for which observations are stored in the file	I6	*
*	PRN / # OF OBS	PRN (sat.number), number of observations for each observation type indicated in the "# / TYPES OF OBSERV" - record. If more than 9 observation types: Use continuation line(s) This record is (these records are) repeated for each satellite present in the data file	3X,A1,I2,9I6 6X,9I6	*
	END OF HEADER	Last record in the header section.	60X	

Records marked with * are optional

TABLE A2

GPS OBSERVATION DATA FILE - DATA RECORD DESCRIPTION

OBS. RECORD	DESCRIPTION	FORMAT
EPOCH/SAT or EVENT FLAG	- Epoch : - year (2 digits, padded with 0 if necessary) - month,day,hour,min, - sec	1X,I2.2, 4(1X,I2), F11.7,
	- Epoch flag 0: OK 1: power failure between previous and current epoch >1: Event flag	2X,I1,
	- Number of satellites in current epoch	I3,
	- List of PRNs (sat.numbers with system identifier, see 5.1) in current epoch	12(A1,I2),
	- receiver clock offset (seconds, optional)	F12.9
	If more than 12 satellites: Use continuation line(s)	32X, 12(A1,I2)
	If epoch flag 2-5:	
	- Event flag: 2: start moving antenna 3: new site occupation (end of kinem. data) (at least MARKER NAME record follows) 4: header information follows 5: external event (epoch is significant, same time frame as observation time tags)	[2X,I1,]
	- "Number of satellites" contains number of special records to follow. Maximum number of records: 999	[I3]
	- For events without significant epoch the epoch fields can be left blank	
	If epoch flag = 6: 6: cycle slip records follow to optionally report detected and repaired cycle slips (same format as OBSERVATIONS records; slip instead of observation; LLI and signal strength blank or zero)	
OBSERVATIONS	- Observation - LLI - Signal strength (rep. within record for each obs.type (same seq as given in header))	m(F14.3, I1, I1)
	If more than 5 observation types (=80 char): continue observations in next record.	
	This record is (these records are) repeated for each satellite given in EPOCH/SAT - record.	
	Observations: Phase : Units in whole cycles of carrier Code : Units in meters Missing observations are written as 0.0 or blanks.	
	Phase values overflowing the fixed format F14.3 have to be clipped into the valid interval (e.g. add or subtract 10**9), set LLI indicator.	
	Loss of lock indicator (LLI). Range: 0-7 0 or blank: OK or not known Bit 0 set : Lost lock between previous and current observation: cycle slip possible Bit 1 set : Opposite wavelength factor to the one defined for the satellite by a previous WAVELENGTH FACT L1/2 line. Valid for the current epoch only. Bit 2 set : Observation under Antispoofing	

		(may suffer from increased noise)	
		Bits 0 and 1 for phase only.	
		Signal strength projected into interval 1-9:	
		1: minimum possible signal strength	
		5: threshold for good S/N ratio	
		9: maximum possible signal strength	
		0 or blank: not known, don't care	

APPENDIX B: PYTHON CODE

```
#! /usr/bin/env python

import re
import sys
import os
import string
usage = "\n The " + sys.argv[0][2:-3] + " program converts from the original RINEX Files format to a basic Raw ASCII \n \
with one single file per satellite observed\n\n \
USAGE: " + sys.argv[0][2:] + " [RinexOriginalFile]\n";

#global variables
next_line_id = False;
id_field = "";
sat_count = 0;
sat_list = []; #list of satellites
sat_tupla = []; #list of satellite with file name
sat_tupla_flags = []; #list of satellite with file name for the flags
id_split = []; #list of id of the satellites for the current block
types_of_observ = " #number of different observation types stored in the file
observ_stored = [] #names in the rinex order of the diferent types of observ stored in the file
last_line = " #it contains the values of the line which is before the one that is being read
read = 1 #with more than 5 types of obs, 1 while reading the first line, 2 while the second

#find the order of observations in the file and store it in the vector observ_stored
def find_types_of_observ(line):
        global observ_stored, types_of_observ

        line = line.replace('\r','')
        line = line.replace('\n','')
        types_of_observ = int(line[0:6])
        index = 10
        for num in range(types_of_observ):
                observ_stored.append(line[index:(index+2)])
                index += 6

# we take one of the lines of the Rinex file and we treat it
def GPS_reformatting(line, subhfile):
        global id_field, next_line_id, sat_count, sat_list, id_split, last_line_counter, types_of_observ, observ_stored, last_line, read

        #we remove all the \n and \r from the lines
```

```
        line = line.replace('\r','')
        line = line.replace('\n','')

        split_line = re.split(" ",line); #we split the line into words, we will have the words separated in the vector split_line

        #still info to be adde to the id_split variable; There is more than one line for the "block" header
        if(next_line_id == True):
                splitted = line.strip(); #remove all the ' ' at the beginning and at the end
                id_split = re.split(" ",splitted); #now only useful strings
                #we delete the last 12 characters of the subheader because those ara the receiver clock offset
                split_line = re.split(" ",line[:68])
                GFind = False;
                for word in id_split:
                        if(re.search("G",word)): #we search for the 'G' character that indicates the id satellite field
                                GFind = True;
                        if(GFind == True):
                                id_field += word;
                next_line_id = False; #to the initial value

                #writing the subheader in its corresponding file
                subhfile.write(line+"\n");

        #new subheader line
        elif(line.find("G") >= 0): #contains a date -> "block" header
                sat_count = 0; #new data block
                id_field = ''

                #we delete the last 12 characters of the subheader because those are the receiver clock offset
                split_line = re.split(" ",line[:68])
                GFind = False;
                for word in split_line: #we are going to save the id of the different satellites
                        if(re.search("G",word)): #we search for the 'G' character that indicates the id satellite field
                                GFind = True;
                        if(GFind == True):
                                id_field += word;

                # reliance test
                id_field = id_field.rstrip('\r');
                id_field = id_field.replace('\r','')
```

```
                id_field = id_field.replace('\n','')

                id_split = re.split("G",id_field);
                if(int(id_split[0]) + 1 != len(id_split)):
                        next_line_id = True; #we still need some data to complete the id
field

                #writing the subheader in its corresponding file
                subhfile.write(line+"\n");

        #data line
        else:
                id_field = id_field.rstrip('\r');
                id_field = id_field.replace('\r','')
                id_field = id_field.replace('\n','')
                id_split = re.split("G", id_field);
                if(int(id_split[0]) + 1 != len(id_split)):
                        print "ERROR: We have found one block with no correct satellite
id", id_split;
                        exit(-1);
                for sats in id_split[1:]: #we do not use the first sample, as it is the total
number of sats
                        if(sat_list.count(sats) == 0): #this satellite has never appeared
before
                                sat_list.append(sats); #new entry in the list
                                output_file_name = sys.argv[1][0:-4] + "_" + sats +
sys.argv[1][-4:] + ".obs";
                                output_flag_file_name = sys.argv[1][0:-4] + "_" + sats +
sys.argv[1][-4:] + ".flg";
                                outfile = open(output_file_name,'w'); #creation of the
output file
                                outfile_flags = open(output_flag_file_name,'w')#creation
of the output file with the flags data
                                sat_tupla.append( [sats, output_file_name, outfile] )
                                sat_tupla_flags.append( [sats, output_flag_file_name,
outfile_flags] )

                if(types_of_observ <= 5):
                        line = reorder(line, types_of_observ, observ_stored)
                        line = clean_line(line)
                        process_line(line)

                else:
                        if(read == 1):
                                read = 2
                        else:
```

```
                last_line = last_line.replace('\r','')
                last_line = last_line.replace('\n','')

                #we need to have 80 characters at the line and last line, if
necessary, we add spaces.
                if (len(line) < 80):
                        rep = 80 - len(line)
                        for num in range(rep):
                                line = line + ' '
                line = line.replace('\r','')
                line = line.replace('\n','')

                if (len(last_line) < 80):
                        rep = 80 - len(last_line)
                        for num in range(rep):
                                last_line = last_line + ' '
                last_line = last_line.replace('\r','')
                last_line = last_line.replace('\n','')

                line = last_line + line
                line = line.replace('\r','')
                line = line.replace('\n','')

                line = reorder(line, types_of_observ, observ_stored)
                line = clean_line(line)
                process_line(line)

                read = 1

#reordering of the observ data from the file to the std order: L1 L2 C1 P1 P2 D1 D2 S1
S2 T1 T2
def reorder(line, types_of_observ, observ_stored):

        reordered_line = ''
        L1 = '          & '
        L2 = '          & '
        C1 = '          & '
        P1 = '          & '
        P2 = '          & '
        D1 = '          & '
        D2 = '          & '
        S1 = '          & '
        S2 = '          & '
        T1 = '          & '
        T2 = '          & '
```

```
index1 = 0
index2 = 0
for num in range(types_of_observ):
        if(observ_stored[index2]=='L1'):
                L1 = line[index1:(index1+16)]
                if(L1 == ''):
                        L1 = '           &  '
                if(len(L1)<16):
                        for ind in range(16-len(L1)):
                                L1 = L1 + ' '
                if(L1.find('              ') >= 0):
                        L1 = '           &  '

        elif(observ_stored[index2]=='L2'):
                L2 = line[index1:(index1+16)]
                if(L2 == ''):
                        L2 = '           &  '
                if(len(L2)<16):
                        for ind in range(16-len(L2)):
                                L2 = L2 + ' '
                if(L2.find('              ') >= 0):
                        L2 = '           &  '

        elif(observ_stored[index2]=='C1'):
                C1 = line[index1:(index1+16)]
                if(C1 == ''):
                        C1 = '           &  '
                if(len(C1)<16):
                        for ind in range(16-len(C1)):
                                C1 = C1 + ' '
                if(C1.find('              ') >= 0):
                        C1 = '           &  '

        elif(observ_stored[index2]=='P1'):
                P1 = line[index1:(index1+16)]
                if(P1 == ''):
                        P1 = '           &  '
                if(len(P1)<16):
                        for ind in range(16-len(P1)):
                                P1 = P1 + ' '
                if(P1.find('              ') >= 0):
                        P1 = '           &  '

        elif(observ_stored[index2]=='P2'):
                P2 = line[index1:(index1+16)]
                if(P2 == ''):
```

```
                P2 = '          &  '
        if(len(P2)<16):
                for ind in range(16-len(P2)):
                        P2 = P2 + ' '
        if(P2.find('            ') >= 0):
                P2 = '          &  '

elif(observ_stored[index2]=='D1'):
        D1 = line[index1:(index1+16)]
        if(D1 == ''):
                D1 = '          &  '
        if(len(D1)<16):
                for ind in range(16-len(D1)):
                        D1 = D1 + ' '
        if(D1.find('            ') >= 0):
                D1 = '          &  '

elif(observ_stored[index2]=='D2'):
        D2 = line[index1:(index1+16)]
        if(D2 == ''):
                D2 = '          &  '
        if(len(D2)<16):
                for ind in range(16-len(D2)):
                        D2 = D2 + ' '
        if(D2.find('            ') >= 0):
                D2 = '          &  '

elif(observ_stored[index2]=='S1'):
        S1 = line[index1:(index1+16)]
        if(S1 == ''):
                S1 = '          &  '
        if(len(S1)<16):
                for ind in range(16-len(S1)):
                        S1 = S1 + ' '
        if(S1.find('            ') >= 0):
                S1 = '          &  '

elif(observ_stored[index2]=='S2'):
        S2 = line[index1:(index1+16)]
        if(S2 == ''):
                S2 = '          &  '
        if(len(S2)<16):
                for ind in range(16-len(S2)):
                        S2 = S2 + ' '
        if(S2.find('            ') >= 0):
                S2 = '          &  '
```

```
                elif(observ_stored[index2]=='T1'):
                        T1 = line[index1:(index1+16)]
                        if(T1 == ''):
                                T1 = '          &  '
                        if(len(T1)<16):
                                for ind in range(16-len(T1)):
                                        T1 = T1 + ' '
                        if(T1.find('              ') >= 0):
                                T1 = '          &  '

                elif(observ_stored[index2]=='T2'):
                        T2 = line[index1:(index1+16)]
                        if(T2 == ''):
                                T2 = '          &  '
                        if(len(T2)<16):
                                for ind in range(16-len(T2)):
                                        T2 = T2 + ' '
                        if(T2.find('              ') >= 0):
                                T2 = '          &  '

                index1 += 16
                index2 += 1

        reordered_line = L1 + L2 + C1 + P1 + P2 + D1 + D2 + S1 + S2 + T1 + T2
        return reordered_line

#cleaning of the observational lines --> no comas and flags
def clean_line(line):
        o1 = line[0:10] + line[11:14]
        o1f = line[14:16]
        o2 = line[16:26] + line[27:30]
        o2f = line[30:32]
        o3 = line[32:42] + line[43:46]
        o3f = line[46:48]
        o4 = line[48:58] + line[59:62]
        o4f = line[62:64]
        o5 = line[64:74] + line[75:78]
        o5f = line[78:80]
        o6 = line[80:90] + line[91:94]
        o6f = line[94:96]
        o7 = line[96:106] + line[107:110]
        o7f = line[110:112]
        o8 = line[112:122] + line[123:126]
        o8f = line[126:128]
        o9 = line[128:138] + line[139:142]
```

```
        o9f = line[142:144]
        o10 = line[144:154] + line[155:158]
        o10f = line[158:160]
        o11 = line[160:170] + line[171:174]
        o11f = line[174:176]

        line = o1 + o2 + o3 + o4 + o5 + o6 + o7 + o8 + o9 + o10 + o11 + ' ' + o1f
+ o2f + o3f + o4f + o5f + o6f + o7f + o8f + o9f + o10f + o11f
        return line;

#processing of the observational lines
def process_line(line):
        global id_field, sat_list, sat_count, id_split, types_of_observ
        sat_count += 1;

        if(sat_count > int(id_split[0])):
                print "CAUTION: Number of processed satellites", sat_count, "could be
greater than the expected", id_split[0];
                return;

        sat_tupla[sat_list.index(id_split[sat_count])][2].write(str(line[:143]) + "\n"); #we
write the data line into the rigth file
        sat_tupla_flags[sat_list.index(id_split[sat_count])][2].write(str(line[143:]) + "\n")
#we write the flags line into the rigth file

def main():
        global types_of_observ, observ_stored, last_line

        # arguments load
        if len(sys.argv) > 2:
                print usage;
                exit();

        # we open the file
        file_name = sys.argv[1];
        file = open(file_name); #open the file
        file_lines = file.readlines(); #split the files into separate lines

        header = True; #are we still inside the header?

        #creation of the file which will contain all the subheaders
        subheaders_file_name = sys.argv[1] + ".sbh";
        subhfile = open(subheaders_file_name,'w'); #creation of the output file

        for line in file_lines: #loop over each of the lines
                #we find the number or different observation types
```

```
                if(line.find("TYPES OF OBSERV") >= 0):
                        find_types_of_observ(line)

                #we are already out of the header
                if(header == False): #we are already out of the header
                        #the line is not a coment nor is empty
                        if(line.find("COMMENT") < 0 and line.find("                    ")
< 0):
                                GPS_reformatting(line, subhfile)
                        #It is a observations header because there are G but it is not a
comment
                        elif(line.find("G") >= 0 and line.find("COMMENT") < 0):
                                GPS_reformatting(line, subhfile)
                        #it is a observations line, because there are also long empty
spaces, there is some decimal dot
                        elif(line.find(".") >= 0 and line.find("          ") >= 0):
                                GPS_reformatting(line, subhfile)

                        elif(line.find("R") >= 0 and line.find("COMMENT") < 0):
                                print 'Error, this rinex file contains GLONASS DATA'
                                return

                        elif(line.find("COMMENT") > 0):
                                subhfile.write(line)
                        else:
                                subhfile.write(line)

                #we write all the header in the first part of the subheaders file
                if(header == True):
                        subhfile.write(line);

                #we find the end of the header
                if(line.find("END OF HEADER") >= 0):
                        header = False; #false

                last_line = line;

main();
```

APPENDIX C: DATA COMPARISON

This appendix includes 11 different tables. Each of these tables contains values that give an idea of which the best method of compression is. The values in the cells are the division between the mean value of all the pre-compressed Hatanaka values and the mean value of all the pre-compressed prediction method values. Thus, when the result is larger than 1 it means that the prediction method gives better results of pre-compression. For the sake of clarity we have also colored different cells. The colors represent which method is better. The red cells mean that Hatanaka's method performs better than our own method, whereas blue and green cells mean that the prediction method overcomes Hatanaka's method. Each table contains data from 12 rows, each from different satellites from different RINEX files, and each column has the division results for one type of observation. The difference between each one of the tables is that the predictions have been done from one specific type of observation.

Table C.1: Predictions for L1

	L2 < L1	C1 < L1	P1 < L1	P2 < L1	D1 < L1	D2 < L1	T1< L1	T2 < L1	S1 < L1	S2 < L1
almr0510_18.rawc	575792	137128	NaN	95470	NaN	NaN	NaN	NaN	NaN	NaN
DPCA001K_21.rawc	NaN	314698	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
esqu0630_16.rawc	89	0	0	0	NaN	NaN	3	NaN	1	NaN
gb100500_14.rawc	NaN	5	NaN	NaN	NaN	NaN	3	NaN	NaN	NaN
gc082360_12.rawc	NaN	7	NaN	NaN	NaN	NaN	3	NaN	NaN	NaN
hel12300_27.rawc	1360155	18357	22398	19885	3	3	NaN	NaN	NaN	NaN
kfrc031b_14.rawc	1922	3	NaN	3	NaN	NaN	NaN	NaN	NaN	NaN
mrkt041j_06.rawc	38846351	6614379	NaN	6299722	NaN	NaN	NaN	NaN	NaN	NaN
nun12250_21.raw	8272	1	NaN	1	NaN	NaN	3	3	NaN	NaN
prdu029u_27.rawc	9755751	3115086	NaN	2287748	NaN	NaN	NaN	NaN	NaN	NaN
SEVI009S_04.rawc	NaN	133103	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ucad0360_31.rawc	0	0	NaN	0	NaN	NaN	NaN	NaN	NaN	NaN

Table C.2: Predictions for L2

	L1 < L2	C1 < L2	P1 < L2	P2 < L2	D1 < L2	D2 < L2	T1< L2	T2 < L2	S1 < L2	S2 < L2
almr0510_18.rawc	575791	117300	NaN	87222	NaN	NaN	NaN	NaN	NaN	NaN
DPCA001K_21.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
esqu0630_16.rawc	86	0	0	0	NaN	NaN	2	NaN	1	NaN
gb100500_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gc082360_12.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
hel12300_27.rawc	1360155	18334	22334	19865	3	3	NaN	NaN	NaN	NaN
kfrc031b_14.rawc	1921	3	NaN	3	NaN	NaN	NaN	NaN	NaN	NaN
mrkt041j_06.rawc	38846352	5884398	NaN	5801609	NaN	NaN	NaN	NaN	NaN	NaN
nun12250_21.raw	8275	1	NaN	1	NaN	NaN	3	3	NaN	NaN
prdu029u_27.rawc	9755750	2369953	NaN	1868902	NaN	NaN	NaN	NaN	NaN	NaN
SEVI009S_04.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ucad0360_31.rawc	2	7	NaN	7	NaN	NaN	NaN	NaN	NaN	NaN

Table C.3: Predictions for C1

	L1 < C1	L2 < C1	P1 < C1	P2 < C1	D1 < C1	D2 < C1	T1< C1	T2 < C1	S1 < C1	S2 < C1
almr0510_18.rawc	137128	117300	NaN	81144	NaN	NaN	NaN	NaN	NaN	NaN
DPCA001K_21.rawc	314697	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
esqu0630_16.rawc	1	1	20041	20176	NaN	NaN	3	NaN	2	NaN
gb100500_14.rawc	8	NaN	NaN	NaN	NaN	NaN	3	NaN	NaN	NaN
gc082360_12.rawc	11	NaN	NaN	NaN	NaN	NaN	3	NaN	NaN	NaN
hel12300_27.rawc	18355	18332	15164	13238	3	3	NaN	NaN	NaN	NaN
kfrc031b_14.rawc	3	3	NaN	102400	NaN	NaN	NaN	NaN	NaN	NaN
mrkt041j_06.rawc	6614379	5884398	NaN	5479533	NaN	NaN	NaN	NaN	NaN	NaN
nun12250_21.raw	4	4	NaN	5978	NaN	NaN	3	3	NaN	NaN
prdu029u_27.rawc	3115086	2369952	NaN	6424553	NaN	NaN	NaN	NaN	NaN	NaN
SEVI009S_04.rawc	133101	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ucad0360_31.rawc	0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table C.4: Predictions for P1

	L1 < P1	L2 < P1	C1 < P1	P2 < P1	D1 < P1	D2 < P1	T1< P1	T2 < P1	S1 < P1	S2 < P1
almr0510_18.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DPCA001K_21.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
esqu0630_16.rawc	1	1	20039	14847	NaN	NaN	3	NaN	2	NaN
gb100500_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gc082360_12.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
hel12300_27.rawc	22396	22333	15164	8	3	3	NaN	NaN	NaN	NaN
kfrc031b_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mrkt041j_06.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nun12250_21.raw	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
prdu029u_27.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SEVI009S_04.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ucad0360_31.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table C.5: Predictions for P2

	L1 < P2	L2 < P2	C1 < P2	P1 < P2	D1 < P2	D2 < P2	T1< P2	T2 < P2	S1 < P2	S2 < P2
almr0510_18.rawc	95469	87221	81144	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DPCA001K_21.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
esqu0630_16.rawc	1	1	20173	14847	NaN	NaN	3	NaN	2	NaN
gb100500_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gc082360_12.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
hel12300_27.rawc	19883	19863	13238	14965	3	3	NaN	NaN	NaN	NaN
kfrc031b_14.rawc	3	3	102400	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mrkt041j_06.rawc	6299723	5801609	5479533	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nun12250_21.raw	4	4	5979	NaN	NaN	NaN	3	3	NaN	NaN
prdu029u_27.rawc	2287747	1868901	6424552	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SEVI009S_04.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ucad0360_31.rawc	0	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table C.6: Predictions for D1

	L1 < D1	L2 < D1	C1 < D1	P1 < D1	P2 < D1	D2 < D1	T1< D1	T2 < D1	S1 < D1	S2 < D1
almr0510_18.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DPCA001K_21.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
esqu0630_16.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gb100500_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gc082360_12.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
hel12300_27.rawc	0,0351	0,0351	0,0351	0,0351	0,0351	340,34	NaN	NaN	NaN	NaN
kfrc031b_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mrkt041j_06.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nun12250_21.raw	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
prdu029u_27.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SEVI009S_04.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ucad0360_31.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table C.7: Predictions for D2

	L1 < D2	L2 < D2	C1 < D2	P1 < D2	P2 < D2	D1 < D2	T1< D2	T2 < D2	S1 < D2	S2 < D2
almr0510_18.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DPCA001K_21.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
esqu0630_16.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gb100500_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gc082360_12.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
hel12300_27.rawc	0	0	0	0	0	340	NaN	NaN	NaN	NaN
kfrc031b_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mrkt041j_06.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nun12250_21.raw	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
prdu029u_27.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SEVI009S_04.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ucad0360_31.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table C.8: Predictions for S1

	L1 < S1	L2 < S1	C1 < S1	P1 < S1	P2 < S1	D1 < S1	D2< S1	T1 < S1	T2 < S1	S2 < S1
almr0510_18.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DPCA001K_21.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
esqu0630_16.rawc	1	1	0	0	0	NaN	NaN	3	NaN	NaN
gb100500_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gc082360_12.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
hel12300_27.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
kfrc031b_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mrkt041j_06.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nun12250_21.raw	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
prdu029u_27.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SEVI009S_04.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ucad0360_31.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table C.9: Predictions for S2

	L1 < S2	L2 < S2	C1 < S2	P1 < S2	P2 < S2	D1 < S2	D2< S2	T1 < S2	T2 < S2	S1 < S2
almr0510_18.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DPCA001K_21.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
esqu0630_16.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gb100500_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gc082360_12.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
hel12300_27.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
kfrc031b_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mrkt041j_06.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nun12250_21.raw	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
prdu029u_27.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SEVI009S_04.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ucad0360_31.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table 9.10: Predictions for T1

	L1 < T1	L2 < T1	C1 < T1	P1 < T1	P2 < T1	D1 < T1	D2< T1	T2 < T1	S1 < T1	S2 < T1
almr0510_18.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DPCA001K_21.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
esqu0630_16.rawc	0	0	0	0	0	NaN	NaN	NaN	1	NaN
gb100500_14.rawc	0	NaN	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gc082360_12.rawc	0	NaN	0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
hel12300_27.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
kfrc031b_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mrkt041j_06.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nun12250_21.raw	0	0	0	NaN	0	NaN	NaN	2	NaN	NaN
prdu029u_27.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SEVI009S_04.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ucad0360_31.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Table C.11: Predictions for T2

	L1 < T2	L2 < T2	C1 < T2	P1 < T2	P2 < T2	D1 < T2	D2< T2	T1 < T2	S1 < T2	S2 < T2
almr0510_18.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DPCA001K_21.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
esqu0630_16.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gb100500_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
gc082360_12.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
hel12300_27.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
kfrc031b_14.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mrkt041j_06.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
nun12250_21.raw	0	0	0	NaN	0	NaN	NaN	1	NaN	NaN
prdu029u_27.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
SEVI009S_04.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
ucad0360_31.rawc	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

APPENDIX D: MATLAB SCRIPTS

In this Appendix we list all the Matlab scripts written for this project. If a full code line does not fit in a document line, it will continue in the following line.

Rinex_hist(file, color, orden, divisiones, psoph, type, adim, lim1, lim2).m

```
function P3=rinex_hist_v5(file,color,orden, divisiones, psoph, type,adim,lim1,lim2)
%---Plots a histogram or graphic for a satellite of a rinex file following the next---
%-->rinex_hist_v5(file,color,orden, divisiones, psoph, type,adim,lim1,lim2)
%
%WARNING: only works with satellite files without flags or commas. It is
%supposed that the first column is the pseudorange and the second one the phase

%Example:
%--> rinex_hist_v5('mlaga0390_13.raw','b',0,100,0,0,1,-100,100)
%
%Variables:
%  - file: rinex file to read ' '
%  - color: between ' ' put the color for the graphic
%  - orden: order of the filter, 1, 2, 3 o 0 without filter
%  - divisiones: number of divisions to show in the histogram
%  - psoph: 0 for P1, 1 for P2, 2 for L1, 3 for L2, 4 for C1, 5 for D1, 6 for D2, 7 for
L1/C1 o 8 for C1/L1,
%  - type: 0 for line or another for histogram
%  - adim: 1 if it is wanted to have the graphic in adimensian Y axis
%  - lim1 y lim2: both extrems of the histogram (0 for automatic)(useful
%  for 2nd and 3rd order filter)

format long
[L1,L2,C1,P1,P2,D1,D2]= textread(file,'%f%f%f%f%f%f%f%*[^\n]');

j=size(C1); %=size(L1); etc...
j=j(1);

%taking the data that the user wants
vector=zeros(1,j);
for i=1:j,
   if psoph==0
      vector(i)=P1(i);
   elseif psoph==1
      vector(i)=P2(i);
   elseif psoph==2
      vector(i)=L1(i);
   elseif psoph==3
      vector(i)=L2(i);
   elseif psoph==4
      vector(i)=C1(i);
   elseif psoph==5
      vector(i)=D1(i);
```

```
   elseif psoph==6
      vector(i)=D2(i);
   elseif psoph==7
      vector(i)=L1(i)/C1(i);
   elseif psoph==8
      vector(i)=C1(i)/L1(i);
   end
end

%1st order
P1=zeros(1,j);
for i=2:j,
   P1(i)=(vector(i)-vector(i-1));
end

%2nd order
P2=zeros(1,j);
for i=3:j,
   P2(i)=(vector(i)-2*vector(i-1)+vector(i-2));
end

%3rd order
P3=zeros(1,j);
for i=4:j,
   P3(i)=(vector(i)-3*vector(i-1)+3*vector(i-2)-vector(i-3));
end

switch orden
   case 0,
      if type==0
         [x,y]=hist((vector),divisiones);

         if adim==1
            %non-dimensional N
            xM=max(x);
            for i=1:divisiones,
               x(i)=x(i)/xM;
            end
         end

         plot(y,x,color);
      else
         if ((lim1==0)&&(lim2==0))
            hist((vector),divisiones);
         else
            n=histc((vector),lim1:lim2);
            bar(lim1:lim2,n,'histc');
         end
      end
      title('histogram for no filter')
   case 1,
      if type==0
         [x,y]=hist((P1),divisiones);
```

```
        if adim==1
            %non-dimensional N
            xM=max(x);
            for i=1:divisiones,
                x(i)=x(i)/xM;
            end
        end

        plot(y,x,color);
    else
        if ((lim1==0)&&(lim2==0))
            hist((P1),divisiones);
        else
            n=histc((P1),lim1:lim2);
            bar(lim1:lim2,n,'histc');
        end
    end
    title('1st order histogram')
case 2
    if type==0
        [x,y]=hist((P2),divisiones);

        if adim==1
            %N non-dimensional
            xM=max(x);
            for i=1:divisiones,
                x(i)=x(i)/xM;
            end
        end
        plot(y,x,color);
    else
        if ((lim1==0)&&(lim2==0))
            hist((P2),divisiones);
        else
            n=histc((P2),lim1:lim2);
            bar(lim1:lim2,n,'histc');
        end
    end
    title('Histogram for 2nd order')
case 3
    if type==0
        [x,y]=hist((P3),divisiones);


        if adim==1
            %non-dimensional N
            xM=max(x);
            for i=1:divisiones,
                x(i)=x(i)/xM;
            end
        end
        plot(y,x,color);
```

```
        else
            if ((lim1==0)&&(lim2==0))
                hist((P3),divisiones);
            else
                n=histc((P3),lim1:lim2);
                bar(lim1:lim2,n,'histc');
            end
        end
        title('Histogram for 3rd order')
end

ylabel('N')
```

Output(file.m)

```
function output(file)
%this script takes a file and compresses it using both hatanaka method and
%prediction method, and giving as a result 2 new files containing each one
%the values of one of the two compressions.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%creation of the output file that contains the data obtained into our version%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
name=file(1:(length(file)-5));
name(length(name):(length(name)+8))='_ieec.txt';
ieec = fopen(name,'w');

fprintf(ieec, '    ************************************************    \n')
fprintf(ieec, '    *** Compressed using IEEC Compression System ***    \n')
fprintf(ieec, '    ************************************************    \n\n')

[L1,L2,C1,P1,P2,D1,D2,T1,T2,S1,S2]=
textread(file,'%f%f%f%f%f%f%f%f%f%f%*[^\n]');

j=size(L1); %= j=size(L2); =...
j=j(1);

%3th order from L1
L1_3=zeros(1,j);
if L1(1)~=0
   L1_3(1) = L1(1);
   L1_3(2) = L1(2) - L1(1);
   L1_3(3) = L1(3) - 2*L1(2) + L1(1);
   for i = 4:j,
         L1_3(i) = (L1(i) - 3*L1(i-1) + 3*L1(i-2) - L1(i-3));
   end
end

%L2 prediction
errorL2=zeros(1,j);
```

```
if L2(1)~=0
   L2predic=zeros(1,j);
   k=zeros(1,j);
   k(1)=L1(1)/L2(1);
   L2predic(1)=L1(1)/k(1); %-->miscalculation(1)=0;
   for i=2:j,
      L2predic(i)=L1(i)/k(i-1);
      k(i)=L1(i)/L2(i);
   end

   for i=1:j,
      errorL2(i)=L2(i)-L2predic(i);
   end
end

%C1 prediction
errorC1=zeros(1,j);
if C1(1)~=0
   k=zeros(1,j);
   C1predic=zeros(1,j);
   k(1)=L1(1)/C1(1);
   C1predic(1)=L1(1)/k(1); %-->miscalculation(1)=0;
   for i=2:j,
      C1predic(i)=L1(i)/k(i-1);
      k(i)=L1(i)/C1(i);
   end

   for i=1:j,
      errorC1(i)=C1(i)-C1predic(i);
   end
end

%P1 prediction
errorP1=zeros(1,j);
if P1(1)~=0
   k=zeros(1,j);
   P1predic=zeros(1,j);
   k(1)=L1(1)/P1(1);
   P1predic(1)=L1(1)/k(1); %-->miscalculation(1)=0;
   for i=2:j,
      P1predic(i)=L1(i)/k(i-1);
      k(i)=L1(i)/P1(i);
   end

   for i=1:j,
      errorP1(i)=P1(i)-P1predic(i);
   end
end

%P2 prediction
errorP2=zeros(1,j);
if P2(1)~=0
   k=zeros(1,j);
```

```
    P2predic=zeros(1,j);
    k(1)=L1(1)/P2(1);
    P2predic(1)=L1(1)/k(1); %-->miscalculation(1)=0;
    for i=2:j,
        P2predic(i)=L1(i)/k(i-1);
        k(i)=L1(i)/P2(i);
    end

    for i=1:j,
        errorP2(i)=P2(i)-P2predic(i);
    end
end

%D1 prediction
errorD1=zeros(1,j);
if D1(1)~=0
    k=zeros(1,j);
    D1predic=zeros(1,j);
    k(1)=L1(1)/D1(1);
    D1predic(1)=L1(1)/k(1); %-->error(1)=0;
    for i=2:j,
        D1predic(i)=L1(i)/k(i-1);
        k(i)=L1(i)/D1(i);
    end

    for i=1:j,
        errorD1(i)=D1(i)-D1predic(i);
    end
end

%D2 prediction
errorD2=zeros(1,j);
if D2(1)~=0
    k=zeros(1,j);
    D2predic=zeros(1,j);
    k(1)=L1(1)/D2(1);
    D2predic(1)=L1(1)/k(1); %-->miscalculation(1)=0;
    for i=2:j,
        D2predic(i)=L1(i)/k(i-1);
        k(i)=L1(i)/D2(i);
    end

    for i=1:j,
        errorD2(i)=D2(i)-D2predic(i);
    end
end

%T1 Prediction
errorT1=zeros(1,j);
if T1(1)~=0
    k=zeros(1,j);
    T1predic=zeros(1,j);
    k(1)=L1(1)/T1(1);
```

```
    T1predic(1)=L1(1)/k(1); %-->miscalculation(1)=0;
    for i=2:j,
        T1predic(i)=L1(i)/k(i-1);
        k(i)=L1(i)/T1(i);
    end

    for i=1:j,
        errorT1(i)=T1(i)-T1predic(i);
    end
end

%T2 Prediction
errorT2=zeros(1,j);
if T2(1)~=0
    k=zeros(1,j);
    T2predic=zeros(1,j);
    k(1)=L1(1)/T2(1);
    T2predic(1)=L1(1)/k(1); %-->miscalculation(1)=0;
    for i=2:j,
        T2predic(i)=L1(i)/k(i-1);
        k(i)=L1(i)/T2(i);
    end

    for i=1:j,
        errorT2(i)=T2(i)-T2predic(i);
    end
end

%S1 Prediction
errorS1=zeros(1,j);
if S1(1)~=0
    k=zeros(1,j);
    S1predic=zeros(1,j);
    k(1)=L1(1)/S1(1);
    S1predic(1)=L1(1)/k(1); %-->miscalculation(1)=0;
    for i=2:j,
        S1predic(i)=L1(i)/k(i-1);
        k(i)=L1(i)/S1(i);
         end

    for i=1:j,
        errorS1(i)=S1(i)-S1predic(i);
    end
end

%S2 Prediction
errorS2=zeros(1,j);
if S2(1)~=0
    k=zeros(1,j);
    S2predic=zeros(1,j);
    k(1)=L1(1)/S2(1);
    S2predic(1)=L1(1)/k(1); %-->miscalculation(1)=0;
    for i=2:j,
```

```
        S2predic(i)=L1(i)/k(i-1);
        k(i)=L1(i)/S2(i);
    end

    for i=1:j,
        errorS2(i)=S2(i)-S2predic(i);
    end
end

fprintf(ieec,'%13.0f %13.0f %13.0f %13.0f %13.0f
%13.0f %13.0f %13.0f %13.0f %13.0f
%13.0f\n',[L1_3; errorL2; errorC1; errorP1; errorP2; errorD1; errorD2; errorT1;
errorT2; errorS1; errorS2]);

fclose(ieec)



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%creation of the output file that will have hatanaka's version%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
name=file(1:(length(file)-5));
name(length(name):(length(name)+8))='_hata.txt';
hata = fopen(name,'w');

fprintf(hata, '    ************************************************    \n')
fprintf(hata, '    *** Compressed using Hatanaka Compression System ***    \n')
fprintf(hata, '    ************************************************    \n\n')

[L1,L2,C1,P1,P2,D1,D2,T1,T2,S1,S2]=
textread(file,'%f%f%f%f%f%f%f%f%f%f%f%*[^\n]');

j=size(L1); %= j=size(L2); =...
j=j(1);

%L1's 3th order
L1_3=zeros(1,j);
if L1(1)~=0
    L1_3(1) = L1(1);
    L1_3(2) = L1(2) - L1(1);
    L1_3(3) = L1(3) - 2*L1(2) + L1(1);
    for i = 4:j,
        L1_3(i) = (L1(i) - 3*L1(i-1) + 3*L1(i-2) - L1(i-3));
    end
end

%L2's 3th order
L2_3=zeros(1,j);
if L2(1)~=0
    L2_3(1) = L2(1);
    L2_3(2) = L2(2) - L2(1);
```

```
    L2_3(3) = L2(3) - 2*L2(2) + L2(1);
    for i = 4:j,
        L2_3(i) = (L2(i) - 3*L2(i-1) + 3*L2(i-2) - L2(i-3));
    end
end
%C1's 3th order
C1_3=zeros(1,j);
if C1(1)~=0
    C1_3(1) = C1(1);
    C1_3(2) = C1(2) - C1(1);
    C1_3(3) = C1(3) - 2*C1(2) + C1(1);
    for i = 4:j,
        C1_3(i) = (C1(i) - 3*C1(i-1) + 3*C1(i-2) - C1(i-3));
    end
end

%P1's 3th order
P1_3=zeros(1,j);
if P1(1)~=0
    P1_3(1) = P1(1);
    P1_3(2) = P1(2) - P1(1);
    P1_3(3) = P1(3) - 2*P1(2) + P1(1);
    for i = 4:j,
        P1_3(i) = (P1(i) - 3*P1(i-1) + 3*P1(i-2) - P1(i-3));
    end
end

%P2's 3th order
P2_3=zeros(1,j);
if P2(1)~=0
    P2_3(1) = P2(1);
    P2_3(2) = P2(2) - P2(1);
    P2_3(3) = P2(3) - 2*P2(2) + P2(1);
    for i = 4:j,
        P2_3(i) = (P2(i) - 3*P2(i-1) + 3*P2(i-2) - P2(i-3));
    end
end

%D1's third order
D1_3=zeros(1,j);
if D1(1)~=0
    D1_3(1) = D1(1);
    D1_3(2) = D1(2) - D1(1);
    D1_3(3) = D1(3) - 2*D1(2) + D1(1);
    for i = 4:j,
        D1_3(i) = (D1(i) - 3*D1(i-1) + 3*D1(i-2) - D1(i-3));
    end
end

%D2's 3th order
D2_3=zeros(1,j);
if D2(1)~=0
    D2_3(1) = D2(1);
```

```
    D2_3(2) = D2(2) - D2(1);
    D2_3(3) = D2(3) - 2*D2(2) + D2(1);
    for i = 4:j,
        D2_3(i) = (D2(i) - 3*D2(i-1) + 3*D2(i-2) - D2(i-3));
    end
end

%T1's 3th order
T1_3=zeros(1,j);
if T1(1)~=0
    T1_3(1) = T1(1);
    T1_3(2) = T1(2) - T1(1);
    T1_3(3) = T1(3) - 2*T1(2) + T1(1);
    for i = 4:j,
        T1_3(i) = (T1(i) - 3*T1(i-1) + 3*T1(i-2) - T1(i-3));
    end
end

%T2's th order
T2_3=zeros(1,j);
if T2(1)~=0
    T2_3(1) = T2(1);
    T2_3(2) = T2(2) - T2(1);
    T2_3(3) = T2(3) - 2*T2(2) + T2(1);
    for i = 4:j,
        T2_3(i) = (T2(i) - 3*T2(i-1) + 3*T2(i-2) - T2(i-3));
    end
end

%S1's 3th order
S1_3=zeros(1,j);
if S1(1)~=0
    S1_3(1) = S1(1);
    S1_3(2) = S1(2) - S1(1);
    S1_3(3) = S1(3) - 2*S1(2) + S1(1);
    for i = 4:j,
        S1_3(i) = (S1(i) - 3*S1(i-1) + 3*S1(i-2) - S1(i-3));
    end
end

%S2's 3th order
S2_3=zeros(1,j);
if S2(1)~=0
    S2_3(1) = S2(1);
    S2_3(2) = S2(2) - S2(1);
    S2_3(3) = S2(3) - 2*S2(2) + S2(1);
    for i = 4:j,
        S2_3(i) = (S2(i) - 3*S2(i-1) + 3*S2(i-2) - S2(i-3));
    end
end
```

```
fprintf(hata,'%13.0f %13.0f %13.0f %13.0f %13.0f
%13.0f %13.0f %13.0f %13.0f %13.0f
%13.0f\n',[L1_3; L2_3; C1_3; P1_3; P2_3; D1_3; D2_3; T1_3; T2_3; S1_3; S2_3]);

fclose(hata)
```

Comparativa(file, apredecir, orig).m

```
function mejor = comparativa_v2(file, apredecir, orig)
%the script plots two graphics showing the values obtained using the
%prediction method and the 3rd order filter method. It also returns a value
%indicating which of both compressions is the best (>1 --> prediction is
%better)
%file corresponds to the rinex rawc file to be read
%put in apredecir the type of observation to predict, for exemple: 'L1' o 'L2' o 'D1' ...
%at orig put the variable from which the prediction is made (for example: 'L1' o 'L2' o
'D1' ...)

format long
%reading the file
[L1,L2,C1,P1,P2,D1,D2,T1,T2,S1,S2]=
textread(file,'%f%f%f%f%f%f%f%f%f%f%f%*[^\n]');

%taking the columns that the user wants to predict
if strcmp(apredecir,'L1')
   vectorp = L1;
elseif strcmp(apredecir,'L2')
   vectorp = L2;
elseif strcmp(apredecir,'C1')
   vectorp = C1;
elseif strcmp(apredecir,'P1')
   vectorp = P1;
elseif strcmp(apredecir,'P2')
   vectorp = P2;
elseif strcmp(apredecir,'D1')
   vectorp = D1;
elseif strcmp(apredecir,'D2')
   vectorp = D2;
elseif strcmp(apredecir,'T1')
   vectorp = T1;
elseif strcmp(apredecir,'T2')
   vectorp = T2;
elseif strcmp(apredecir,'S1')
   vectorp = S1;
elseif strcmp(apredecir,'S2')
   vectorp = S2;
end

%taking the column which the user wants to predict from
if strcmp(orig,'L1')
   vectororig = L1;
elseif strcmp(orig,'L2')
```

```
    vectororig = L2;
elseif strcmp(orig,'C1')
    vectororig = C1;
elseif strcmp(orig,'P1')
    vectororig = P1;
elseif strcmp(orig,'P2')
    vectororig = P2;
elseif strcmp(orig,'D1')
    vectororig = D1;
elseif strcmp(orig,'D2')
    vectororig = D2;
elseif strcmp(orig,'T1')
    vectororig = T1;
elseif strcmp(orig,'T2')
    vectororig = T2;
elseif strcmp(orig,'S1')
    vectororig = S1;
elseif strcmp(orig,'S2')
    vectororig = S2;
end

j=size(vectorp);
j=j(1);
%3th order (hatanaka)
vectorp_3=zeros(1,j);
if vectorp(1)~=0
    vectorp_3(1) = vectorp(1); %0 order
    vectorp_3(2) = vectorp(2) - vectorp(1); %1st order
    vectorp_3(3) = vectorp(3) - 2*vectorp(2) + vectorp(1); %2nd order
    for i = 4:j,
        vectorp_3(i) = (vectorp(i) - 3*vectorp(i-1) + 3*vectorp(i-2) - vectorp(i-3)); %3rd
order
    end
end

plot(vectorp_3,'b')
hold on

%calculation of the value that will represent the graphic, it consist on
%making the mean value from the absolute value of all the values of vectorp_3:
vectorp_3med = sum(abs(vectorp_3))/length(vectorp_3);


%miscaalculation (from prediction calculations)
K=zeros(1,j);
vectorppredic=zeros(1,j);
K(1)=vectororig(1)/vectorp(1);
vectorppredic(1)=vectororig(1)/K(1); %-->miscalculation(1)=0;
%prediction method
for i=2:j,
        vectorppredic(i)=vectororig(i)/K(i-1);
    K(i)=vectororig(i)/vectorp(i);
end
```

```
error=zeros(1,j);

for i=1:j,
   error(i)=vectorp(i)-vectorppredic(i);
end

plot(error,'r')
ylim([-1000 1000])
title(file)
ylabel('Compressed values')
legend('3rd order filter compression','prediction compression')

%calculation of the value that will represent the prediction
%miscalculation's graphic , it consists on making
% the mean value of the absolute value from all the values of the
% miscalculations.
error_med = sum(abs(error))/length(error);

%finally, a division between the calculated medias is made, so if the division is higher
than
%1, the miscalculation will be smaller than the miscalculation from the 3th
%order, and if the division is smaller the 3th order will have a smaller
%miscalculation.

mejor = vectorp_3med/error_med;
```