University of Southampton

# The design and development of a satellite navigation simulator with ionospheric correction

by Mikaël Thibaudeau

Master of Philosophy

Faculty of Engineering and Applied Sciences
Aeronautics and Astronautics

October 1999

University of Southampton

**Abstract**

Faculty of Engineering and Applied Sciences

Department of Aeronautics and Astronautics

Master of Philosophy

# The design and development of a satellite navigation simulator with ionospheric correction

by Mikaël Thibaudeau

The need for precise navigation is now a necessity for all mobile vehicles. For example, a navigator needs to know where his competitors are during a race, a rocket has to reach the chosen target, and a pilot has to land his plane even if the runway is invisible. The navigation requirement of today and the future demand the most accurate equipment.

This thesis describes the design and development of a satellite navigation simulator. The simulator has all the characteristics of a GPS receiver (i.e. architecture and algorithm), and can use satellite data from STK simulations, or from real data files acquired by a GPS dual frequency receiver located in Toulouse (France). The aim of such a simulator is to enable the user to study the influence of several parameters on the navigation solution accuracy, and to study some new improvements to the basic GPS system.

A significant part of the thesis is dedicated to the study of the ionospheric delay, which alters the satellite signals broadcast to the receiver. A new tool was added to the program, to compute the ionospheric corrections provided by EGNOS. This new module is being used to help the EGNOS designers to choose the most accurate model of the ionosphere used in the EGNOS algorithm.

## Acknowledgement

I am very grateful to many people who have contributed to the success of this year and to the completion of this work.

I first wish to thank my supervisor, Dr A.R.L. Tatnall, who provided me with constant support and his constructive criticisms.

I am also grateful to Frédéric Brunel, from Alcatel Space Industries in Toulouse (France), who agreed to spend a lot of his time answering my questions, and who provided me with the main part of the documents and data needed during this year. I also would like to thank Frédéric Cornet, with whom I worked for the last months of this project.

Thanks are also due to the staff of ENSICA, which allows me to come at the University of Southampton as an exchange student and to live a unique experience for my last year of study.

I would like to thank the postgraduates and friends from the department of Aeronautics and Astronautics for their linguistic support, and above all for their welcome during all this year.

Je voudrais finallement remercier mes parents, qui m'ont toujours apporté leur soutient au cours de mes études et permis de choisir ma voie. Cette année à Southampton n'aurait pas été possible sans eux.

# Table of Content

# Table of figure

## Acronyms

| | |
|---|---|
| C/A | : Coarse Acquisition Code. |
| CNES | : Centre National d'Etudes Spatial. |
| DFLR | : Deutsche Forschungsanstald für Lunftund Raumfahrt. |
| DGPS | : Differential GPS. |
| DoD | : US Department of Defence. |
| DOP | : Dilution Of Precision. |
| DUNSS | : Dynamic User Navigation Solution Simulator. |
| EGNOS | : European Geostationnary Navigation Overlay System. |
| ESA | : European Space Agency. |
| GEO | : Geostationary Earth Orbit. |
| GLONASS | : GLObal NAvigation Satellite System. |
| GNSS | : Global Navigation Satellite System (generation I and II). |
| GPS | : Global Positioning System. |
| IGP | : Ionospheric Grid Point. |
| IPP | : Ionospheric Pierce Point. |
| L1 | : First GPS frequency, 1575.42 MHz. |
| L2 | : Second GPS frequency, 1227.6 MHz. |
| RIMS | : Ranging and Integrity Monitoring Station. |
| SA | : Selective Availability. |
| STK | : Satellite ToolKit. |
| TEC | : Total Electron Content. |
| WGS | : World Geodetic System. |

# 1 Introduction

In recent years, there has been a going requirement for precise navigation for civilian use as well as military applications. This trend is due to the growing importance of precision, either for military (rockets...) or for civilian (leisure sailing, car driving...) use, and also to a new navigation system, cheaper and more precise that the previous ones, and easily available to everybody, i.e. satellite navigation.

Satellite navigation is a young type of navigation system (Logsdon, 1986 [1]). The first system, called TRANSIT, was launched in the beginning of the 60's by the DoD (US Department of Defense). Therefore, a lot of improvements are still being made in order to improve the only current system that is working properly: the GPS (Global Positioning System).

This thesis describes the work undertaken by the author in contributing to these improvements.

The first aim was to use the current knowledge on satellite navigation, and especially on GPS, to write a program that could simulate a satellite navigation receiver. The purpose was to provide a simulator that can be used by the industry to study the influence on the receiver position of several parameters:

- Error on satellite signals.
- Receiver environment.
- Receiver movement.
- Number of satellites in view.

This part of the thesis was mainly a work of synthesis of all the current knowledge available on satellite navigation, in order to design and build a new tool. The validation of this tool was made by doing several parametric studies, especially concerning the error on satellite signal, to make the simulator as close as possible to a real receiver.

The second stage was the study of one of the errors on GPS satellite signal, i.e. the ionospheric delay. It is one of the most important sources of error because of its great variations dependent on the time (e.g. difference between daytime and nighttime, solar activity) and on the receiver and satellites positions.

The model used by a GPS receiver for ionospheric correction is the model of Klobuchar (Klobuchar, 1987 [23], 1996 [5]), but a lot of other algorithms are being studied in order to improve this model. The ESA (European Space Agency) is developing, in collaboration with European companies, a new system, called EGNOS [11], which is planned to be used as a complement to GPS.

Alcatel Space Industries, which is contributing to this development, needed a tool to test the performances of the ionospheric corrections provided by EGNOS, and to compare them with those given by the model of Klobuchar. Therefore, the author also added a new module to the program, dedicated to EGNOS ionospheric corrections, and made a first set of tests to compare the corrections provided by both models.

The structure of this thesis follows the different stages of the work: The first part is dedicated to the generalities of satellite navigation, and to the algorithm used in the GPS receiver to compute its position. The second part deals with the structure of the program (requirement, design...) and the results of the first parametric studies. The third part presents the different ionospheric models, and a comparison of their results with the real ionospheric delay. Finally a conclusion summarises the results of the work, and tries to define the studies that could be done later with this program.

# 2 Satellite Navigation System

## 2.1 Generalities

### 2.1.1 History

The first satellite navigation system, called TRANSIT Navigation System, operated from 1963 (Logsdon, 1986 [1]). It was launched by the U.S. Department of Defense (DoD) to provide to the U.S. army an accurate and reliable positioning system. There are currently two satellite navigation systems: The GPS (Global Positioning System) funded by and controlled by the DoD, and the GLONASS (GLObal NAvigation Satellite System) [32][33], developed by the Russians, but which is not actually very efficient because of the non-replacement of the old satellites.

These satellite navigation systems provide the time, position and velocity to every user everywhere in the world, military or civilian. It requires cheap equipment, the information provided is accurate, and free; and it can be used in boats, aircraft, or cars.

### 2.1.2 Operation

To explain the satellite navigation operation, most of the data examples will come from the only satellite navigation system, which actually works efficiently: the GPS.

The system can be divided into three parts as shown in *Figure 2-1* [33]: the space segment, the control segment and the user segment. The space segment (navigation satellites) sends to the user segment (receiver) and to the control segment (control station), data on its state (satellites position, satellites time, corrections). The control segment sends to the space segment some corrections on its state.

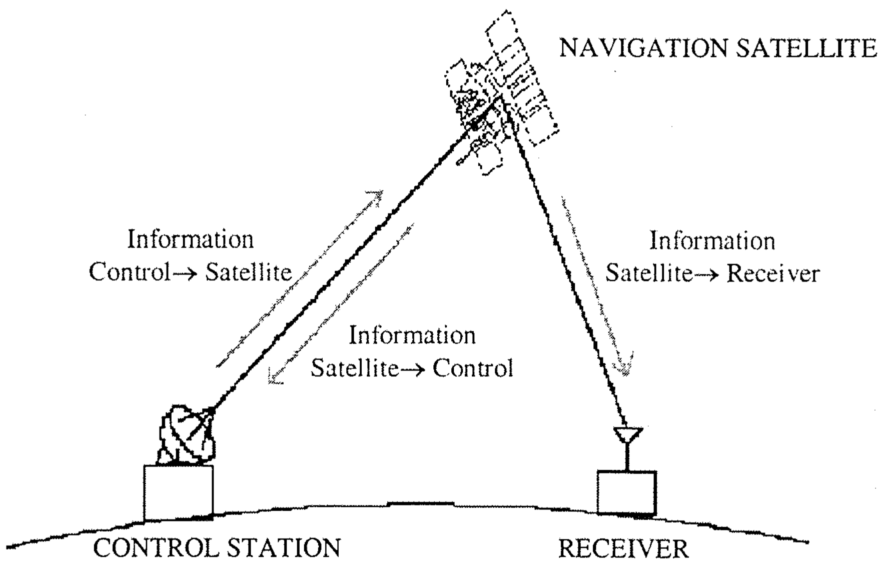

NAVIGATION SATELLITE
Information Control→ Satellite
Information Satellite→ Control
Information Satellite→ Receiver
CONTROL STATION
RECEIVER

**Figure 2-1- Three Segments Repartition**

### *2.1.2.1 The space segment*

The space segment is composed of the satellites, which sends radio signals from space to the user. The satellites constellation configuration is chosen in order to provide the best coverage of the whole Earth surface as shown in *Figure 2-2* and *Figure 2-3* [33] In the case of GPS, there are 24 satellites that orbit the Earth in 12 hours, at 20,200 km altitude. The orbit attitude is such that the satellites repeat the same track and configuration over any point approximately each 24 hours (exactly each sideral day, 23h56mn). There are six orbital planes inclined at about 55 degrees with respect to the equatorial plane, equally spaced (60 degree apart), with four satellites in each. This configuration is designed to have at least four satellites in view from everywhere in the Earth.

The GLONASS was planned to use 24 satellites also, but Russia does not replace the satellites that stop working.

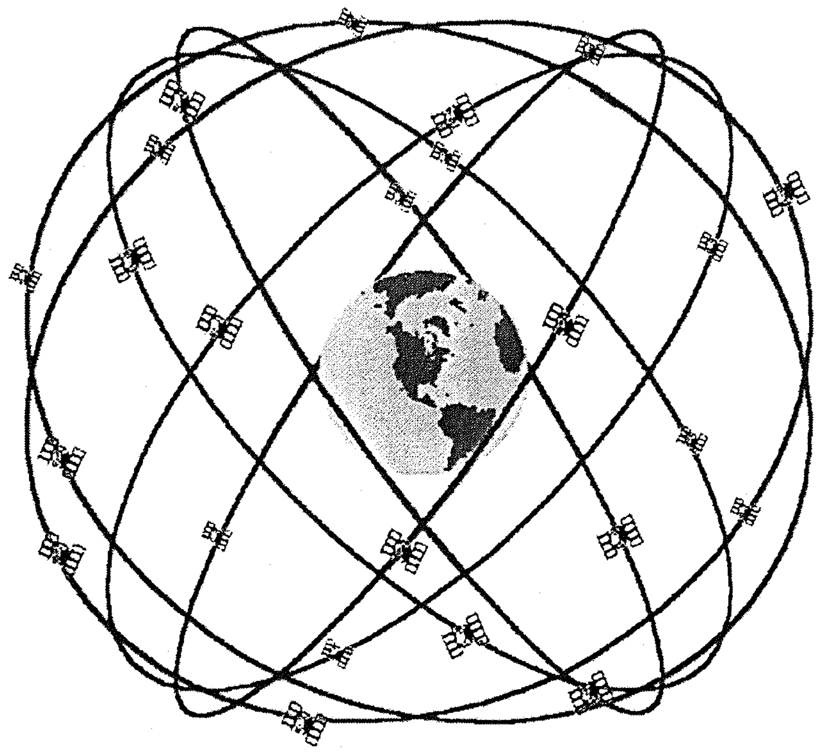


**Figure 2-2- GPS Constellation**

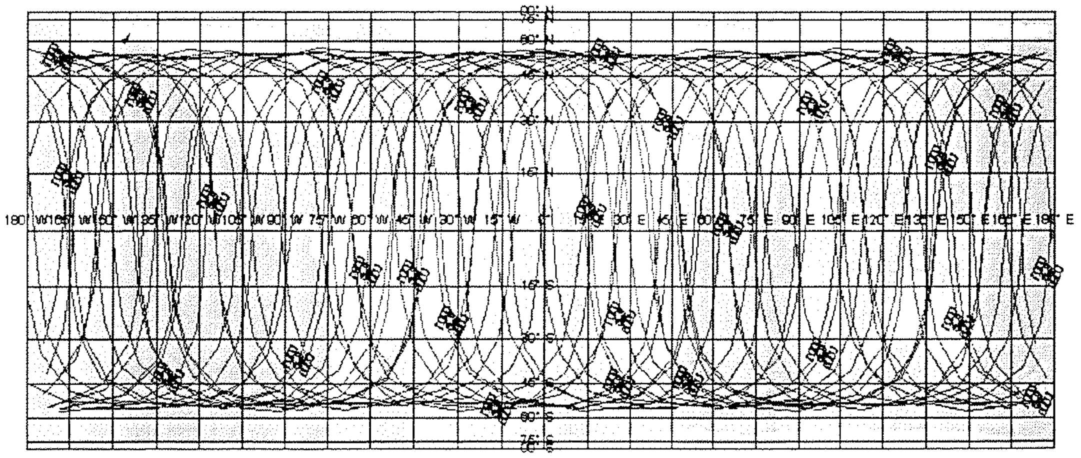


**Figure 2-3- GPS Ground Trace**

A satellite navigation system could use other sorts of satellite orbits. The satellites in a geostationary orbit, for example, are fixed relative to a point on the ground. These satellites rotate in the equatorial plane and provide a good coverage of a part of the earth, but they can not cover the regions at latitudes higher than 70 degrees. Therefore, this type of orbit is not suitable for navigation. The best coverage is obtained with types of constellation used for GPS or GLONASS.

The GPS satellites can be divided into major families having similar physical characteristics (Logdson, 1986 [1]. [32]):

- The initial block I satellites weighed 760 kg each and generated 420 watts end-of-life electrical power. With one exception, all 11 of the block I satellites reached orbit successfully and all of them operated approximately 3 years or more. Some were still functioning 13 years after launch. The only launch which failed was carrying the GPS satellite number I-07. Almost all satellites from the block I were launched by the Atlas rocket.
- The block II (*Figure 2-4* [32]) is composed of 27 satellites, each weighing 1665 kg, and generating 700 watts end-of-life electrical power. They were planned to be launched by the shuttle, but after the explosion of Challenger, they were reassigned to the DELTA rocket.
- The block IIR composed of 1 satellite launched in 1998, weighing 2032 kg, and generating 1,000 watts end-of-life electrical power.

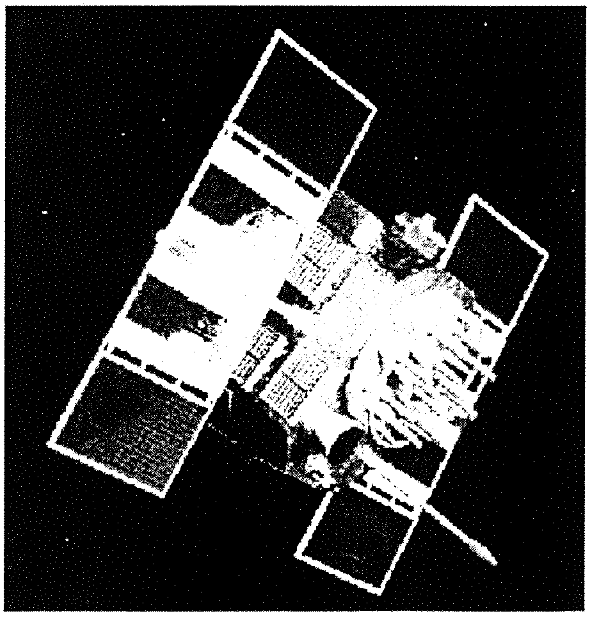


**Figure 2-4- GPS Satellite (Block II)**

The navigation satellites provide a signal, transmitted on two frequencies: L1=1575.42 MHz and L2=1227.6 MHz (Logdson, 1986 [1]. Spilker, 1996 [2]). To determinate its position, a receiver measures the signal travel times associated with the binary pulse train from four or more satellites. The signal travel time, multiplied by the speed of light equals the slant range from the satellite to the user. By receiving the signal from four satellites, the receiver has four equations with four unknown values: the three receiver position values, and its time (which is not exactly the system time).

All the carrier waves from the satellites are right-hand circular polarised. This is accomplished by using 12 spiral-wound helical antennas arranged in a tight pattern. Four of them are located in the centre quad. The other eight are arranged in a circular ring surrounding the centre quad. Circular polarisation allows the user-set antennas to access the faint satellite signals without boresight pointing.

Every satellite transmits continuously on the same two L band frequencies. The receiver uses code-division multiple access to distinguish the satellites from one another. Before 1993, two different binary codes, the C/A code and the P code were superimposed on the two L band carrier waves emanating from each satellite. The C/A code (Coarse Acquisition Code) is available free of charge to civilian users all around the world. The P code (Precision Code) is reserved for high precision military users. It is protected through encryption techniques that restrict access and deny full accuracy to non-authorised users. Since 1993, the P code is not directly transmitted, but is first mixed with the C/A code within the Y code, which is broadcast.

Each of the satellites in the GPS constellation is assigned its own unique C/A code and its own unique P code. The C/A code has a chipping rate of 1 million bits per second, with a repetition interval of 1,023 bits. Thus, it repeats after approximately one-thousandth of a second. The P code has a chipping rate of 10 million bits per seconds. Its repetition interval is approximately $6.10^{12}$ bits. Seven days elapse before the P code sequence repeats. Both the C/A and the P codes are pseudorandom binary pulse sequences, with a high degree of "randomness" in their binary 1s and 0s. The "randomness" is only apparent, the binary codes are generated by precise mathematical relationships with total predictability. Phase shift key modulation is used to mark the interfaces between the binary 1s and 0s. This means that the L1 and L2 carrier waves experience sharp mirror image reflections whenever the C/A code or the P code switches from a binary 1s to a binary 0s, and vice versa. An opportunity for an instantaneous phase shift occurs every one-millionth of a second for the C/A code, and every ten-millionth of a second for the P code.

The L1 frequency carries both the C/A code and P code transmitted in phase quadrature (90 degrees out of phase to one another), and the L2 carrier wave is modulated with the military P code only.

With the P-code, we have a predictable accuracy of:

- 22 metres on horizontal accuracy
- 27.7 metres on vertical accuracy
- 100 nanoseconds on the time accuracy

With the C/A code, which is voluntarily degraded by the DoD using Selective Availability (SA), we have a predictable accuracy of:

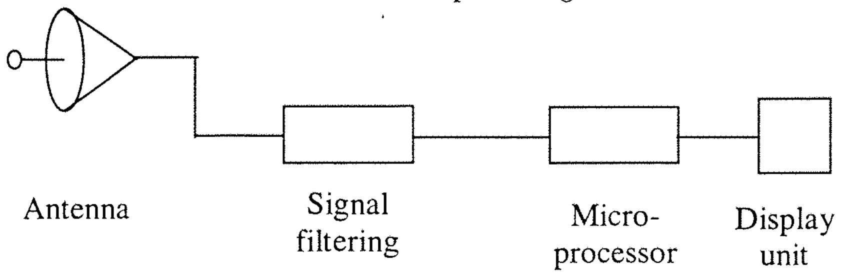
- 100 metres on horizontal accuracy
- 156 metres on vertical accuracy
- 340 nanoseconds on the time accuracy

The accuracy can be improved using some additional techniques like Differential GPS, or other navigation systems integrated with GPS.

#### 2.1.2.2 The User Segment

The purpose of the user segment is to process the time and position an information from four or more satellites (either simultaneously or sequentially), to obtain accurate position, velocity, and timing measurements. A receiver can be divided into three major components: the antenna with its associated electronics, the receiver-processor unit, which picks up the satellite signals and performs the navigation solution, and the control-display unit, which provides information display and a convenient interface between the user and the satellite navigation system.

The GPS receivers are used for navigation in boats, planes, and now cars. They are also used in geodetic control, tectonic sciences, and atmospheric studies. A schematic is shown below and an example in *Figure 2-5*.



GPS receiver: GARMIN GPSMAP 175
These kinds of GPS receiver are really small (193*74*53 mm, 635 grams) efficient and precise. They can provide an accuracy of 15 meters on the receiver position using the Differential GPS. They can also provide an accuracy of 0.1 knot on the velocity if the user's speed is stabilised.

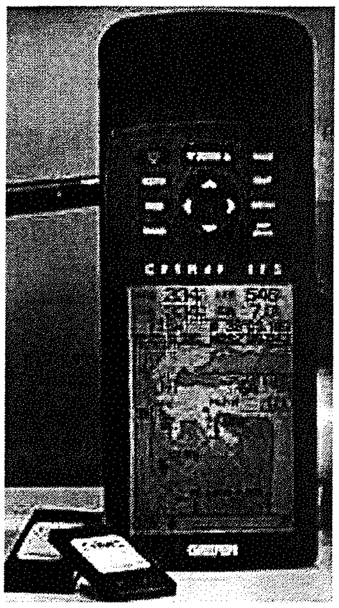


**Figure 2-5- GPS receiver**

### *2.1.2.3 The Control Segment*

The purpose of the control segment is to track the navigation satellites and provide them with periodic updates, correcting their ephemerides constants and their clock-bias errors. Five unmanned monitor stations serve the GPS block II constellation (see *Figure 2-6* [33]). A Master Control Station is located at Colorado Spring.
To correct the error on satellite parameters, four monitor stations pick up the navigation signals from a particular satellite at the same time. The monitor stations are independently surveyed to fix their position, and they are equipped with synchronised cesium atomic clocks to identify the time to high degree of precision. The four pseudo-range measurements are then transmitted to the master control station, used in an inverted navigation solution (see *part 2.2*) to fix the location of the satellite and to determine the timing errors in its onboard atomic clock. This information are then relayed to each satellite once per day on S band from various antennas positioned around the globe.

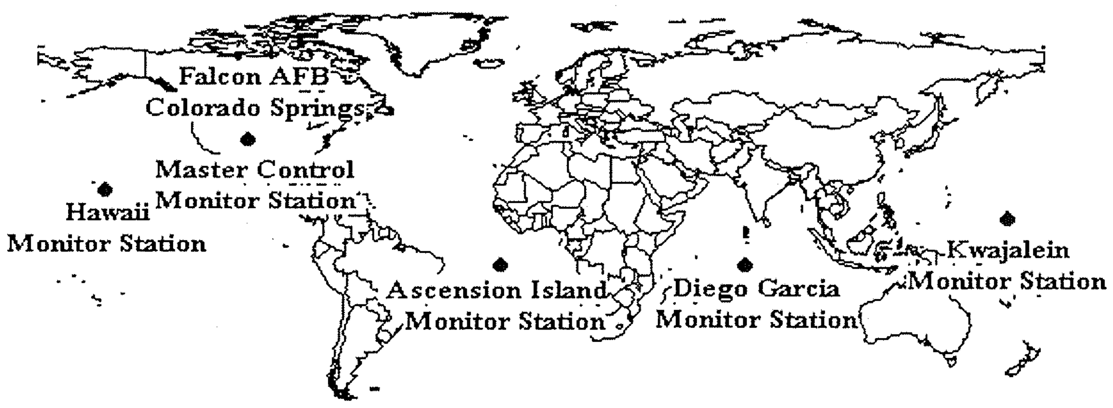


**Figure 2-6- Five Control Stations**

### 2.1.3 Integration of satellite Navigation with another navigation system

The combination of two Navigation Systems can greatly improve the accuracy on the user position. It is called an integrated navigation system (Logdson, 1986 [1]).
One populår type of integrated navigation system links an inertial navigation unit with GPS receivers so that both devices can contribute inputs to the current navigation solution. A combined system of this type provides consistently superior results, compared with those that could be achieved by either unit working alone. An integrated navigation system has been constructed, tested, and installed aboard many different types of vehicles (e.g. helicopters, aircraft, ships, submarines...). It is very interesting for military applications, because GPS antennas could lose the satellites signals during military manoeuvres.
Compared with a stand-alone GPS receiver, an integrated navigation system usually provides substantial improvements in accuracy and stability, jamming integrity and high-dynamic operations. The GPS receiver helps the inertial navigation system by providing faster signal acquisition and reacquisition, and providing to the inertial system some real-time and periodic updates. The inertial navigation system improves the jamming immunity and dynamic operation of GPS receiver.
In particular, the inertial navigation system can help the GPS by providing accurate initial estimates on position and velocity, thus reducing the time required for the

GPS receiver to lock onto the signals streaming down from the satellites. If one or more of the satellite signals is subsequently lost due to receiver malfunction, terrestrial blockages, wing shielding, or enemy jamming, the inertial navigation system can help achieve reacquisition quickly and efficiently. The continuous velocity measurements obtained from the inertial navigation system allow the GPS receiver to estimate the magnitude of the current Doppler shift so that it can narrow the bandwidth of its tracking loops. This improves the dynamic operation of the integrated navigation system and also increases its jamming immunity.

The GPS receiver can help the inertial navigation system with accurate and real-time updates of the receiver position. This helps it to provide more stable and accurate navigation solutions.

In *Figure 2-7*, (Logdson, 1986 [1]), the position error is shown as a function of the satellites in view. When four satellites are in view from the receiver, the error with only the GPS or with an integrated navigation system is nearly the same (difference insignificant on this experiment). But when the GPS receiver is combined with inertial system, the loss of two satellites is rapidly compensated whereas using only the GPS induces a more important error, which needs more time to be compensated.

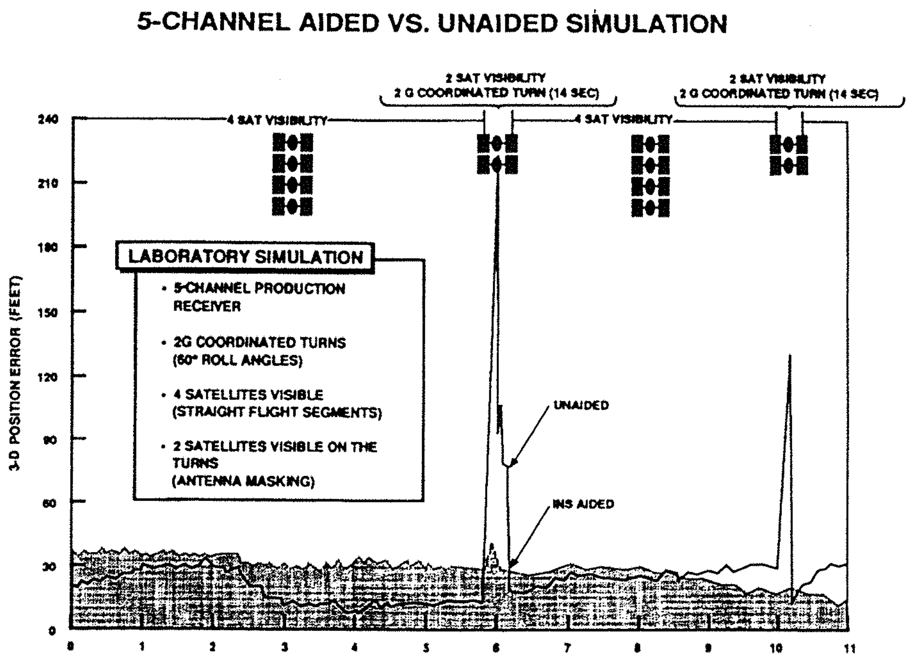


Figure 2-7- Influence of integrated navigation on the position accuracy

## 2.2 Navigation Solution

The receiver processor, after filtering the signal coming from the satellite, is in charge of computing the receiver position from the satellite data. The following sections display the algorithm used in the GPS receiver processor (Axelrad and Brown, 1996 [3]).

### 2.2.1 Basic equations

The navigation satellite number i provides to the receiver its position and its time: $X_i, Y_i, Z_i$ and $t_{ti}$ (transmission time of satellite i).

The receiver needs to know its position and its time: $X_U, Y_U, Z_U$ and $t_{ru}$ (the user time at the satellite signal reception).

Therefore, we can write the equations:

$$(X_i - X_U)^2 + (Y_i - Y_U)^2 + (Z_i - Z_U)^2 = c(t_{ru} - t_{ti})^2$$

**Equation 2-1**

where $i = \{1,2....n\}$ with n the number of satellites in view. As we have four unknown values to find, we need four equations. Therefore, n shall be greater than or equal to four.

This equation can be interpreted as shown in the *Figure 2-8* [33]: the receiver is at the intersection of the four spheres centred on each satellite, and with a radius of $\left\| \overrightarrow{receiver - satellite} \right\|$. The measured pseudorange between the satellite and the receiver is not exactly the real pseudorange. This error is due to the receiver clock bias (difference between receiver time and GPS time: ten nanoseconds are equivalent to 3 meters of error), and also to several types of error on the satellite signal (cf *part 2.2.2.2*).

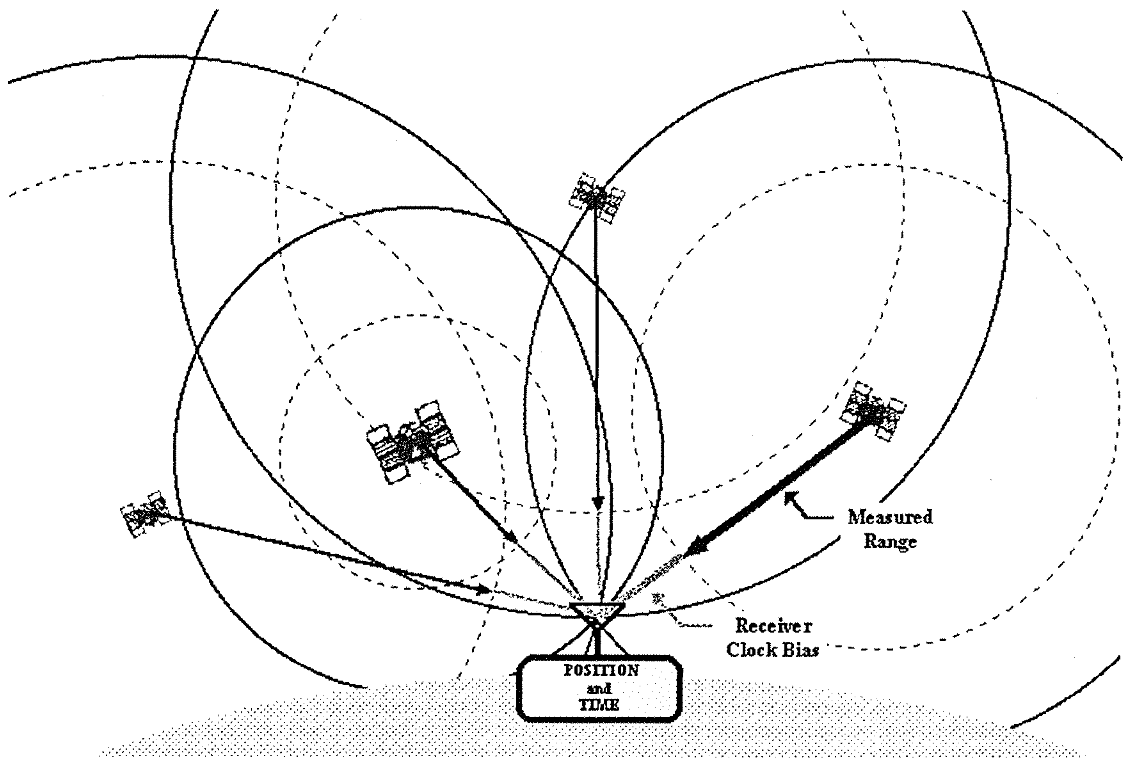


**Figure 2-8- The receiver is at the intersection of four spheres**

Unfortunately, the Equation 2-1 is not linear, and needs the use of an iterative algorithm to be solved. The algorithm most adapted to this problem, and usually used in GPS receivers, is the least squares method.

### 2.2.2 Single point solution

The method which gives a single-point solution uses the navigation estimate provided by the least squares solution to the measurement equations made at a single time (D.E. Weells and al., 1986 [18]. Axelrad and Brown, 1996 [3]).

First of all, the computation of the pseudorange is required. Then, this pseudorange is used in the least squares algorithm.

### 2.2.2.1 Pseudorange

When the signal processor delay lock loop finds the point of maximum correlation with a given GPS satellite signal, it produces an observation of the code phase, or equivalently, signal transmit time $t_T$ for the current local receive time $t_R$. The observed signal propagation delay is ($t_R$ - $t_T$). The pseudorange observable is merely this time interval scaled by the speed of light in vacuum: $\rho = c.(t_R - t_T)$.

The pseudorange observation between the user and the satellite i can be related to the user position and clock states as follows (Axelrad and Brown, 1996 [3]):

$$\rho_i = |r_i - r_u| + c.b_u + \varepsilon_{\rho i}$$

**Equation 2-2**

where $r_i$ is the satellite position at transmit time (in meter); $r_u$ is the receiver position at receive time (in meter); $b_u$ is the bias in the receiver clock (in second); $\varepsilon_{\rho i}$ is the composite of errors produced by atmospheric delays, satellite ephemeris mismodeling, selective availability (SA), receiver noise, etc (in meter).
The state to be estimated, consisting of $r_u$ and $c.b_u$ is embedded in this measurement equation. To extract it, we must linearize the measurement equation about some nominal value, for example, around our current best estimate.

In the following explanations, we use the notation $\overline{\ldots}$ for the estimated values and estimated vectors.

Given an a priori estimate of the state $\overline{x} = \left[ \overline{r}_u^T \quad c\overline{b}_u \right]$ and an estimate of the bias contributions caused by ionospheric and tropospheric delay, relativistic effects, and satellite clock errors $\overline{\varepsilon}_{\rho i}$, we can predict that the pseudorange measurement should be as follows:

$$\overline{\rho}_i = |r_i - \overline{r}_u| + c.\overline{b}_u + \overline{\varepsilon}_{\rho i}$$

**Equation 2-3**

The measurement $\Delta\rho$ which is the difference between the predicted and actual measurement, can be modeled as linearly related to the error in the state estimate,

$\Delta x \equiv \left[\Delta r_u^T \quad c.\Delta b_u\right]$, by performing a Taylor expansion about the current state estimate. The linearized result is given by the following equation:

$$\Delta\rho_i = \bar{\rho}_i - \rho_i = \begin{bmatrix} -\bar{1}_i^T & 1 \end{bmatrix} \begin{bmatrix} \Delta r \\ c.\Delta b \end{bmatrix} + \Delta\varepsilon_{\rho i}$$

**Equation 2-4**

where $\bar{1}_i \equiv \dfrac{r_i - \bar{r}_u}{\left|r_i - \bar{r}_u\right|} = \dfrac{1}{\sqrt{(X_i - X_u)^2 + (Y_i - Y_u)^2 + (Z_i - Z_u)^2}} \begin{bmatrix} X_i - \bar{X}_u \\ Y_i - \bar{Y}_u \\ Z_i - \bar{Z}_u \end{bmatrix}$

$$\Delta r \equiv \bar{r}_u - r_u = \begin{bmatrix} \bar{X}_u - X_u \\ \bar{Y}_u - Y_u \\ \bar{Z}_u - Z_u \end{bmatrix},\ \Delta b \equiv \bar{b}_u - b_u,\ \Delta\varepsilon_{\rho i} \equiv \bar{\varepsilon}_{\rho i} - \varepsilon_{\rho i}$$

$\bar{1}_i$ is the estimated line of sight unit vector from the user to the satellite number i.

$\Delta\varepsilon_{\rho i}$ is the residual error after the known biases have been removed.

The *Equation 2-4* is the fundamental GPS pseudorange measurement equation.

### 2.2.2.2 *Error on satellite signal*

In this section, a general presentation of the $\Delta\varepsilon_{\rho i}$ is given (E.H. Martin, 1980 [26]. Parkinson, 1996 [4]). This term represents the difference between the estimated error on the satellite signal, and the real error on the satellite signal. $\Delta\varepsilon_{\rho i}$ can be decomposed into several errors: $\Delta\varepsilon_{\rho i} = \sum_{k=1}^{6} \Delta_k \varepsilon_{\rho i}$.

Each term of the sum represents a type of error on the satellite signal:

- The error on the satellite ephemeris: each satellite transmits in its signal its position, which is an estimated one, different from the real one. This type of error is sometimes really important ($\approx$100 meters on the satellite signal), and produces error on the pseudorange (J.F. Zumberge, W.I Bertiger, 1996 [8]).

- The error on the satellite clock: In the same way, the satellite time transmitted in the satellite signal is not the exact one. For example, a typical error of ten nanoseconds on the satellite time induces an error of 3 meters on the pseudorange (J.F. Zumberge, W.I Bertiger, 1996 [8]).
- The ionospheric delay: When the signal meets the free electrons of the atmosphere (which are mainly in the ionosphere), it is delayed, and the pseudorange is modified (J.A. Klobuchar [5], 1996). The ionospheric delay can vary quickly and with a great amplitude.
- The tropospheric delay: the atmospheric gases contained in the troposphere delay the satellite signal (J.J. Spilker Jr, 1996 [6]). The dry part of the troposphere is the cause of the main part of the tropospheric delay, but varies slowly (less than 1% in a few hours) and is reasonably predictable. The wet part has a slighter influence on the satellite signal, but varies markedly and is not easily predictable.
- The multipath: The signal can be reflected on some surface around the receiver. The apparent time of signal travel between the satellite and the receiver is then altered (M.S. Braasch, 1996 [7]). The multipath error has a sinusoidal behaviour due to the periodical signal repetition.
- The noise: The user antenna and the user environment induce some noise on the satellite signal, which modify the pseudorange.

In *part 3.6.3*, the influence of these errors is studied, by computing them with white noise. In *part 4*, two models of ionospheric error are shown.

#### 2.2.2.3 Least squares method

For each satellite tracked by the receiver, the predicted pseudorange is formed using *Equation 2-2*, and the linearized observation *Equation 2-4* is formed (Axelrad and Brown, 1996 [3]). All the measurements are then combined into a set of equations:

$$\Delta\rho = G.\Delta x + \Delta\varepsilon_{\rho}$$

**Equation 2-5- Observation equation**

$$\text{where: } \Delta\rho \equiv \begin{bmatrix} \Delta\rho_1 \\ \Delta\rho_2 \\ \vdots \\ \Delta\rho_n \end{bmatrix}, \ G = \begin{bmatrix} -\bar{1}_1^T & 1 \\ -\bar{1}_2^T & 1 \\ \vdots & \vdots \\ -\bar{1}_n^T & 1 \end{bmatrix}, . \Delta x \equiv \begin{bmatrix} \Delta r_u \\ c.\Delta b_u \end{bmatrix}, \ \Delta\varepsilon_\rho = \begin{bmatrix} \Delta\varepsilon_{\rho 1} \\ \Delta\varepsilon_{\rho 2} \\ \vdots \\ \Delta\varepsilon_{\rho n} \end{bmatrix}$$

In GPS, "G" is frequently referred to as the geometry matrix, and corresponds to the measurement connection matrix, commonly named "H" in the more general literature on filtering (T. Kailath, 1977 [17]).

If a weight matrix W is assigned to each observation, the weighed least square estimate is as follows:

$$\Delta\bar{x} \equiv \left(G^T W G\right)^{-1} G^T W \Delta\rho$$

**Equation 2-6- Normal equation**

Afterwards, the current step estimate is computed with:

$$\bar{x} = \bar{x} - \Delta\bar{x}$$

**Equation 2-7**

And then the equations are iterated.

The process is iterated following two loops: the data acquisition loop (every second for GPS receiver) and the iteration loop, included in the first one. At each acquisition, the data are read. At each iteration the error vector $\Delta\varepsilon_\rho$, the G matrix, the weighed matrix W, the $\Delta\rho$ and the $\Delta x$ are computed.

If the a priori estimate used to construct G is in error by a lot (typically more than a few kilometres), the least squares solution may be iterated until the change in the estimate is sufficiently small. Because G only depends upon the line of sight unit vector, it is not very sensitive to errors in the receiver position.

### 2.2.2.4 *Solution accuracy and dilution of precision*

The accuracy of the single point solution depends on two factors: the measurement quality ($\Delta\varepsilon_\rho$ ), and the user to satellites geometry (B.W. Parkinson, 1996 [4]).

The precision in the measurement is described in the following parts. It is described by the variance of the measurement error, which for a typical pseudorange is in the range of 0.3 to 300 metres, depending on the error conditions.
The geometry is described by the G matrix, which is composed of line of sight vectors and "1" for clock states.

The solution error covariance can be expressed as follows (P. Axelrad, R.G. Brown, 1996 [3]):

$$E\left[\Delta\bar{x} \quad \Delta\bar{x}^T\right] = E\left[\left(G^TG\right)^{-1}G^T\Delta\rho \quad \Delta\rho^T G\left(G^TG\right)^{-1}\right] = \left(G^TG\right)^{-1}G^TV^{-1}G\left(G^TG\right)^{-1}$$

**Equation 2-8**

where $V^{-1}$ is the pseudorange measurement covariance. If it is assumed (somewhat incorrectly, as described later) that the measurement errors are uncorrelated and have equal variance $\sigma^2$, then $V^{-1}=\sigma^2.\mathrm{Id}$, with Id the identity matrix, and the point solution error covariance reduces to the following:

$$E\left[\Delta\bar{x} \quad \Delta\bar{x}^T\right] = \sigma^2.\left(G^TG\right)^{-1}$$

**Equation 2-9**

If the state is written in the receiver local reference so that $\Delta x = \left[\Delta East \quad \Delta North \quad \Delta Up \quad c.\Delta b\right]^T$, where ΔEast, ΔNorth, ΔUp are the east north and up position errors, and c.Δb is the clock bias error, then the variance of the state estimates is given by the following:

$$E\left[\Delta\bar{x} \quad \Delta\bar{x}^T\right] = \begin{bmatrix} E\left[\Delta E^2\right] & E[\Delta E\Delta N] & E[\Delta E\Delta U] & E[\Delta Ec.\Delta b] \\ E[\Delta N\Delta E] & E\left[\Delta N^2\right] & E[\Delta N\Delta U] & E[\Delta Nc.\Delta b] \\ E[\Delta U\Delta E] & E[\Delta U\Delta N] & E\left[\Delta U^2\right] & E[\Delta Uc.\Delta b] \\ E[c.\Delta b\Delta E] & E[c.\Delta b\Delta N] & E[c.\Delta b\Delta U] & E\left[c.\Delta b^2\right] \end{bmatrix}$$

**Equation 2-10**

Most of the time, the diagonal elements are the only interesting terms. The following DOPs summarise the contribution of the geometry:

$A = \left(G^TG\right)^{-1}$

$GDOP = \sqrt{trace(A)}$ Geometrical DOP

| | |
|---|---|
| $PDOP = \sqrt{A_{11} + A_{22} + A_{33}}$ | Position DOP |
| $HDOP = \sqrt{A_{11} + A_{22}}$ | Horizontal DOP |
| $VDOP = \sqrt{A_{33}}$ | Vertical DOP |
| $TDOP = \sqrt{A_{44}}$ | Time DOP |
| $EDOP = \sqrt{A_{11}}$ | East DOP |
| $NDOP = \sqrt{A_{22}}$ | North DOP |

**Equation 2-11**

Therefore, the total position error magnitude can be estimated by $\sigma \times PDOP$, the vertical position error by $\sigma \times VDOP$, the horizontal position error by $\sigma \times HDOP$, and so on...The $\sigma$ is the same as in the *Equation 2-9*. However, this is only an approximation, because of the assumption that all satellite pseudorange measurements errors are independent and have the same statistics. In fact the value of $\sigma$ depends on the position of the observed satellite. To compute an expected error on the receiver position, a value of $\sigma$ with a satellite at 20 degrees of elevation from the receiver is used.

Typical modern receivers track 5-12 satellites simultaneously. More satellites produce improved geometry, generally leading to a more accurate single point navigation solution (an exception can occur if the ranging error to the additional satellite is exceptionally poor).

GPS ranging errors are magnified by the range vector differences between the receiver and the satellites. The volume of the shape described by the unit-vectors from the receiver to the satellites used in a position fix is inversely proportional to GDOP (no mathematical relationship between the volume and the GDOP). A poor GDOP represents a large value, and so a small unit vector-volume whereas a good GDOP represents a small value, and so a large unit vector-volume as shown in *Figure 2-9* [33].

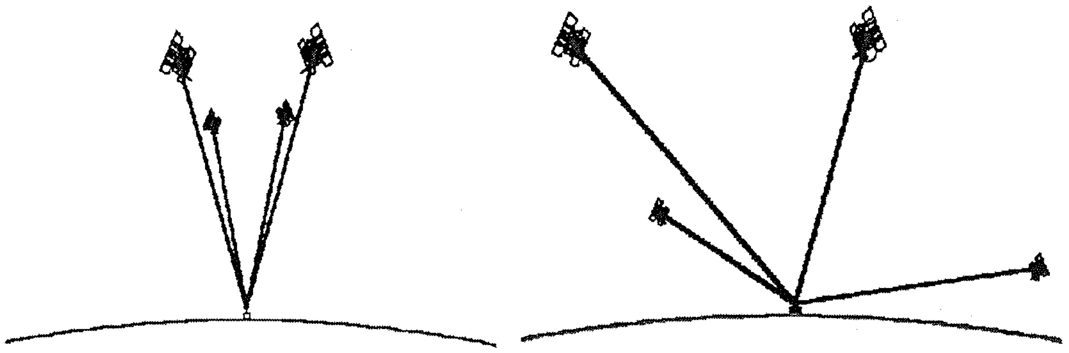


**Figure 2-9- Geometrical dilution of precision**

# 3 Program definition and first simulations

In this part, a general overview of the program developed by the author is given. The program is called DUNSS, which means Dynamic User Navigation Solution Simulator. The definition of DUNSS's aims and organisation is given. The functional requirements, and the design choices to achieve them, are also defined.

## 3.1 Program purpose

The Dynamic User Navigation Solution Simulator (DUNSS) is a program, which has to simulate a satellite navigation receiver, tracking navigation satellites sending deteriorated signals.

DUNSS has to compute the current step estimate using:

- The input parameters chosen by the user.
- The chosen satellite ephemerides provided by STK or by real data files.
- The models chosen by the user (in those proposed by DUNSS), to deteriorate the satellites signal (errors on the ephemerides, on the clock, ionospheric and tropospheric delay, multipath, noise).

DUNSS has to compute the error between the current step estimate and the position given by the equations of the receiver trajectory. It also has to provide the results to the user in a configuration as friendly as possible.

The program is decomposed into the following five main functions or sub-programs:

- A Database Management function, which is in charge of making the interface between the user and the computer, and the acquisition and the storage of the input data.
- The Navigation Solution Computation function, which has to compute the current step estimate from the in view satellites data.
- A function in charge of the satellites data acquisition. These data come from STK, or from real data files (depends on the choice made by the user).
- An Error Computation function to compare the value computed by the Navigation Solution Computation function and the "true" receiver position (given by the equations of the trajectory).
- A display function, to display the input and output data into tables, graphics and maps.

*Figure 3-1* shows the basic organisation between the different functions.

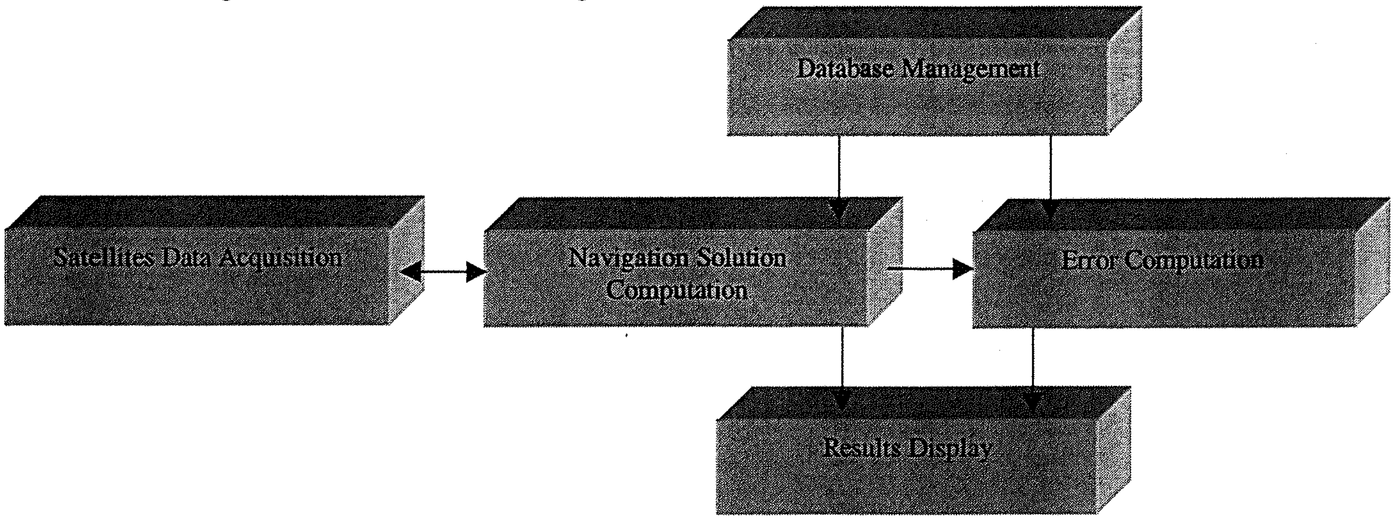


**Figure 3-1- General organisation of DUNSS**

## 3.2 General processing

The main function of DUNSS is the Navigation Solution Computation function, but it requires all four functions to work.

The Database Management function has to request to the user to enter the input parameters (listed below), to store them, and to provide them to the Navigation Solution Computation function.

The Satellite Data Acquisition function is in charge of taking the satellites data and providing them to the Navigation Solution Computation function. If the user had chosen to use data coming from STK, the Satellite Data Acquisition function has to compute an interpolation of the satellite position every second, between two values provided by STK. If the user had chosen to use the real data coming from a GPS receiver, the Satellite Data Acquisition function must be able to read these data in the format they are available.

The Navigation Solution Computation function, at each computation-step, has to:

- Use the satellite positions provided by the Satellite Data Acquisition function.
- With these values, compute which satellites are in view from the receiver.
- Deteriorate the in view satellites signal
- Compute the current step estimate with all these parameters.
- Store the current step estimate, and provide it to the Error Computation function.

The Error Computation function, at each computation, has to:

- Compute the "True" receiver position from the equations provided by the Database Management function.
- Compare it to the current step estimate, and compute the error between the two values.
- Store this error.

The Display function is in charge of processing the input data, the in view satellites data, the receiver state estimate at each step of the simulation, the "True" position, the error between these two values, and the satellites data, with the aim of drawing some curves. It shall store all these data into some files, under a format directly usable by the software Matlab.

The *Figure 3-2* shows the general process during the computation.

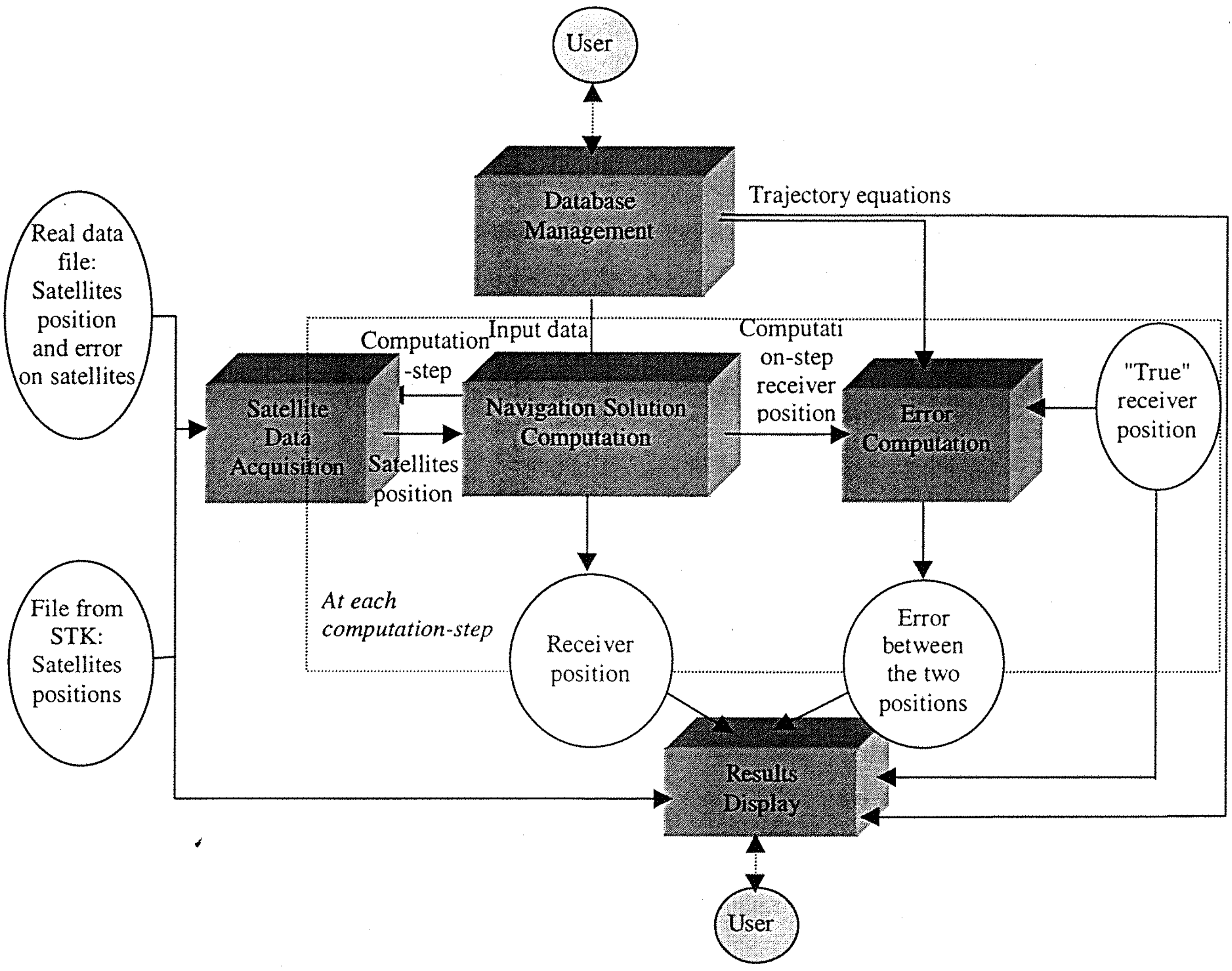

User
Database Management
Trajectory equations
Real data file: Satellites position and error on satellites
Computation-step
Input data
Computation-step receiver position
Satellite Data Acquisition
Navigation Solution Computation
Error Computation
"True" receiver position
Satellites position
File from STK: Satellites positions
At each computation-step
Receiver position
Error between the two positions
Results Display
User

**Figure 3-2- General computation process**

## 3.3 Functional requirement

### 3.3.1 The Database Management function

The Database Management function is in charge of printing on the computer screen questions to ask to the user about the values of the input parameters. It has to store these values and to provide them to the Navigation Solution Computation function.

The input parameters are:

- The duration of the simulation.
- The initial receiver estimated position.
- The initial true position.
- The equations of the receiver trajectory.
- The different contributions of the error budget: error on satellite ephemerides, error on clock, ionospheric and tropospheric delay, multipath, and noise.

### 3.3.2 The Satellite Data Acquisition function

If the user has chosen to use the satellite data coming from STK files, the Satellite Data Acquisition function must:

- Open the STK files.
- Read and store in arrays the satellites state.
- Close the files.

If the user has chosen to use real satellite data, the Satellite Data Acquisition function must:

- Open the real data files (called log_txt and log_mik.ion).
- Read and store in arrays the satellites state.
- Close the files.

In both cases, the Satellite Data Acquisition function stores the satellites data at the beginning of the simulation. If the user has chosen to use some error models (applied to satellites signal), which need some data from the real data files (like alpha and beta coefficients for the Klobuchar ionospheric model [5]), the Satellites Data Acquisition function has to read and store them.

If the user wants to work with the data coming from STK, he chooses the satellites when he launches the STK simulation. He can choose, for example, the GPS or/and Glonass constellation(s).

To make the two simulations compatible:

- The start time of DUNSS simulation must be equal to the start time of the STK simulation, or to the starting time of the real data files.
- The end time of DUNSS simulation must be before the end time of the STK simulation, or before the end time of the real data files.

### 3.3.3 The Navigation Solution Computation function

The Navigation Solution Computation function can be decomposed into the following functions.

#### 3.3.3.1 *Process of the input data*

The Navigation Solution Computation function has to process the input data provided by the Database Management function.

#### 3.3.3.2 *Link with the Satellite Data Acquisition function*

The Navigation Solution Computation function has to process the satellites position provided by the Satellite Data Acquisition function.

#### 3.3.3.3 *Satellites in view*

The Navigation Solution Computation function has to compute which satellites from the chosen constellation are in view from the receiver, by computing their elevation angle.

#### 3.3.3.4 *Deterioration of the satellites signal*

The Navigation Solution Computation function has to deteriorate the signal, coming from the in view satellites, at each acquisition (every second), using the following errors: error on the satellites ephemerides, error on the clock, ionospheric and tropospheric delay, multipath, noise.

To compute these errors, the Navigation Solution Computation function could use two ways:

- Use some budget errors (Gaussian noise).
- Use the models chosen by the user, among the proposed ones, at the beginning of the simulation.

The gaussian noise is supposed to give a good approximation of the errors on the satellite signals: in fact they just give values which are in the expected range. They do not take into account some parameters like receiver environment (for the noise and multipath), or differences between day and night (tropospheric and ionospheric delay).

#### 3.3.3.5 Computation of the receiver position

The Navigation Solution Computation function has to compute the receiver current step estimate. This computation requires several stages:

- Computation of the azimuth and elevation angles for every in view satellites, using the in view satellites positions provided by the Interface function.
- Computation of the G matrix using the Azimuth and elevation angles of all in view satellites.
- Computation of the W matrix using the estimated $\sigma$ of all errors on all in view satellites.
- Computation of the error due to the ephemeris errors, clock errors, ionospheric and tropospheric delay, multipath and noise.
- Computation of residual pseudorange.
- Computation of the least squares method solution.
- Computation of the receiver position.
- Computation of the DOP values.

### 3.3.3.6 Storage and link with the Error Computation function

The Navigation Solution Computation function has to store the computed values (receiver position, DOP, expected range of error…) into a file, and to provide every second the state estimate to the Error Computation function.

### 3.3.3.7 Summary

*Figure 3-3* shows the general process to compute the receiver position.

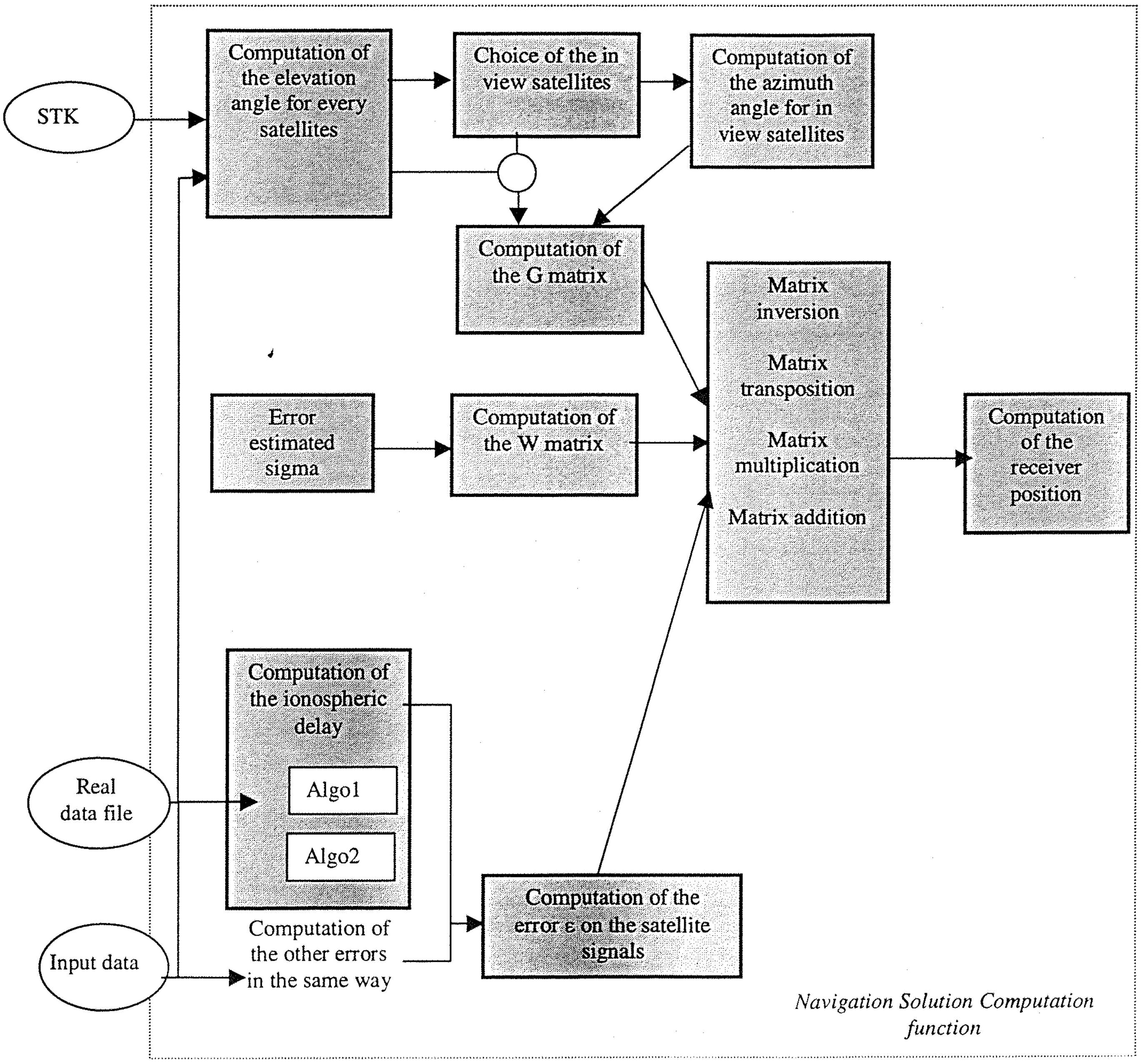

STK
Computation of the elevation angle for every satellites
Choice of the in view satellites
Computation of the azimuth angle for in view satellites
Computation of the G matrix
Matrix inversion
Matrix transposition
Matrix multiplication
Matrix addition
Error estimated sigma
Computation of the W matrix
Computation of the receiver position
Computation of the ionospheric delay
Algo1
Algo2
Real data file
Input data
Computation of the other errors in the same way
Computation of the error ε on the satellite signals
Navigation Solution Computation function

**Figure 3-3- Main algorithm organisation**

### 3.3.4 The Error Computation function

At each computation step, the Error Computation function receives the computed receiver position from the Navigation Solution Computation function.
Therefore, it is in charge of computing the true receiver position at this step, given by its trajectory equations, and computing the error between the two values.
It has also to store the "true" receiver position and the error.

### 3.3.5 The Display function

The Display function is in charge of processing the output data and the input data: computed receiver position, "True" receiver position, error between the two values, errors on the satellites signal, expected error, etc....
The aim is to draw some curves using Matlab:

- the error on satellites signal as a function of the time
- the receiver position error (on the three co-ordinates, and the global error) as a function of the time
- the receiver position (three co-ordinates) as a function of the time
- the expected error as a function of the time
- the number of in view satellite as a function of the time
- the azimuth and elevation angles of the in view satellites as a function of the time

## 3.4 Program design

To satisfy to the functional requirements, the choices of program design are described function by function.

### 3.4.1 Database Management function

To enter the receiver initial position, the user can choose between two types of coordinates: The cartesian coordinates (X, Y, Z), or the polar coordinates ($\rho$, $\lambda$, $\varphi$). Both are in the Earth centred WGS84 referential (1984 World Geodetic System).

To enter the receiver trajectory equation, the user chooses between a static or dynamic receiver. If the receiver is dynamic, the user writes a parametric equation (in function of the time, in second) for each coordinate (in function of the previous choice).

### 3.4.2 The Satellite Data Acquisition function

The Satellite Data Acquisition function is formatted to open the STK files under the name "gps_2-i.sa" with i the number of the satellite. It is formatted to open the real data files under the name "log.txt" and "log_mik.ion". If the user wants to open some other files, he has to change the code.

To avoid storing some huge STK files, the choice is made to use some STK files with a time step of 1 minute. However, the GPS, or other satellite navigation systems, take the satellites data every second. To simulate this, the Satellite Data Acquisition function has to be able to compute an interpolation of the satellite positions every second. A linear interpolation of the satellite position between two minutes is chosen.

### 3.4.3 Navigation Solution Computation function

The cartesian coordinates in WGS 84 (Logdson, 1986 [1]) are chosen to carry out all the computations. All the variables are in scientific international units.

Further computational details are given in *Annexe III.*

### 3.4.4 Display function

The Display function stores all the output data under a format easily usable by the software Matlab. Matlab is used to draw the curves of results.

## 3.5 Program validation

### 3.5.1 Navigation Solution Computation function validation

#### 3.5.1.1 Reference example

To validate the main algorithm of the program (least squares method), the numerical example given by P. Axelrad and R.G. Brown [3] is used. This example provides the receiver position computed with the algorithm developed in part 2.2, and uses the real GPS satellite position during the period of the simulation. The set of results provided by DUNSS is compared with the data coming from this example.

The initial parameters of this example are in cartesian coordinates in referential WGS 84 (Logdson, 1986 [1]):

Initial estimated position:

X_ini=[6 377 000.0 3 000.0 4 000.0 0.0]$^T$ meters.

Initial true position:

X_true=[6 378 137.0 0.0 0.0 85 000.0]$^T$ meters.

The receiver is static.

Seven satellites are in view from the receiver (with a mask angle of 10 degrees). The positions of the satellites in view are:

| satellite | X position (meter) | Y position (meter) | Z position (meter) |
|---|---|---|---|
| SV 01 | 22 808 160.9 | -12 005 866.6 | -6 609 526.5 |
| SV 02 | 21 141 179.5 | -2 355 056.3 | -15 985 716.1 |
| SV 08 | 20 438 959.3 | -4 238 967.1 | 16 502 090.2 |
| SV 14 | 18 432 296.2 | -18 613 382.5 | -4 672 400.8 |
| SV 17 | 21 772 117.8 | 13 773 269.7 | 6 656 636.4 |
| SV 23 | 15 561 523.9 | 3 469 098.6 | -21 303 596.2 |
| SV 24 | 13 773 316.6 | 15 929 331.4 | -16 266 254.4 |

The global error put on the satellites signal is:

| satellite | SV 01 | SV 02 | SV 08 | SV 14 | SV 17 | SV 23 | SV 24 |
|---|---|---|---|---|---|---|---|
| Error (meter) | 7.79 | 9.11 | 5.29 | -8.77 | -15.12 | 5.56 | 2.05 |

*Note:* The values of these errors are not given in the example but can be computed by making the difference between the measured pseudorange given in the article [3], and the pseudorange computed by the formula:

$$\rho_{k\_estimated} = \left| r_k - r_{user\_estimated} \right| + cb_{user\_estimated}$$

### *3.5.1.2 Validation of the main algorithm*

Using the values of the example described in the *3.5.1.1* above, the program computes exactly the same results as those given in [3].

First iteration:

$\Delta\bar{x} = [-1131.8 \quad 2996.8 \quad 3993.1 \quad -84996.4]^T$ meters.

$x_{estimated}[1] = [6378131.8 \quad 3.2 \quad 6.9 \quad 84996.4]^T$ meters.

And the error between the two positions is:

$\Delta x[1] = [-5.2 \quad 3.2 \quad 6.9 \quad -3.6]^T$ meters.

Second iteration:

$\Delta\bar{x} = [0.3 \quad -0.1 \quad -0.2 \quad 0.6]^T$ meters.

$x_{estimated}[2] = [6378131.5 \quad 3.3 \quad 7.1 \quad 84995.8]^T$ meters.

And the error between the two positions is: $\Delta x[2] = [-5.5 \quad 3.3 \quad 7.1 \quad -4.2]^T$ meters.

The algorithm confirms the values given by the example. This test was used to validate the program.

This part of the program was also validated by checking the results on other examples and compared with the results given by a pocket calculator.

### 3.5.1.3 Validation of the satellites errors computation function

The error function has to compute the error on the satellite signals. Before using more accurate models of errors (see *part 4*), the program assumes that these errors are white noise, and are computed using a gaussian function. The gaussian distribution is the same for the six types of error. Each error is then computed by weighing the gaussian distribution with a coefficient. These six coefficients are chosen depending on the relative weight of each type of error.

The validation of this function was made by checking the values of the error provided: rough estimated, variations at each computation. *Figure 3-4* shows the distribution (1000 computations) of the error on the clock: as expected, a gaussian distribution is obtained. As the results were satisfactory, the error function is validated.

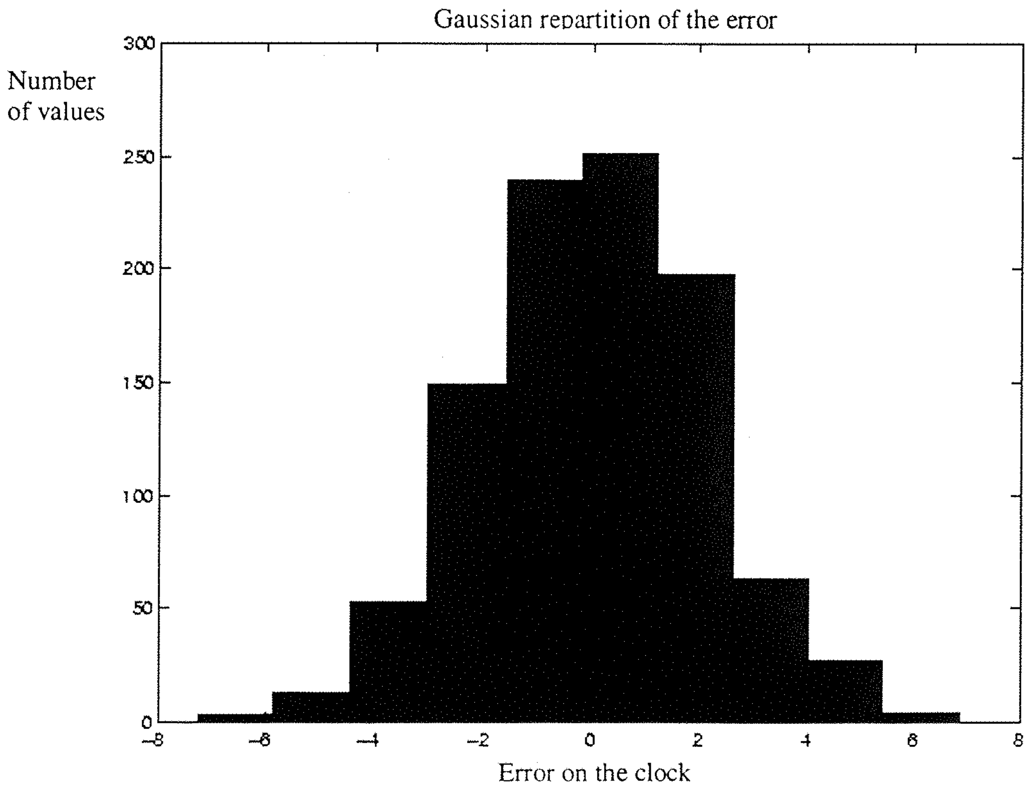


**Figure 3-4- Clock error distribution**

### 3.5.2 Other functions validation

The validation of the other function (Database management function, Satellite Data Acquisition function, Error computation function and Display function) was performed by printing the results and comparing them with the results provided by the user, STK, real data file, or an independent way of computation.

## 3.6 Parametric study

### 3.6.1 Number of iteration

At each acquisition (one acquisition every second), the program computes several times the estimated position of the receiver, with the same satellites data. The choice of the number of iterations can be optimised.

To study the influence of the number of iterations, and to choose the optimal value, the following experimentation was carried out: Global error was compared with two and four iterations per acquisition.

#### *3.6.1.1 First example*

The satellites positions come from STK, using the GPS constellation on 01/07/1998 between 0h00m00s and 0h10m00s with a time step of 1 minute.

*Figure 3-5* shows the global error (equal to $\sqrt{(x_{True} - x_{estimated})^2 + (y_{True} - y_{estimated})^2 + (z_{True} - z_{estimated})^2}$ ) on the receiver position when the initial error is the same as in the example described in the *part 3.5.1.1*(error on the satellites signals computed by white noise). The variations on the global error show that the receiver estimated is not improved between the first iteration and the second one.

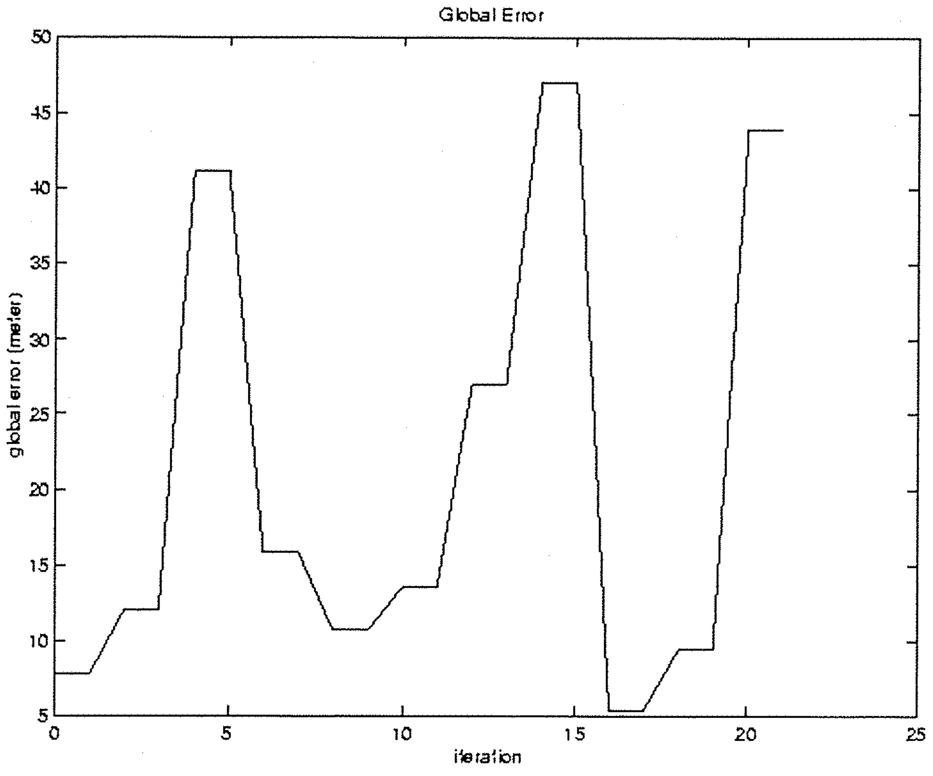


**Figure 3-5- Error on receiver position. Algorithm iterated twice per second.**

### 3.6.1.2 *Second example*

The *Figure 3-6* shows the global error on the receiver position when the initial error between the estimated and the true receiver position is more important. Here, $x_{true} = [6378137.0 \quad 0.0 \quad 0.0 \quad 85000.0]^T$ as in the first case, but $x_{initial\_estimated} = [6370000.0 \quad 30000.0 \quad 320673.6 \quad 0.0]^T$. The initial error is then more important. The error on the satellites signals is the same as in the first case.

The *Table 3-1* summarises the values of the global error drawn on the graph. It can be seen that except for the first computation at the second 1, the difference between the first and the second iterations is negligible.

| | iteration 1 | iteration 2 |
|---|---|---|
| second 1 | 709.413574 | 7.790563 |
| second 2 | 12.003731 | 12.003726 |
| second 3 | 41.104912 | 41.104969 |
| second 4 | 15.872428 | 15.872420 |
| second 5 | 10.682732 | 10.682724 |
| second 6 | 13.581177 | 13.581178 |
| second 7 | 26.990475 | 26.990490 |
| second 8 | 47.094810 | 47.094791 |
| second 9 | 5.408005 | 5.407994 |
| second 10 | 9.448669 | 9.448670 |
| second 11 | 43.957180 | 43.957230 |

**Table 3-1-error global on receiver position (2 iterations) in meters**

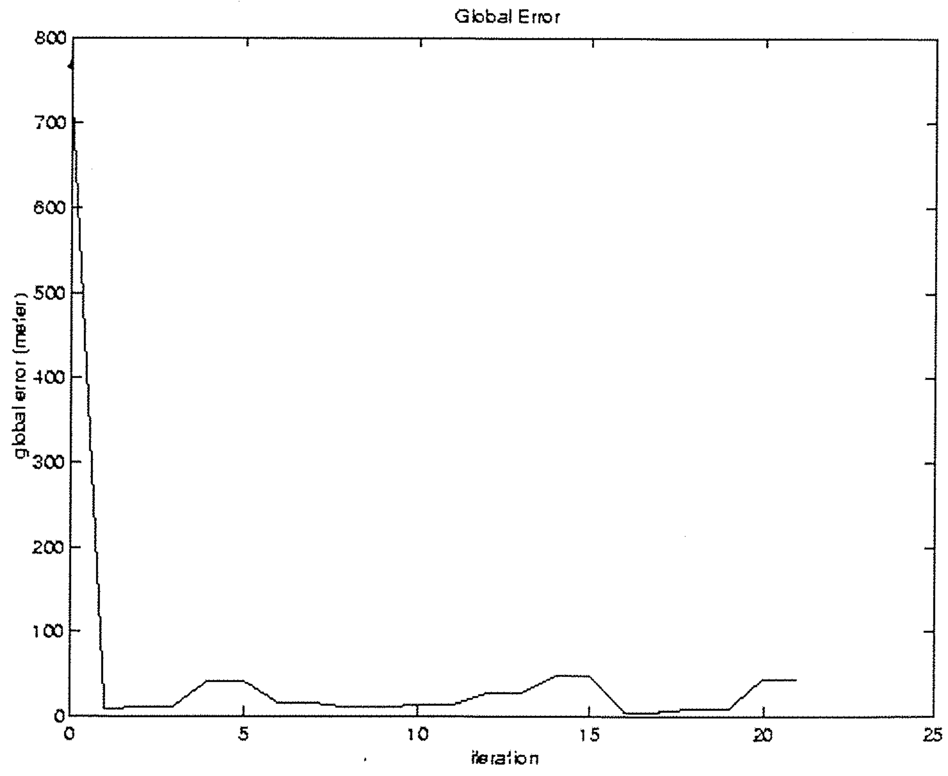


**Figure 3-6- Error on receiver position. Algorithm iterated twice per second. High initial error.**

The *Table 3-2*, and *Figure 3-7* show the same results as the preview example, but with four iterations. It can be seen that the iterations 3 and 4 do not improve the accuracy on the estimated receiver position, even for the first computation step.

| | iteration 1 | iteration 2 | iteration 3 | iteration 4 |
|---|---|---|---|---|
| second 1 | 709.413574 | 7.790563 | 7.790563 | 7.790563 |
| second 2 | 12.003731 | 12.003726 | 12.003726 | 12.003726 |

**Table 3-2- error global on receiver position (4 iterations)**

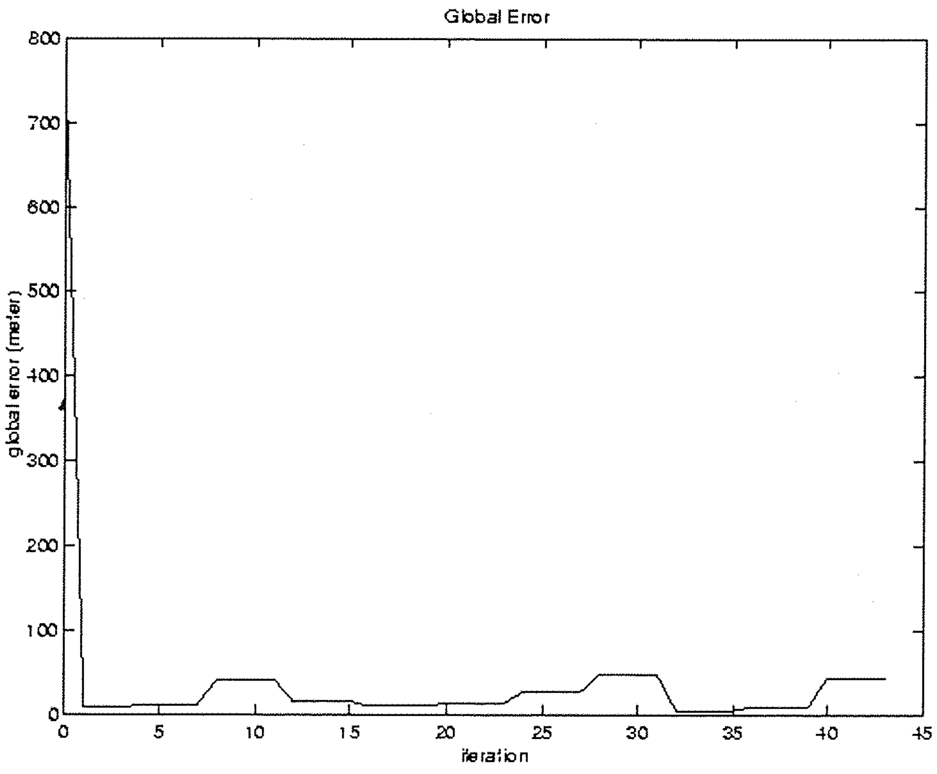


**Figure 3-7- Error on receiver position. Algorithm iterated four times per second**

### 3.6.1.3 Conclusion

These results can be explained as follows:

At the first iteration, the error between the two receiver positions is important, then $\Delta\rho$ (difference between the two pseudoranges) is important, and $\Delta x_{estimated}$ is also important.

But since the estimated position of the receiver is close to the true position (less than fifty meters), the error between the two receiver positions is small, then $\Delta\rho$ is small, and $\Delta x_{estimated}$ is also small. At the following iteration, the estimated receiver position is the same as at the previous step, the matrix G does not change, and so $\Delta x_{estimated}$ is still small. Then the estimated receiver position is the same as at the previous step, and so on...
This is summarised in the *Figure 3-8*.

In conclusion, if the receiver has enough satellites in view, and if the initial error on the receiver position is not huge, two iterations every second are sufficient. To make some experiments in a borderline case, the program user can change the number of iterations. But on no account, can the accuracy of the result be improved by increasing the number of iterations.

*Note:* The user of a GPS receiver cannot modify the number of iterations. The previous simulations do not represent a real case, but are done to study the influence of the number of iteration per second.

| | | | |
|---|---|---|---|
| second 1 | iteration 1 | Error matrix computation | |
| | | Computation of G | G[1][1] |
| | | Computation of $\Delta\rho$ | $\Delta\rho$ is important |
| | | Computation of $\Delta x_{estimated}$ | $\Delta x_{estimated}$ important |
| | | Computation of $x_{estimated}$ | $x_{estimated}[1][1] \neq x_{estimated}[0]$ |
| | iteration 2 | Computation of G | $G[1][2] \neq G[1][1]$ |
| | | Computation of $\Delta\rho$ | $\Delta\rho$ is small |
| | | Computation of $\Delta x_{estimated}$ | $\Delta x_{estimated}[1][2] << \Delta x_{estimated}[1][1]$ |
| | | Computation of $x_{estimated}$ | $\lvert x_{estimated}[1][2] - x_{True} \rvert$<50m |
| | iteration 3 | Computation of G | $G[1][3] \approx G[1][2]$ |
| | | Computation of $\Delta\rho$ | $\Delta\rho$ is small |
| | | Computation of $\Delta x_{estimated}$ | $\Delta x_{estimated}[1][3]$ small (<<1m) |
| | | Computation of $x_{estimated}$ | $\lvert x_{estimated}[1][3]-x_{estimated}[1][2]\rvert \approx 0$ |
| ... | ... | ... | ... |
| ... | ... | ... | ... |
| second k | iteration 1 | Error matrix computation | $Error[k] \neq Error[k-1]$ |
| | | Computation of G | $G[k][1] \approx G[k-1][last\_ite]$ |
| | | Computation of $\Delta\rho$ | $\Delta\rho$ small, $\neq \Delta\rho[k-1][last\_ite]$ |
| | | Computation of $\Delta x_{estimated}$ | $\Delta x_{estimated}$ small, > 10m |
| | | Computation of $x_{estimated}$ | $x_{estimated}[k][1] \neq x_{estimated}[k][last\_ite]$, $\lvert x_{estimated}[k][1]-x_{True}\rvert$<50m |
| | iteration 2 | Computation of G | $G[k][2] \neq G[k][1]$ |
| | | Computation of $\Delta\rho$ | $\Delta\rho$ is small |
| | | Computation of $\Delta x_{estimated}$ | $\Delta x_{estimated}[k][2]$ < 1m |
| | | Computation of $x_{estimated}$ | $\lvert x_{estimated}[k][2]-x_{estimated}[k][1]\rvert$<1m |
| | ... | ... | ... |
| | | | |
| | | | |

**Figure 3-8- General computation process**

### 3.6.2 Influence of the initial error

As shown in *part 3.6.1*, the initial error on the receiver position can only influence the computation for the two first steps. Afterwards, the program succeeds in computing a position very close to the true one.

### 3.6.3 Influence of the error on satellite signals

#### *3.6.3.1 Generalities*

In the first instance, the error included on the satellite signal is computed as a white noise. This way of computation provides an error on satellite signal, which is in the expected range of error on the satellite signal, but which varies unrealistically. Actually, the six types of errors have the following behaviour as functions of the time:

- Nearly constant (very slow variations) for tropospheric delay, clock and ephemeris error, even if the ephemeris error contains also some faster varaitions.
- Sinusoidal for multipath error due the periodic repetition of the signal.
- Fast variations for the ionospheric delay.
- Fast variations that can be modelled with white noise for the receiver noise.

The aim of the study, more than simulating a real case, is to study the behaviour of the simulator, and the relative importance of each type of error.

The error range is determined by some method of measurements: dual frequency receiver for ionospheric delay, measurement campaign for ephemeris error etc...
The *Table 3-3* gives the usual range of values and a larger range for each type of error (data provided by Alcatel Space Industries Toulouse).
The errors on satellite signal are in the usual range when the navigation takes place with calm conditions (low solar activity, favourable environment...). The errors on

satellite signal are in the larger range of value when the navigation conditions are not extreme (e.g. higher solar activity, but not solar storm).

Therefore, the values given in *Table 3-3* do not mean that the errors on satellite signal are always in the range. It just means that in most of the cases, they are within this range.

| Error | Usual range (meter) | Larger range (meter) | Observation |
|---|---|---|---|
| Ephemeris | ±5 | ±100 | For GPS satellites from block II |
| Clock | ±15 | ±30 | |
| Ionospheric* | 0-5 | 0-100 | Highly dependent on the solar activity |
| Tropospheric* | 0-1.5 | 0-20 | Stable. Dependent on the water vapour in the troposphere |
| Multipath* | ±2.5 | ±100 | Highly dependent on the receiver environment and its technology |
| Noise | ±5 | ±5 | |

*: With an elevation angle higher than 10 degrees

**Table 3-3- Range of error on satellite signal**

In fact, with the exception of ionospheric delay, it is virtually impossible to determine the exact error in satellite signal in real time. All these models provide only an estimation of this error. In the case of white noise, it is a merely raw estimation: the errors provided do not follow the influence of external parameters such as:

- Difference between day and night for ionospheric delay, but also for ephemeris error (the Earth reflected radiations induce a drag).
- Solar activity for ionospheric delay, for ephemeris error.
- Water vapour content in the troposphere for tropospheric delay.
- User environment for multipath and noise.

The equations used to compute the six types of errors with white noise are:

$$error\_ephemeris = \sigma_{ephemeris} \times random\_value$$
$$error\_clock = \sigma_{clock} \times random\_value$$
$$error\_ionospheric = \sigma_{ionospheric} \times random\_value \div \sin(El_i)$$
$$error\_tropospheric = \sigma_{tropospheric} \times random\_value \div \sin(El_i)$$
$$error\_multipath = \sigma_{multipath} \times random\_value \div \tan(El_i)$$
$$error\_noise = \sigma_{noise} \times random\_value$$

**Equation 3-1**

The random value represents a numeric value taken randomly within a normalised gaussian distribution (values between 0 and 1). The gaussian distribution used is the same for the six errors. The units of these errors and sigma are meters. These values represent the $\Delta\varepsilon_{\rho i}$ of the least squares algorithm (*Equation 2-4*).

In this part, the errors have a gaussian distribution, which is unrealistic. Most of the errors on satellite signal vary slowly as a function of the time (ephemeris, clock, tropospheric, ionospheric). The multipath error has sinusoidal variations, and only the noise on the receiver can be realistically modelled with white noise. The aim of this set of simulations is not to simulate a realistic case, but to do a parametric study of the errors on the satellite signal. The values of sigma were chosen such that the errors on satellite signals may be in a realistic range of values. These equations are used to calculate the values of errors, which change randomly at each computation, in a range that depends on the values of σ. In fact, in DUNSS, in order to be able to re-play exactly the same simulation, the random variable was fixed. Therefore, for the same period of simulation, using the same satellites, the errors put into their signals will be the same.
These errors can be separated into two sorts of errors: the errors, which depend on the satellite position (ionospheric and tropospheric delay, multipath), and the error independent of the satellite position (ephemeris, clock and noise errors).
The following tables and graphics describe the influence of the values of σ on the error put on the satellite signals. A simulation was started on the 30th of April 1999, for 100 seconds. The positions of these satellites were real signals obtained using a GPS receiver. The receiver was located at Alcatel Space Industries in Toulouse (France).

The following values are given in cartesian coordinates in WGS 84 reference.
Initial estimated position of the receiver:
x=4628000.000 meters, y=113000.000 meters, z=4372000.000 meters, bias=0 second.

Initial true position of the receiver:
x=4629365.4041 meters, y=112099.4237 meters, z=4371618.9541 meters, bias=0.00028352948 second.

The receiver is static during the simulation.

Nine satellites are in view from the receiver position. For this test only the error on satellite signal number 1 (high elevation angle: 77.781 degrees), 8 (small elevation angle: 0.371 degrees), 9 (average elevation angle: 41.544 degrees) are considered. We do not use a mask angle for this simulation, which enable us to use the satellite number 8. Satellite number 9 is used as a reference one. At each simulation, the results of the error on the satellite signal, and the global error $\sqrt{(x_{True}-x_{estimated})^2+(y_{True}-y_{estimated})^2+(z_{True}-z_{estimated})^2}$ on the receiver position are presented. The figures showing the global error on the receiver position also shows a stippled line. This stippled curve is the expected range of error (68.27 % of the error shall be between the two curves) equal to $GDOP \times uere$. $uere$ is the total sigma for a satellite at 20 degrees of elevation from the receiver:

$$uere = \sigma_{ephemeris} + \sigma_{clock} + \sigma_{noise} + \sigma_{iono} \div \sin(20) + \sigma_{tropo} \div \sin(20) + \sigma_{mupath} \div \tan(20)$$

The expected error represents the error that the receiver would have on its position with only satellites located at 20 degrees of elevation.

In the following sections, several simulations are described, using different values of σ, in order to see the importance and the influence of each type of error on the receiver position. The S/A degradation can reach 300 meters of error on the satellite signal. To have realistic values of error, the total error on satellite signal is kept lower than this value.

### 3.6.3.2 First simulation

In the first simulation, the values of sigma are chosen in order to give the typical relative importance of each type of error compared with the others, and to have the numeric values of error in the range given in *Table 3-3*.

| error | ephemeris | clock | iono | tropo | mupath | noise |
|---|---|---|---|---|---|---|
| σ (meter) | 2.1 | 2.1 | 0.7 | 0.15 | 0.3 | 0.5 |

**Table 3-4-Usual values of σ**

With satellite number 1, we can see in the *Figure 3-9* and *Figure 3-10* that the total error on the satellite signal follows the ephemeris and clock error. These errors have the greatest σ, and the high elevation angle of the satellite masks the error dependent from the satellite position.

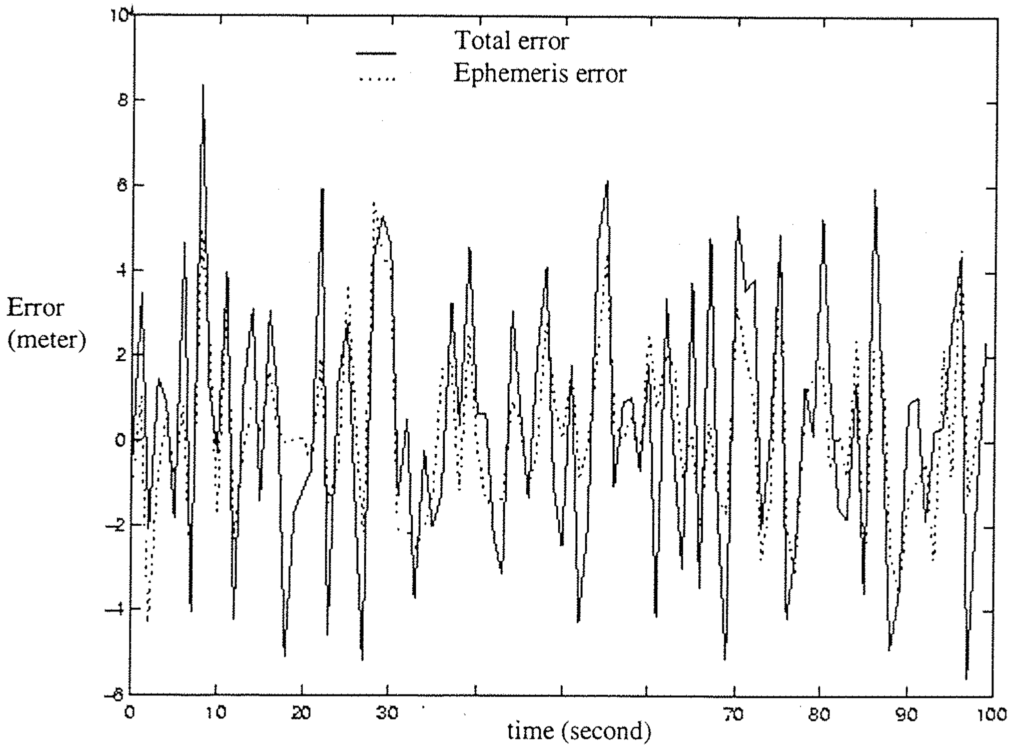


**Figure 3-9- Ephemeris and total error on satellite number 1**

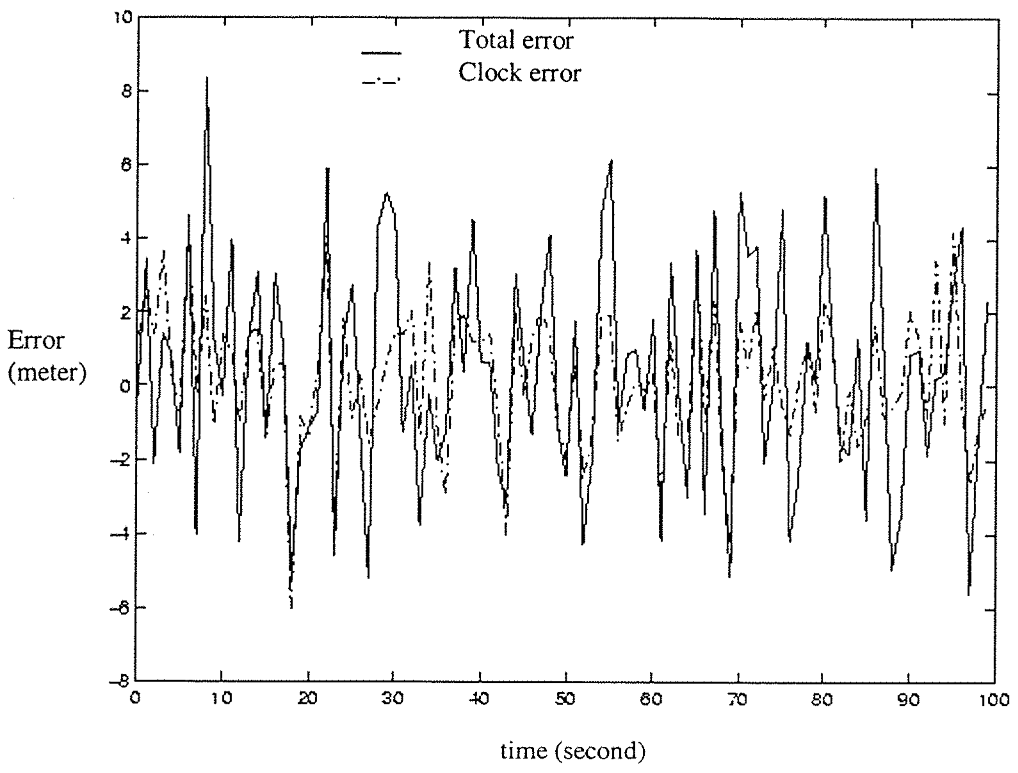


**Figure 3-10- Clock and total error on satellite number 1**

Unlike satellite number 1, the error on satellite number 8 is mainly influenced by the ionospheric delay (see *Figure 3-11*): The small elevation angle increases the weight of the errors dependent on the satellite elevation angle, and masks the influence of the independent errors. It can also be noted that the total error on the satellite signal is more significant than for the satellites number 1 and 9 (see below): the small elevation angle magnifies the error so much, that the total error is between ten and twenty times more important than for a satellite with a lower elevation angle. This is why a GPS receiver does not use the satellite signal below a minimum elevation angle (typically taken between 5 and 10 degrees).

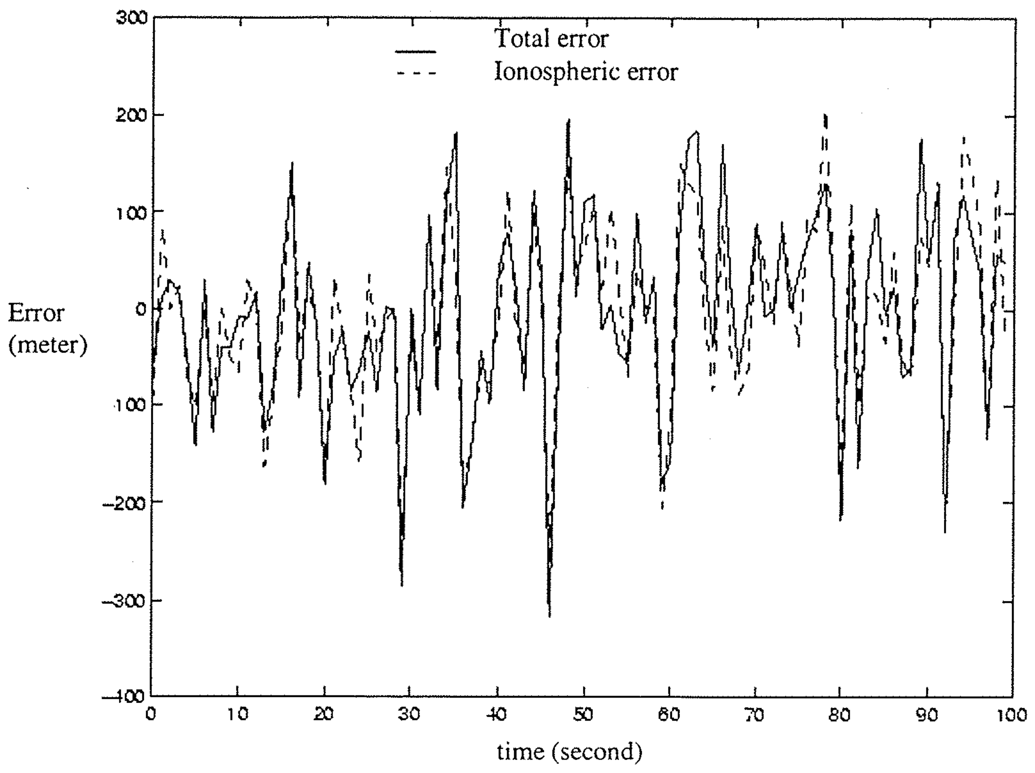
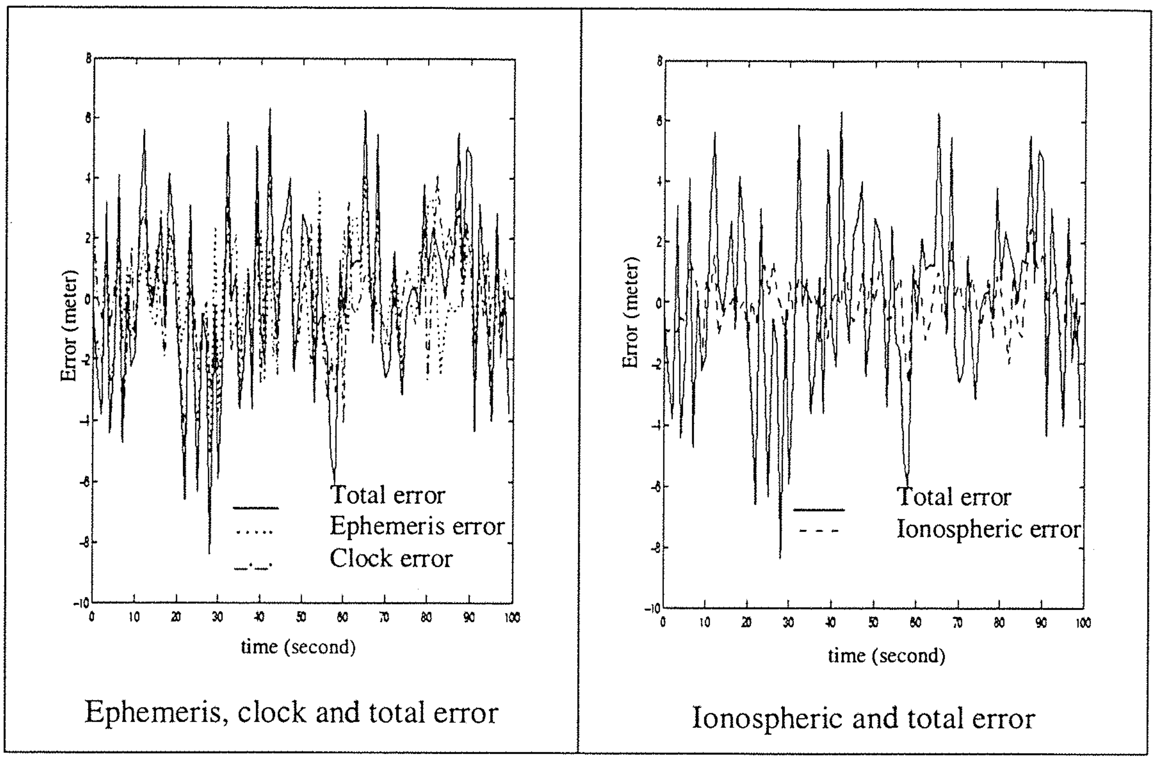


**Figure 3-11- Ionospheric and total error on satellite number 8**

For satellite number 9, the two graphics of *Figure 3-12* show that the main errors on the satellite signal are the ephemeris and clock error (largest sigma), but the ionospheric error is also important.



Ephemeris, clock and total error

Ionospheric and total error

**Figure 3-12- Error on satellite number 9**

With these values of σ, which give an error on the satellites signals in the common range of values, the error on the receiver position is as shown in *Figure 3-13*.

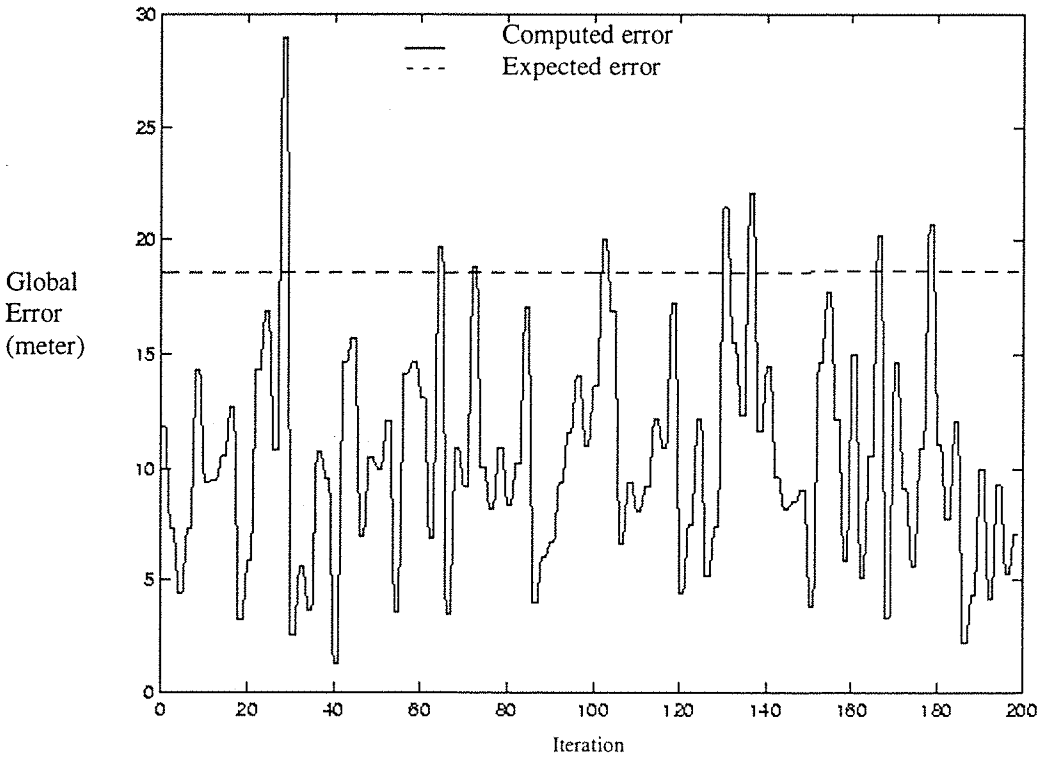


**Figure 3-13- Error on receiver position**

### *3.6.3.3 Second simulation*

In the second simulation we give more importance to one of the errors independent of the satellites elevation angle. We choose to select the ephemeris error, but the results would have been similar if we had chosen the clock or noise error.

| error | ephemeris | clock | iono | tropo | mupath | noise |
|---|---|---|---|---|---|---|
| $\sigma$ (meter) | 21.0 | 2.1 | 0.7 | 0.15 | 0.3 | 0.5 |

**Table 3-5- Sigma ephemeris ten times more important**

It is shown in the following graphics that the ephemeris error is still within the larger range of value given in *Table 3-3*.

As can be seen on *Figure 3-14*, satellite number 1 follows the ephemeris error completely, except when this one is weak (around acquisition 20) and then does not mask the other types of error. The total error is now nearly ten times greater than in the first simulation.

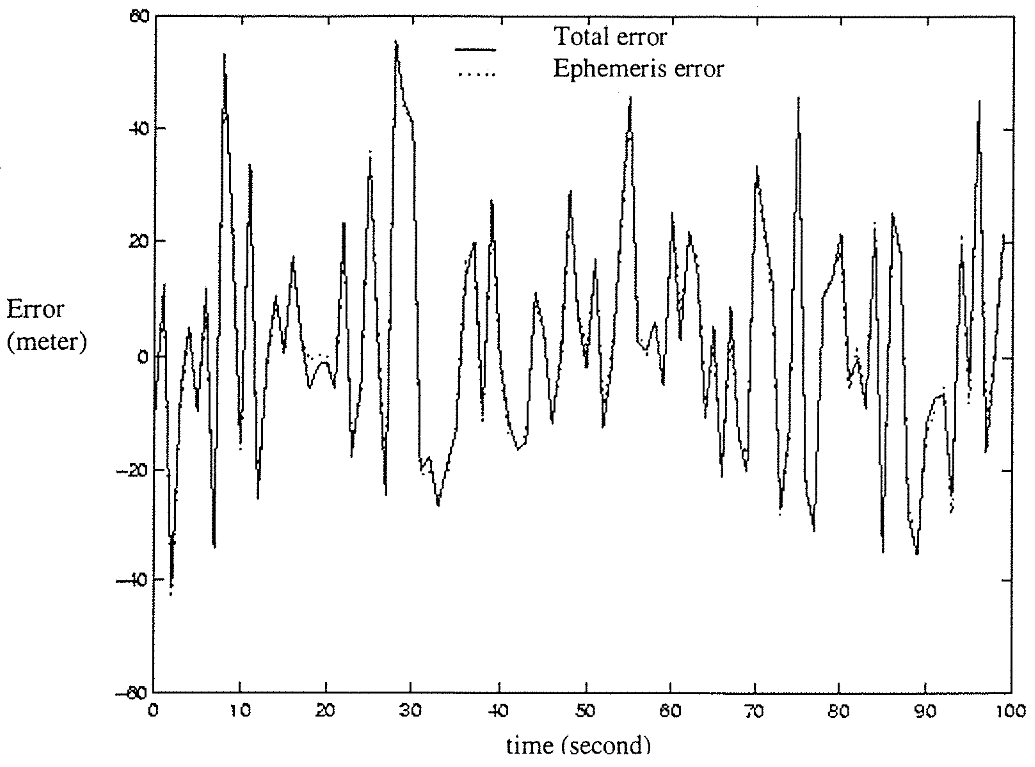


**Figure 3-14- Ephemeris and total error on satellite number 1**

In the *Figure 3-15*, it is shown that the ephemeris error can not completely mask the ionospheric delay which is still the most influential error for the satellite number 8. The total error has not increased like satellite number 1.

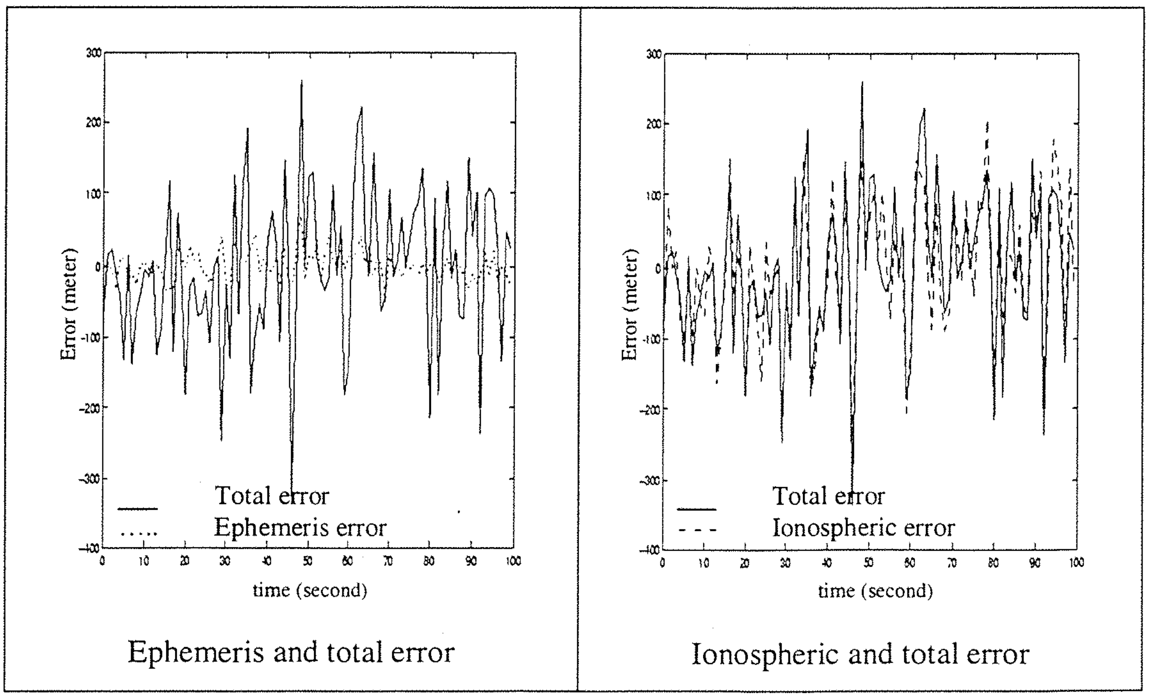


**Figure 3-15- Error on satellite number 8**

Satellite number 9 now mainly follows the ephemeris error. The total error increases compared with the first simulation, but is less than for satellite number 1.

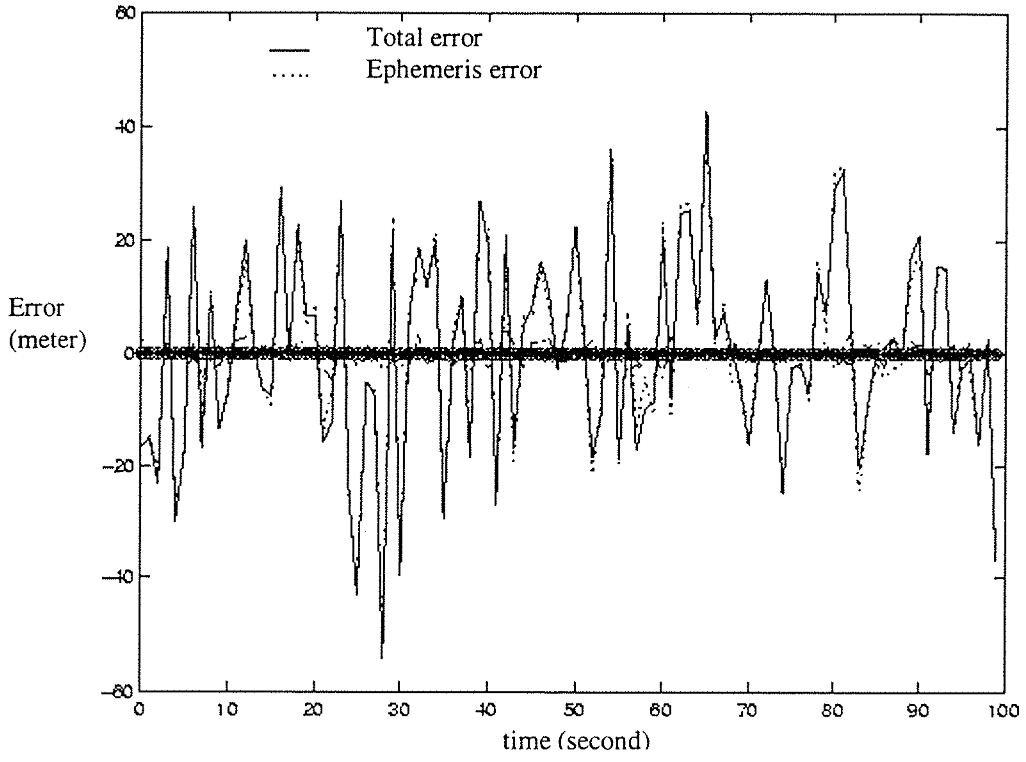


**Figure 3-16- Error on satellite number 9**

The global error on the receiver position increases compared with the first simulation (see *Figure 3-17*), but the expected error increase in the same rough estimate. Therefore, 68.27 % of the error is still to be under the expected error.

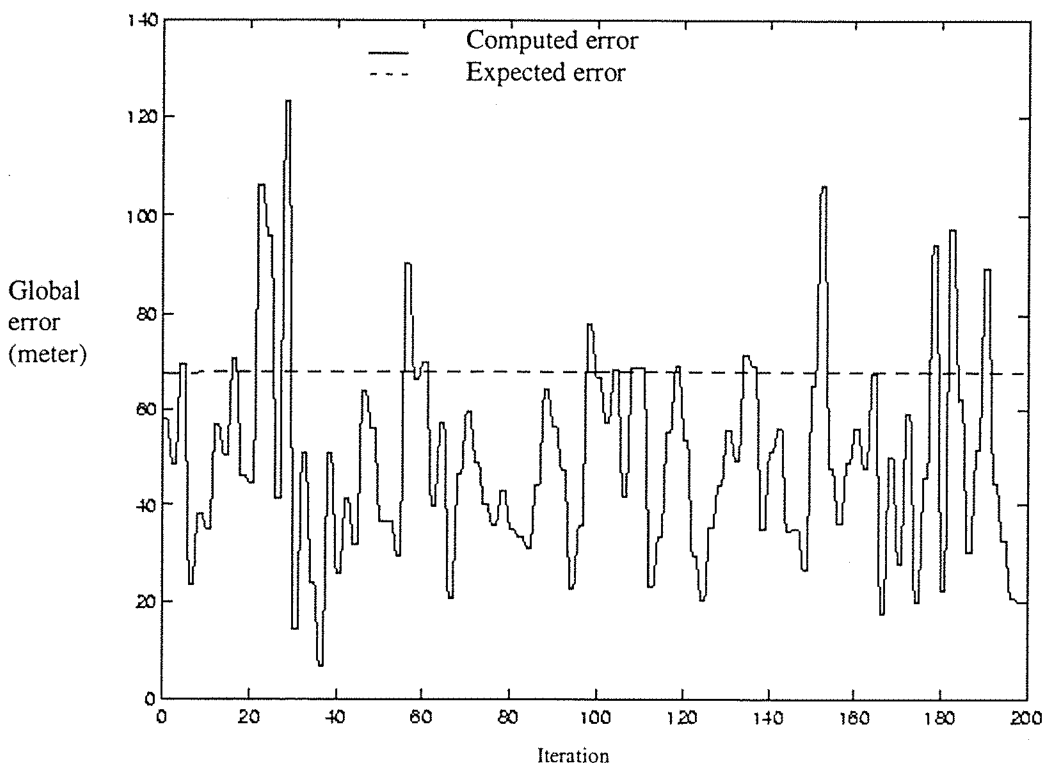


**Figure 3-17- Error on receiver position**

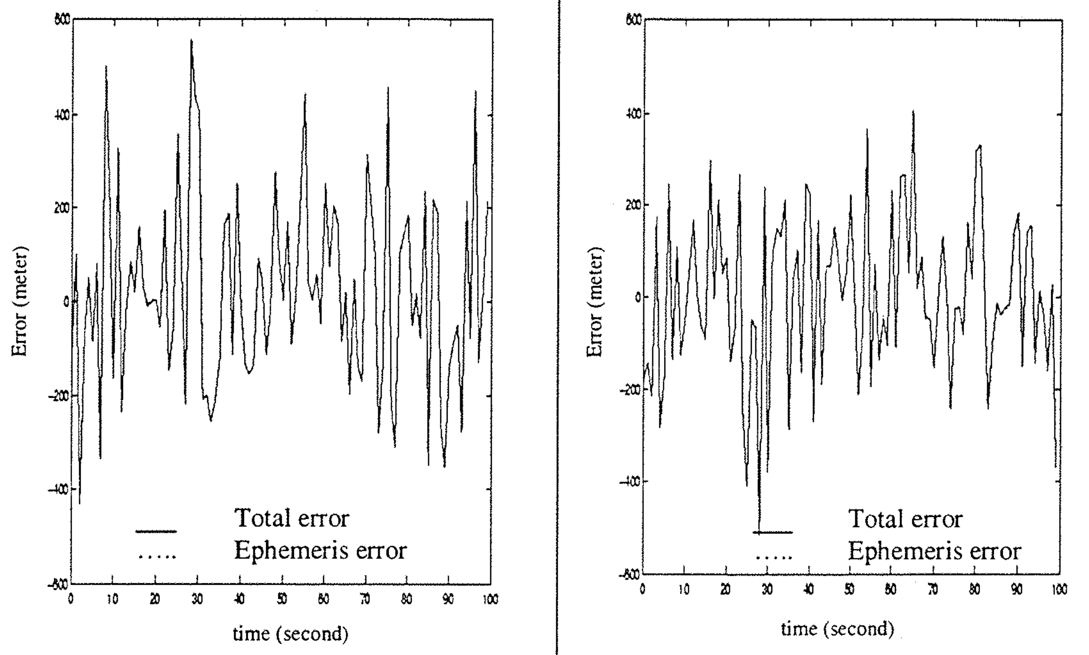
### 3.6.3.4 *Third simulation*

In the third simulation, we now give hundred times more importance to the ephemeris error than in the first simulation. The ephemeris error is then greater than the maximum value given in *Table 3-3*. The simulation is a borderline case, theoretically, it could happen with a very low statistical probability. It is more dedicated to study the influence on the receiver position of huge error, than to represent a real case.

| error | ephemeris | clock | iono | tropo | mupath | noise |
|---|---|---|---|---|---|---|
| σ (meter) | 210.0 | 2.1 | 0.7 | 0.15 | 0.3 | 0.5 |

**Table 3-6- Sigma ephemeris one hundred times more important**

In *Figure 3-18*, we can see that the total error on satellites number 1 and 9 follows completely the ephemeris error (the two curves are superposed). This total error is nearly one hundred times more important for satellite number 1 than in the first simulation. It is less important for satellite number 9.



Ephemeris and total error on satellite number 1

Ephemeris and total error on satellite number 9

**Figure 3-18- Error on satellite number 1 and 9**

In *Figure 3-19*, it is shown that even though the ephemeris error is more influential now, the ionospheric delay is not completely masked in the total error on the satellite number 8.

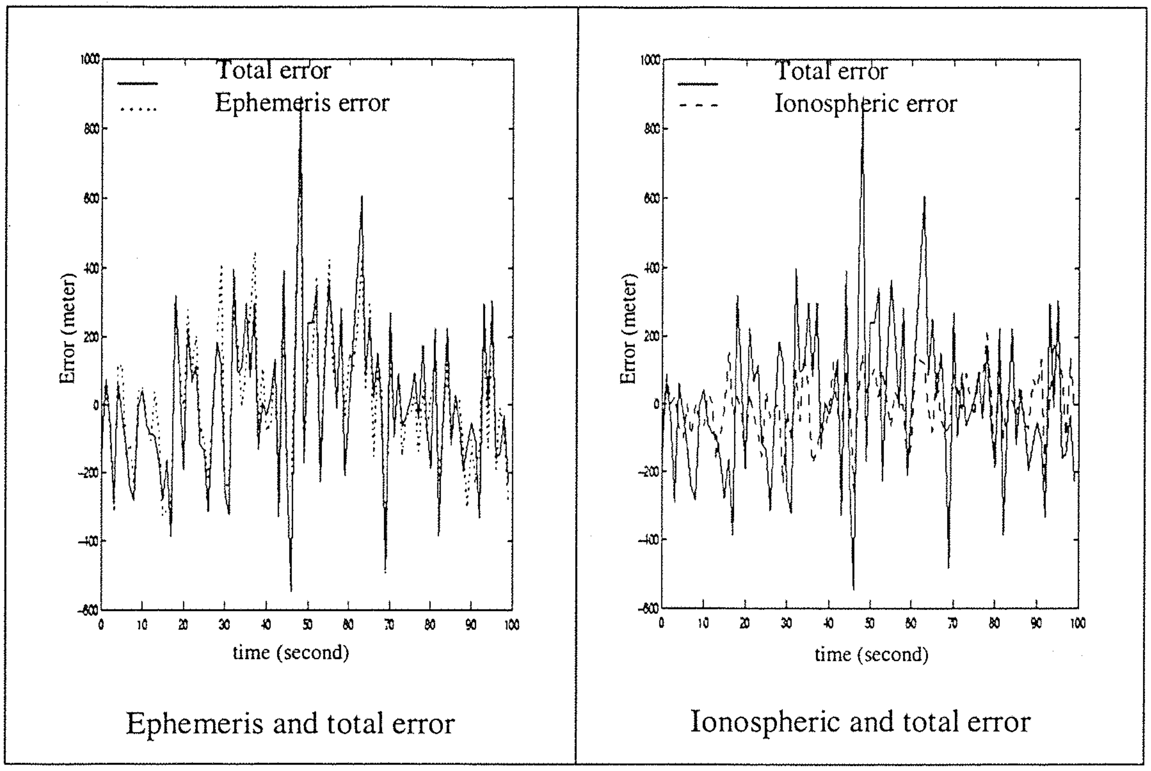


**Figure 3-19- Error on satellite number 8**

We can see in *Figure 3-20* that the global error on the receiver position increases up to one kilometre. The expected error increases in the same rough estimate.

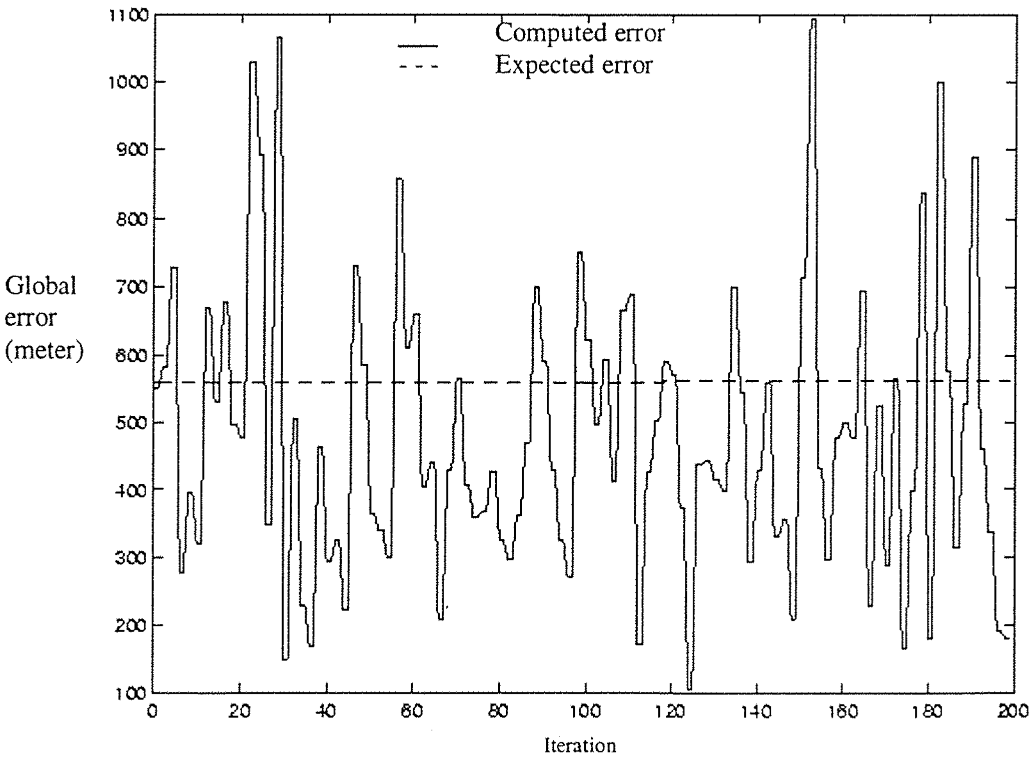


**Figure 3-20- Error on global receiver position**

### 3.6.3.5 Fourth simulation

In the fourth simulation, we give ten times more importance to the ionospheric error than in the first simulation.

| error | ephemeris | clock | iono | tropo | mupath | noise |
|---|---|---|---|---|---|---|
| σ (meter) | 2.1 | 2.1 | 7.0 | 0.15 | 0.3 | 0.5 |

**Table 3-7- Sigma ionospheric ten times more important**

With $\sigma_{iono} = 7.0$, the following graphics show that the ionospheric error is still within the larger range of value given by *Table 3-3*, if the elevation angle is not too low. As it is shown in *Figure 3-21*, the total error on the satellite number 1 mainly follows the ionospheric error, even if the ephemeris error still has some influence. The total error is not ten times more important than in the first simulation, but only double that value.

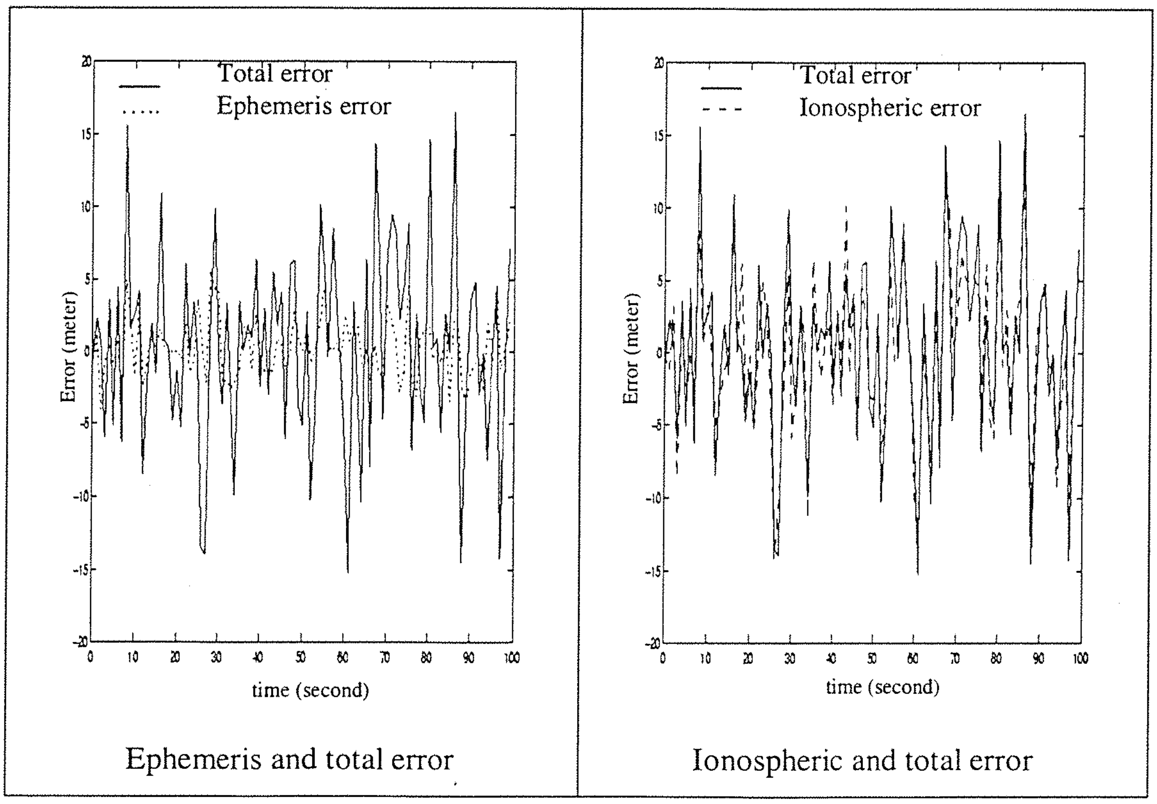


**Figure 3-21- Error on satellite number 1**

The error on the satellite 8 follows the ionospheric error completely (see *Figure 3-22*). And the values of the total error are huge (more than one kilometre on the satellite signal).

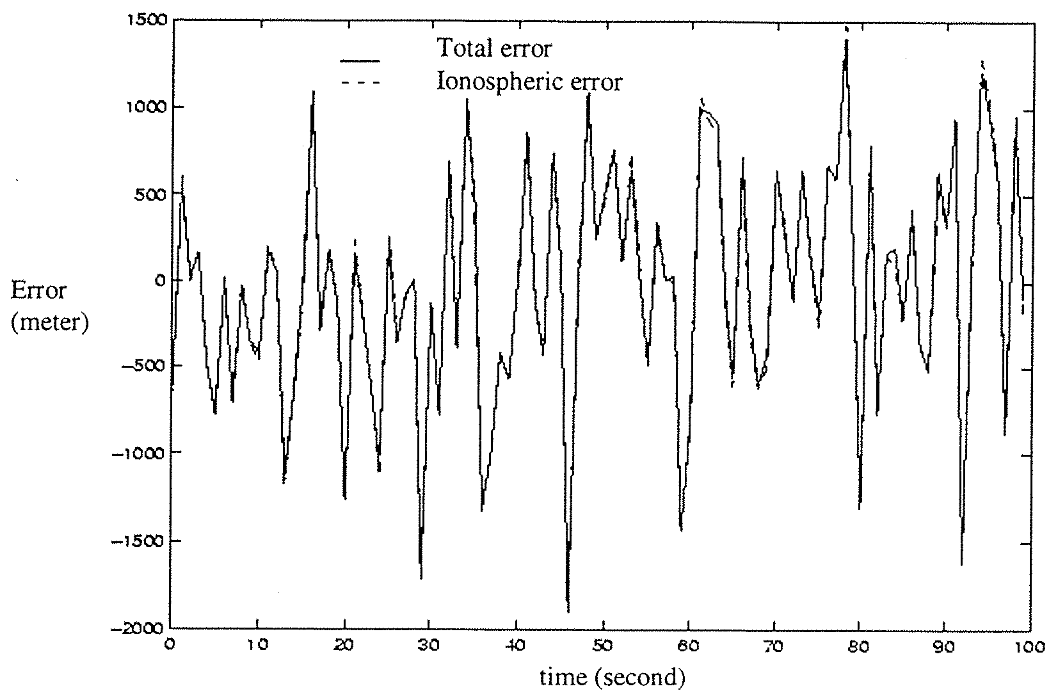


**Figure 3-22- Error on satellite number 8**

The error on satellite number 9 looks like the error on satellite number 1: It mainly follows the ionospheric error, but not totally, and the values of the total error are two or three times the values of the first simulation.

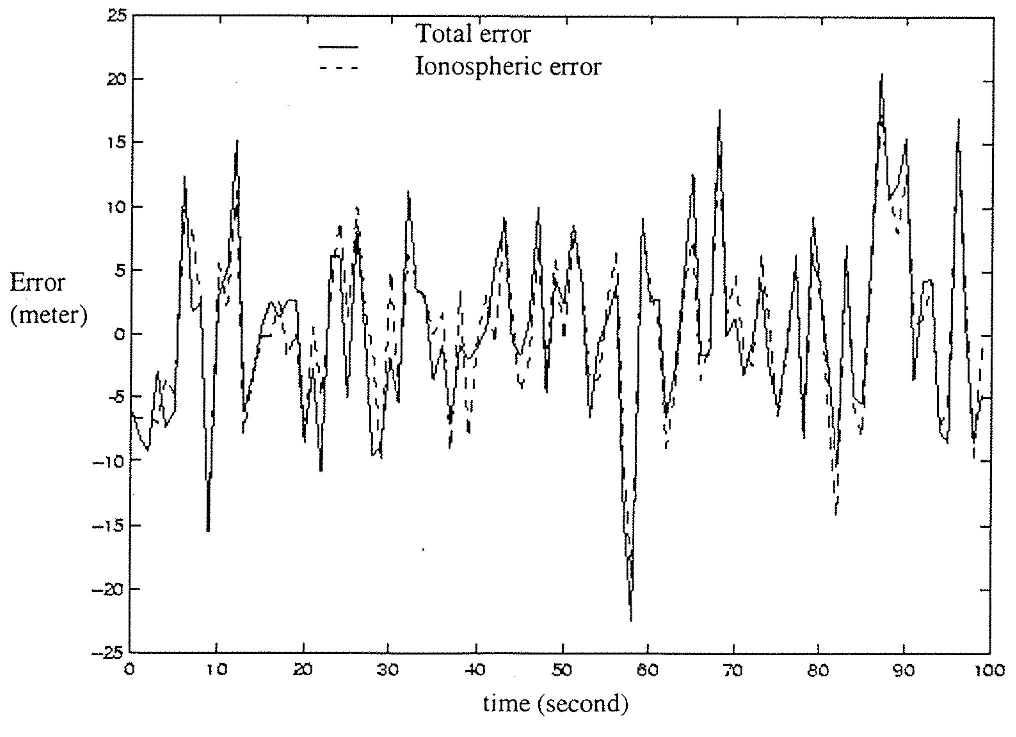


**Figure 3-23- Ionospheric and total error on satellite number 9**

As shown in *Figure 3-24*, the global error on the receiver position increases a lot compared with the first simulation (X8). But the expected error does not increase so much (X3).

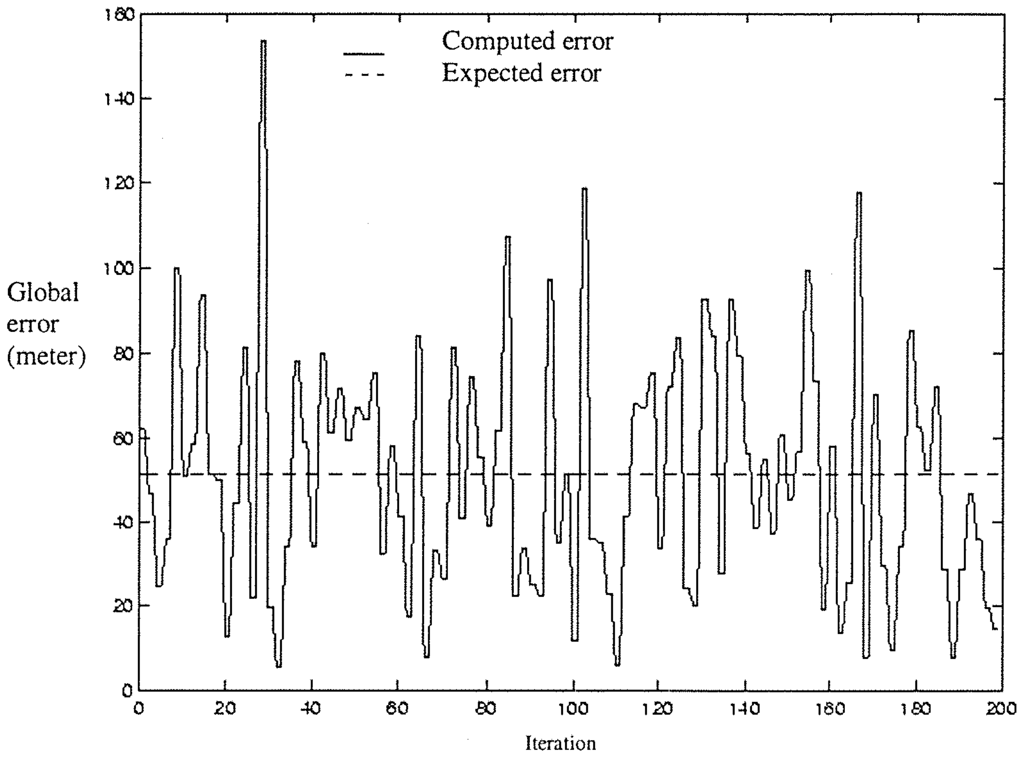


**Figure 3-24- Global error on the receiver position**

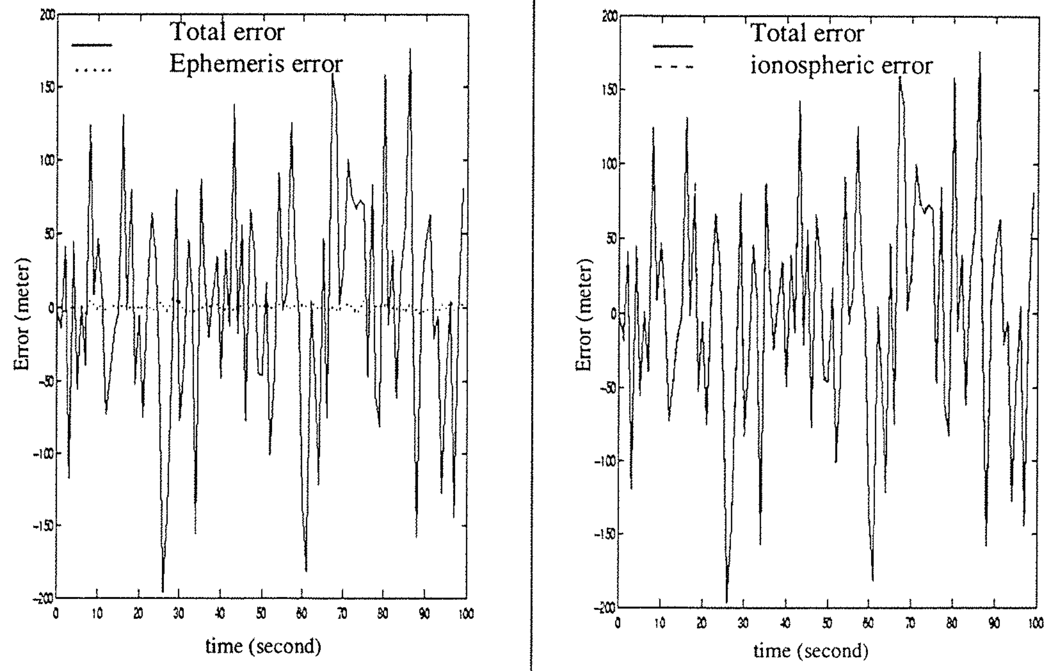
### 3.6.3.6 *Fifth simulation*

In the fifth simulation, we give one hundred times more importance to the ionospheric error than in the first simulation. As shown in the following graphics, the ionospheric error is greater than the maximum value given in the *Table 3-3*. Therefore, this simulation has got a very low probability to happen in reality. It is done to study the behaviour of the error on the receiver position.

| error | ephemeris | clock | iono | tropo | mupath | noise |
|---|---|---|---|---|---|---|
| $\sigma$ (meter) | 2.1 | 2.1 | 70.0 | 0.15 | 0.3 | 0.5 |

**Table 3-8- Sigma ionospheric one hundred times more important**

As we can see in *Figure 3-25*, the total error on satellite number 1 signal follows now completely the ionospheric error: The influence of the other error (like ephemeris error on the first graphic) can be neglected.



Ephemeris and total error

Ionospheric and total error

**Figure 3-25- Error on the satellite number 1**

The total error on the signal of satellites number 8 and 9 completely follow the ionospheric error (two curves superposed, *Figure 3-26*). The values of the total error are still realistic for the satellite number 9, but are now too huge for satellite number 8 (more than 20 kilometres sometimes).

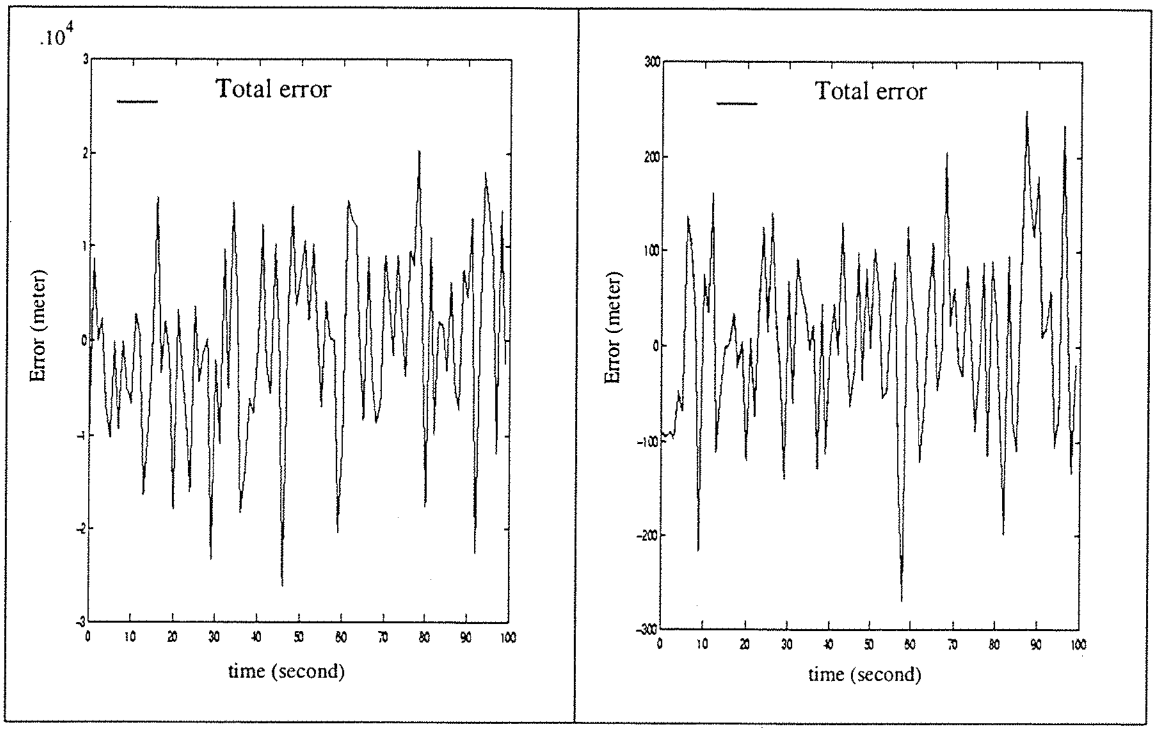


**Figure 3-26- Error on satellites number 8 and 9**

As we can see on the *Figure 3-27*, the global error on the receiver position is now really important and is not still under the expected error.

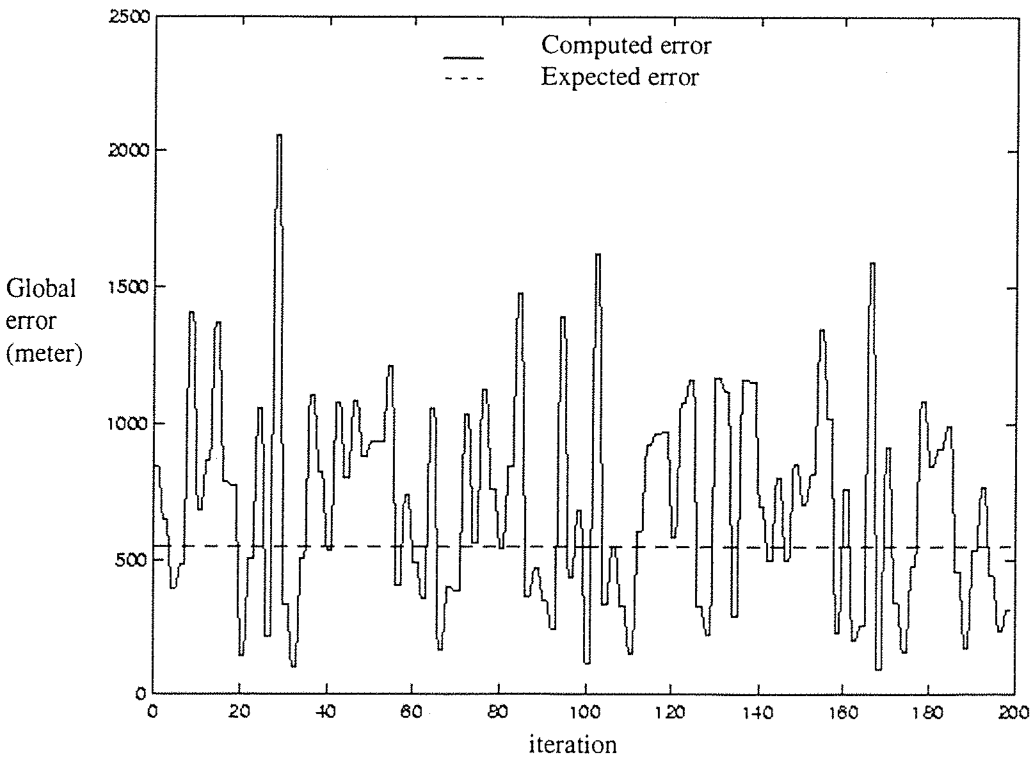


**Figure 3-27- Global error on the receiver position**

#### 3.6.3.7 Conclusions

We can see that the two types of errors on satellite signals influence the error on the receiver position differently:

- For the error independent of satellite position, if the $\sigma$ is most important, all the satellites are affected in the same way.
- For the errors dependent on the satellite positions (through the elevation angle), the variations of $\sigma$ influence the different satellites differently, depending on their position. Moreover, if the sigma of one of the errors dependent on the satellite position increases, the effect will be more important for all the satellites than if we increase a sigma from an independent error: This because the ionospheric, and tropospheric delays divide the sigma by $\sin(El_i)$, which is always lower than 1.

The global error on the receiver position is also influenced differently by the two types of error:

- If we compare the simulation number 2 and 4 (ten times more importance given to ephemeris or ionospheric sigma), and the simulation 3 and 5 (one hundred times more importance given to ephemeris or ionospheric sigma) we can see that the values of the global error on the receiver position is double when the ionospheric sigma is magnified than if the ephemeris sigma is magnified (e.g. *Figure 3-20* and *Figure 3-27*). It comes directly from the greater influence of the ionospheric sigma on the satellites error.
- The expected error does not increase as the global error on the receiver position: The expected error is equivalent to the receiver error with only satellites in view at 20 degrees elevation, which means that a major part of the ionospheric delay is minimised (There are no satellites with low elevation angle). Therefore, if we increase the ephemeris sigma, the expected error will increase directly in the

same proportion. But if we increase the ionospheric sigma, the elevation angle of 20 degrees will reduce the effect on the receiver position error.

To improve the accuracy on the receiver position, we can try to use only the satellites that have the smallest error on their signal. To do this, we can try to estimate the error on each satellite and then weight the importance of each one. This is done in the algorithm by the W matrix, which contains the weighing of each satellite.
For all the experiments described above, the matrix W was equal to identity. In the following part, the influence of the W matrix will be studied.

*Note:* As underlined in paragraph *3.6.3.1*, excepted for the noise, none of the errors on the satellite signal can be realistically simulated by white noise. The aim of this study was not to provide a realistic behaviour of the error as functions of the time, but to study the behaviour of the simulator, and the relative importance of each error compared with the others.

### 3.6.4 Influence of the satellite weighing

#### *3.6.4.1 Satellite weighing: equations*

At any time, some of the satellites in view from the receiver have a greater error on their signal than some others. The idea of the satellite weighing is to give more importance to the satellites with a small error on their signal, than to the satellites with a big one.
This can be done in the position computation algorithm by using the matrix, W.
The general equations of the algorithm are (P. Axelrad, R.G. Brown, 1996 [3]):

$$\Delta\rho[i,k] = G_{i,k}(x_{true} - x_{estimated}[i,k-1]) + Error_i$$

$$\Delta x_{estimated}[i,k] = \left(G_{i,k}{}^T W_i G_{i,k}\right)^{-1} G_{i,k}{}^T W_i . \Delta\rho[i,k]$$

with i the current acquisition, and k the current iteration.

The matrix W is the inverse of the covariance matrix. In fact the weighed least squares solution is also a minimum variance solution.
In all the previous simulations the matrix, W, was equal to the identity matrix. To weight the importance of each satellite, we have to compute the matrix, W.

The algorithm assumes that the error sources for each satellite are uncorrelated with the error sources for any other satellite.
Thus, the covariance matrix is diagonal. The terms of the matrix, W, are given by the following expression:

$$w_i = \frac{1}{\sigma^2_{ephemeris} + \sigma^2_{clock} + \sigma^2_{noise} + \sigma^2_{ionospheric} \div \sin^2(El_i) + \sigma^2_{tropospheric} \div \sin^2(El_i) + \sigma^2_{multipath} \div \tan^2(El_i)}$$

**Equation 3-2**

where $El_i$ is the elevation angle of the in view satellite number i from the receiver.
The $\sigma_{error}$ used in matrix, W, are the estimated sigma of the error put on the satellite signal. The closer the sigma are to the real values, the better is the weighing and the smaller is the error on the receiver position. Each term of the denominator corresponds to the same term of the error. The white noise provides to the error a random factor around a mean value. Therefore matrix, W, is an estimation of this error.

Matrix, W, is computed at each computation step, because it depends on the elevation angle of the satellites in view from the receiver estimate.

In the following part, the influence of the sigma values in W on the receiver position error is studied.

### 3.6.4.2 *Variation of the estimated sigma*

In all the following simulations, the same input parameters as in part *3.5.1.1* are taken, with an error on the satellites signals computed with white noise.
The values of sigma used in the matrix W must be in a realistic range. As the S/A degradation can reach 300 metres (most of the time the S/A degradation is about 30 meters) all the errors under this value can be considered realistic, because they can

be confused with the S/A degradation. Therefore, the total term of the $w_i$ denominator shall be under 300 metres.

*Table 3-9* describes the different experiment:

| Number of simulation | Description |
|---|---|
| 1 | Comparison between unweighed and weighed simulation |
| 2 | $10*\sigma_{ephemeris}$ |
| 3 | $100*\sigma_{ephemeris}$ |
| 4 | $0.1*\sigma_{iono}$ |
| 5 | $10*\sigma_{iono}$ |
| 6 | $10*\sigma_{ephemeris}$ , $10*\sigma_{clock}$ , $10*\sigma_{noise}$ |
| 7 | $100*\sigma_{ephemeris}$ , $100*\sigma_{clock}$ , $100*\sigma_{noise}$ |
| 8 | $10*\sigma_{iono}$, $10*\sigma_{tropo}$ , $10*\sigma_{mupath}$ |

**Table 3-9 Influence of satellite weighing**

3.6.4.2.1 First simulation

In the first simulation, we put the same values of sigma in the error and W. The values are shown in the *Table 3-10*. These are the common values used in W in the GPS receiver.

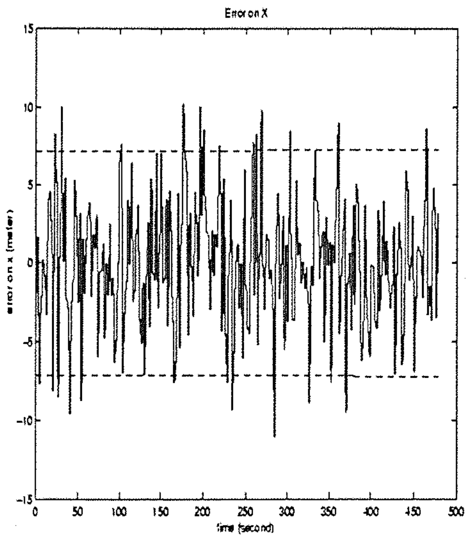
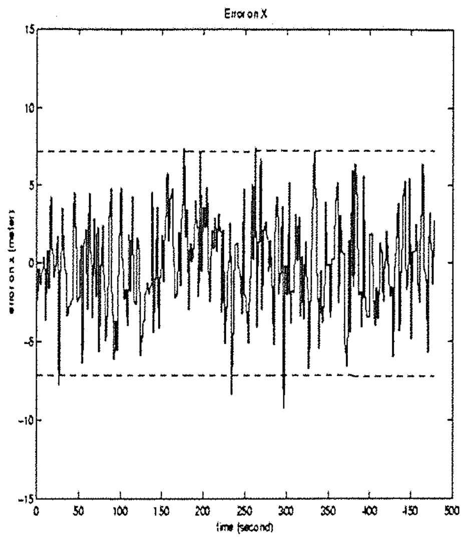
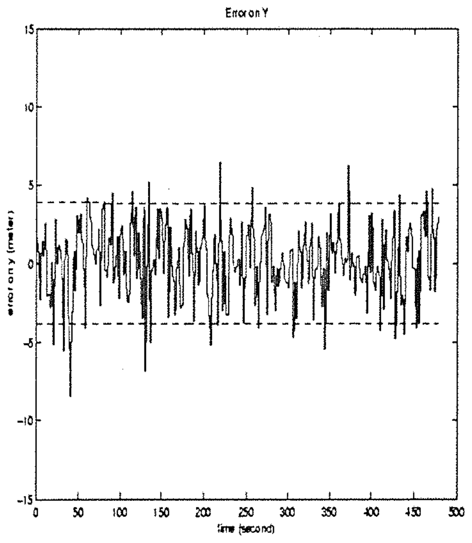
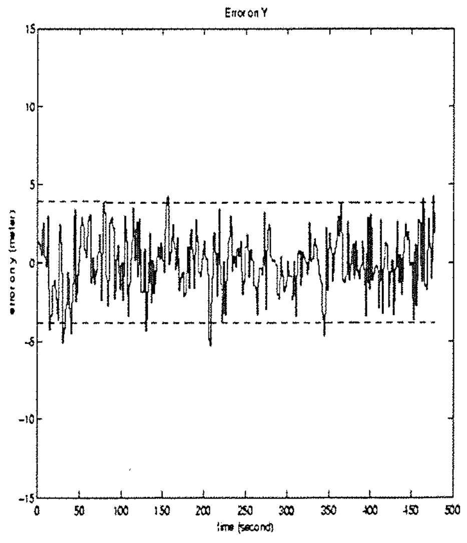
| | ephemeris | clock | ionospheric | tropospheric | multipath | noise |
|---|---|---|---|---|---|---|
| W matrix | 2.1 | 2.1 | 0.5 | 0.15 | 0.3 | 0.5 |
| Error matrix | 2.1 | 2.1 | 0.5 | 0.15 | 0.3 | 0.5 |

**Table 3-10- Weighed simulation 1**

The following graphics show the error on the receiver position between a weighed simulation and a non-weighed one. The stippled curve is the expected range of error (68.27 % of the error shall be between the two curves) equal to $GDOP \times uere$. $uere$ is the total sigma for a satellite at 20 degrees of elevation from the receiver:

$$uere = \sigma_{ephemeris} + \sigma_{clock} + \sigma_{noise} + \sigma_{iono} \div \sin(20) + \sigma_{tropo} \div \sin(20) + \sigma_{mupath} \div \tan(20)$$

The expected error represents the error that the receiver would have on its position with only satellites located at 20 degrees of elevation.

| Unweighed computation | Weighed computation |
|---|---|
|  average=-0.0211m, standard_deviation=4.317 |  average=-0.135m, standard_deviation=3.346 |
|  average=0.131m, standard_deviation=2.525 |  average=-0.067m, standard_deviation=2.037 |

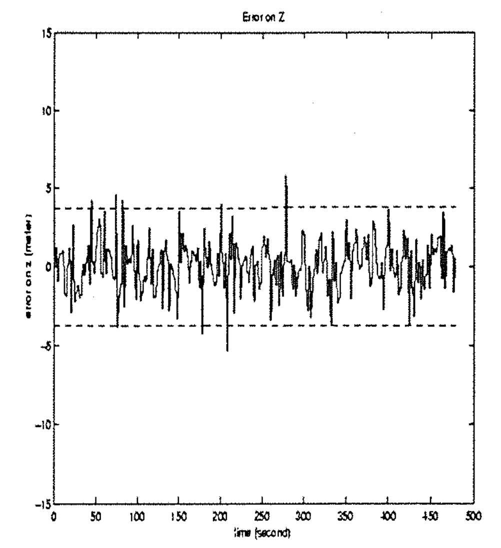
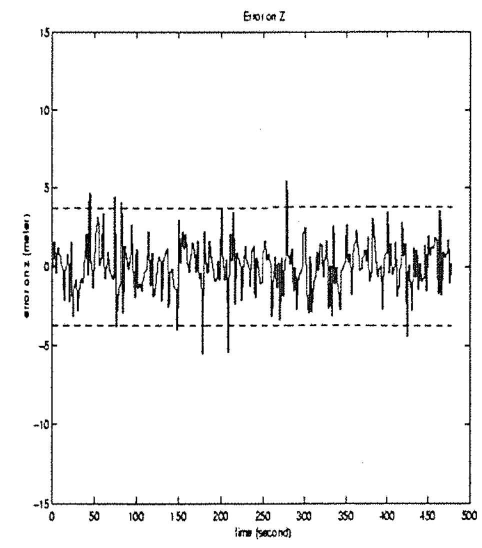
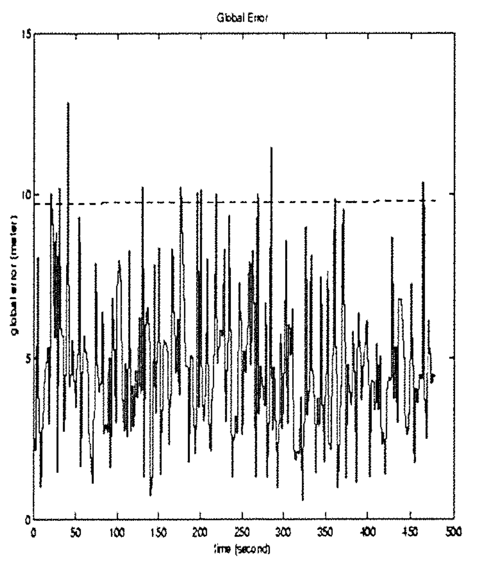
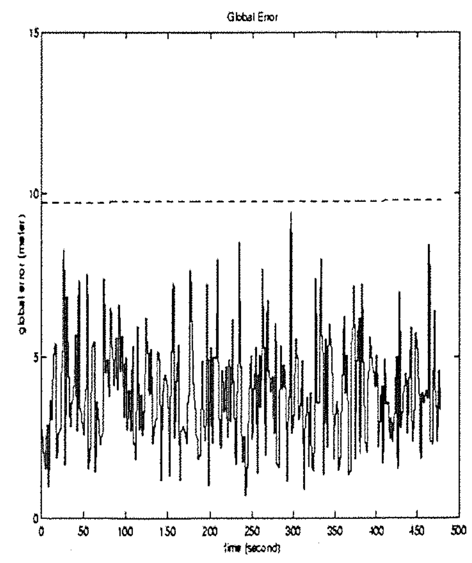
| | |
|---|---|
|  average=0.085m, standard_deviation=1.730 |  average=-0.028m, standard_deviation=1.711 |
|  average=4.727m, standard_deviation=2.385 |  average=3.901m, standard_deviation=1.754 |

**Figure 3-28- Comparison weighed/non weighed computation**

As shown in *Figure 3-28*, using the weighed matrix, W, with the same values of sigma as in the error matrix improves the accuracy on the receiver position. The global error, instead of moving between 0 and 13 metres, now moves between 0 and 10 metres. This improvement is mainly due to the elimination of the satellites at a low elevation angle: The error on their signal is greater (for ionospheric delay, tropospheric delay and multipath), than on the other satellite signals.

Now the influence of the accuracy on the weighing has been studied, it is important to know if this benefit needs a very precise computation of the W sigma.

#### 3.6.4.2.2 Second and third simulation

In the second simulation, the influence on the receiver position of a sigma value from one error independent from the satellite position (i.e. ephemeris, clock or noise error) is considered.

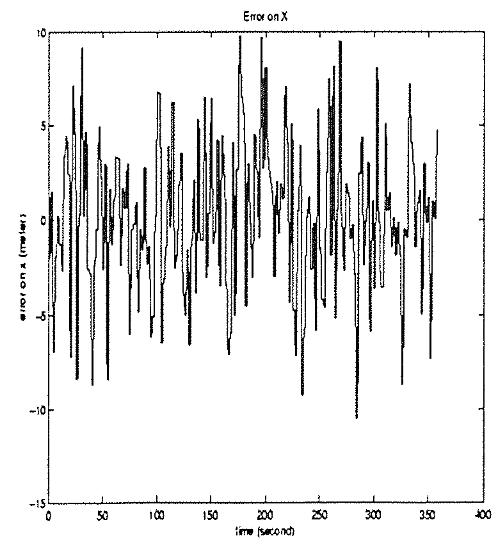
The error chosen is the ephemeris error, but the results would be the same for clock or noise error.

The *Table 3-11* shows the values used for sigma in the W and error matrix.

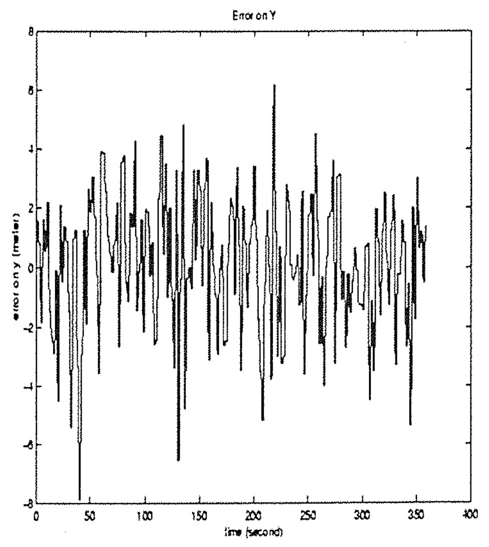
| | ephemeris | clock | ionospheric | tropospheric | multipath | noise |
|---|---|---|---|---|---|---|
| W matrix | 21.0 | 2.1 | 0.5 | 0.15 | 0.3 | 0.5 |
| Error matrix | 2.1 | 2.1 | 0.5 | 0.15 | 0.3 | 0.5 |

**Table 3-11- Weighed simulation 2-1**

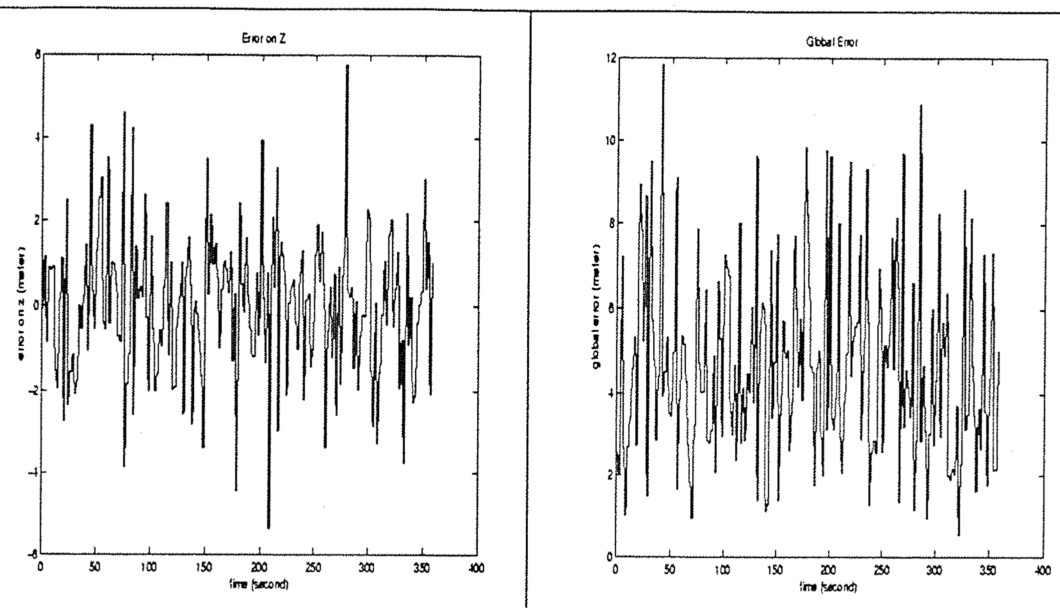
The W ephemeris sigma is ten times greater than in the error matrix. The results on the receiver position are (weighed computation):



average=0.071m, standard_deviation=4.221



average=0.075m, standard_deviation=2.425



average=0.010m, standard_deviation=1.766 average=4.620m, standard_deviation=2.342

**Figure 3-29- Ephemeris sigma ten times greater in W than in the error matrix**

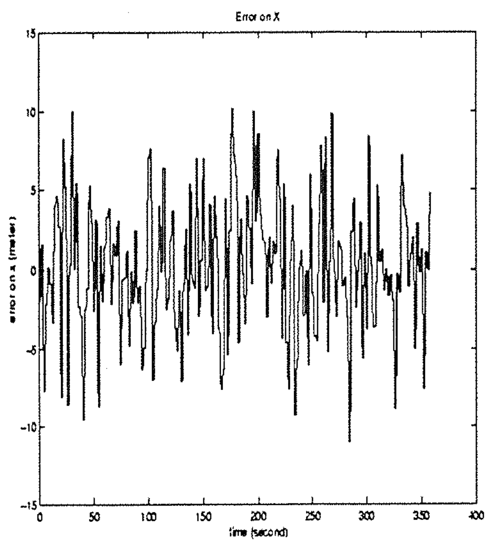
The results are nearly the same as for a simulation without weighing matrix.

The *Table 3-12* shows the second experiment on ephemeris sigma error:

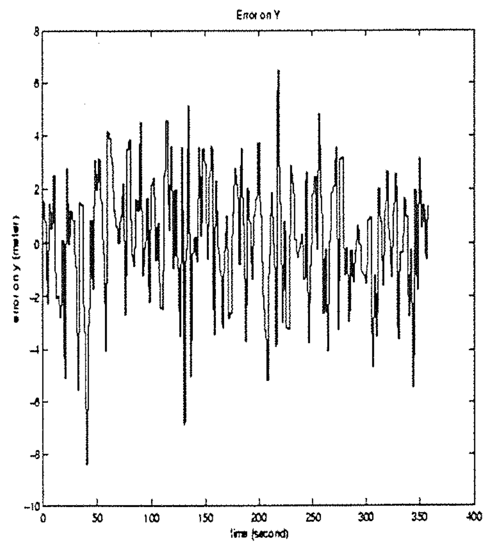
| | ephemeris | clock | ionospheric | tropospheric | multipath | noise |
|---|---|---|---|---|---|---|
| W matrix | 210.0 | 2.1 | 0.5 | 0.15 | 0.3 | 0.5 |
| Error matrix | 2.1 | 2.1 | 0.5 | 0.15 | 0.3 | 0.5 |

**Table 3-12- Weighed simulation 2-2**

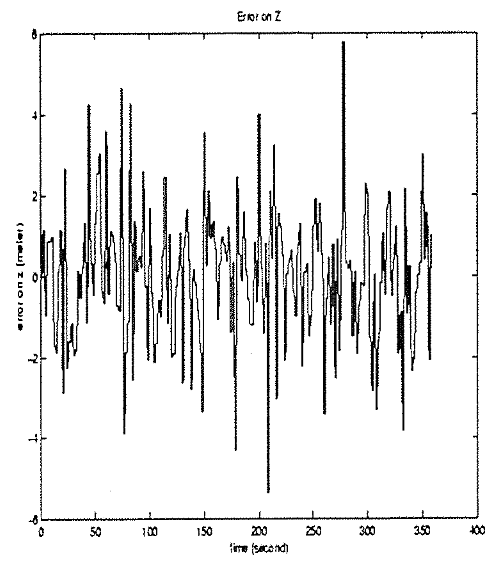
The W ephemeris sigma is now one hundred times greater than in the error matrix. The results on the receiver position are (weighed computation):



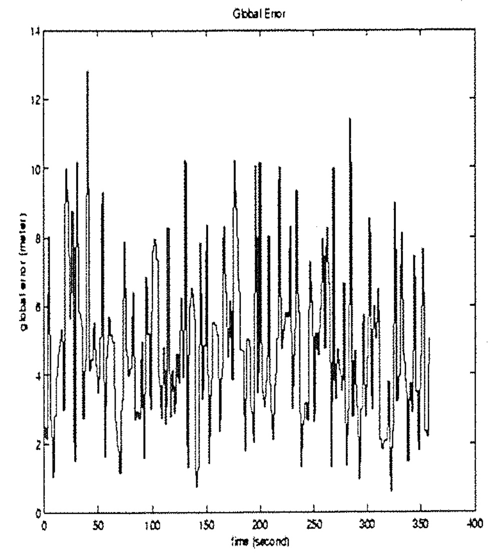
average=0.103m, standard_deviation=4.421



average=0.102m, standard_deviation=2.541



average=-0.0003m, standard_deviation=1.771



average=4.790m, standard_deviation=2.494

**Figure 3-30- Ephemeris sigma one hundred times greater in W than in the error matrix**

Compared with the previous experimentation (2-1), the global error average and the standard deviation do not increase a lot. The results are similar to a simulation without weighing matrix.

### 3.6.4.2.3 Fourth and fifth simulation

In the third simulation, the influence on the receiver position of a sigma value from one error dependent from the satellites positions (i.e. ionospheric, tropospheric and multipath error) is considered.

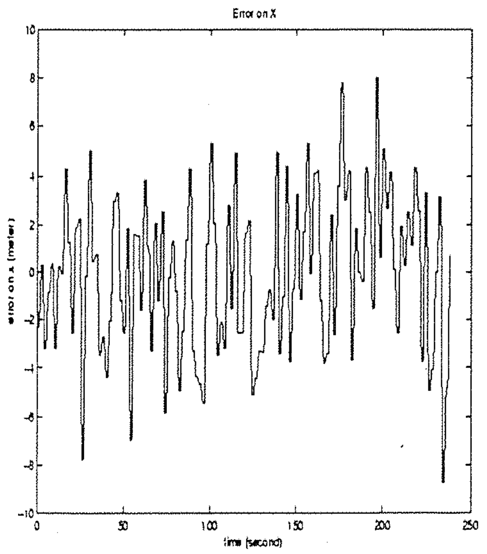
The error chosen is the ionospheric error, which is the most important error within these three ones.

The *Table 3-13* shows the values of the sigma for the first experiment on the ionospheric error.

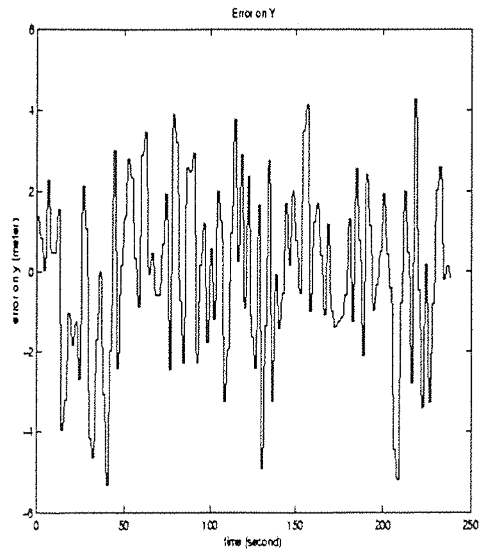
| | ephemeris | clock | ionospheric | tropospheric | multipath | noise |
|---|---|---|---|---|---|---|
| W matrix | 2.1 | 2.1 | 0.05 | 0.15 | 0.3 | 0.5 |
| Error matrix | 2.1 | 2.1 | 0.5 | 0.15 | 0.3 | 0.5 |

**Table 3-13-Weighed simulation 3-1**

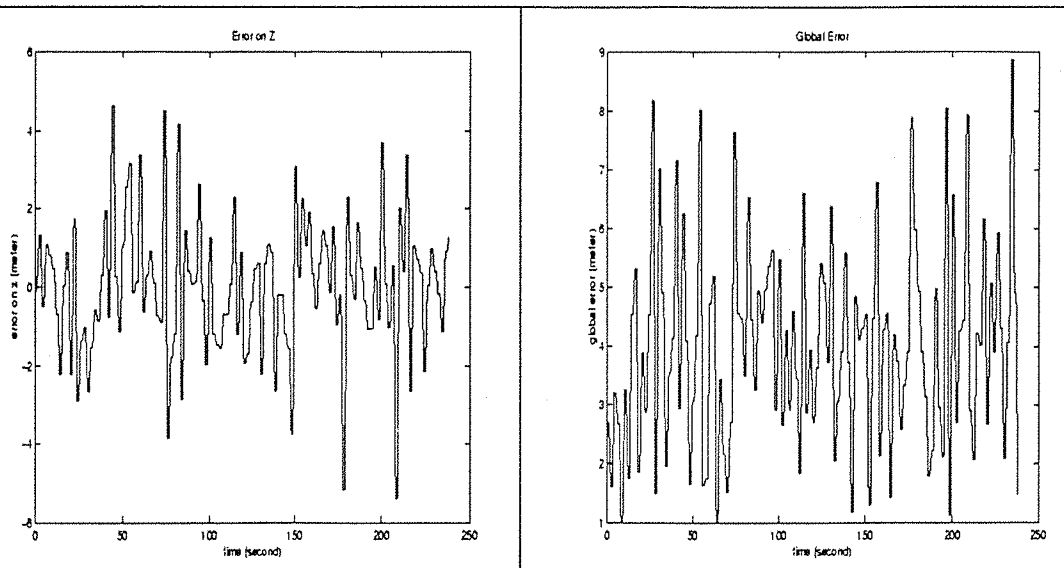
The W ionospheric sigma is ten times smaller than in the error matrix. The results on the receiver position are (weighed computation):



average=-0.100, standard_deviation=3.360



average=-0.019, standard_deviation=2.182



average=0.014, standard_deviation=1.780

average=4.008, standard_deviation=1.779

**Figure 3-31- Ionospheric sigma ten times greater in the error matrix than in W**

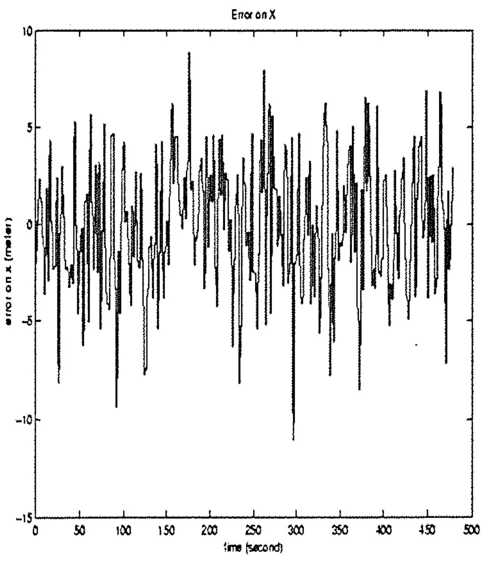
The error on the receiver position is nearly the same as for a weighed simulation with the best coefficient (first simulation).

The table *Table 3-14* shows the values of the sigma for the second experiment on the ionospheric error.

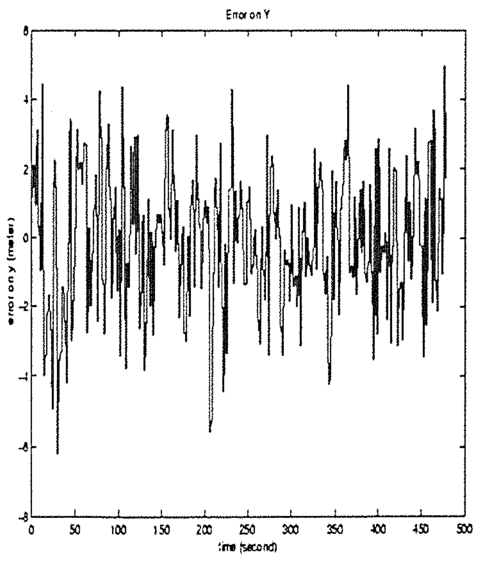
| | ephemeris | clock | ionospheric | tropospheric | multipath | noise |
|---|---|---|---|---|---|---|
| W matrix | 2.1 | 2.1 | 5.0 | 0.15 | 0.3 | 0.5 |
| Error matrix | 2.1 | 2.1 | 0.5 | 0.15 | 0.3 | 0.5 |

**Table 3-14-Weighed simulation 3-2**

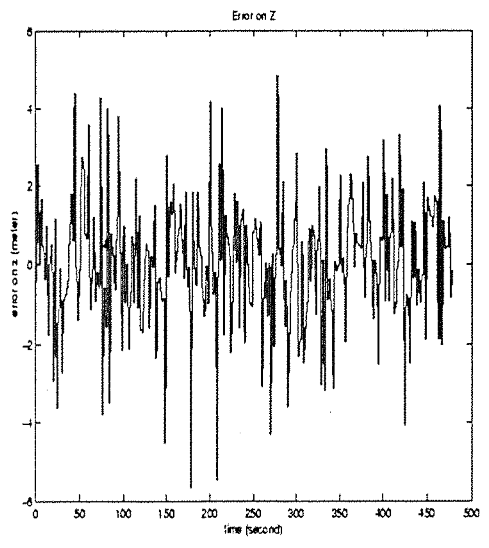
The W ionospheric sigma is ten times greater than in the error matrix. The results on the receiver position are shown in *Figure 3-32* (weighed computation).



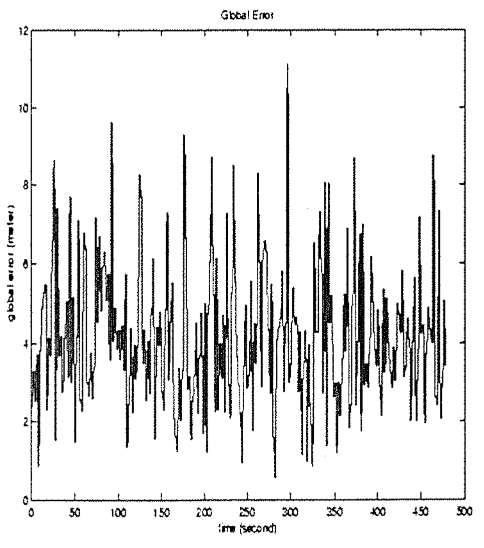
average=-0.244m, standard_deviation=3.626



average=-0.227m, standard_deviation=2.188



average=-0.066m, standard_deviation=1.802



average=4.169m, standard_deviation=1.979

**Figure 3-32- Ionospheric sigma ten times greater in W than in the error matrix**

The error maxima and average increase, but in the same order as for the second simulation: the average of the global error increases less than 1 metre compared with the first example, and the maximum is about 2.5 metres more important than for the first simulation. The results are nearly similar to a non-weighed simulation.

The results with $\sigma_{iono}(W) = 100 * \sigma_{iono}(Error)$ are nearly the same than for $\sigma_{iono}(W) = 10 * \sigma_{iono}(Error)$.

### 3.6.4.2.4 Sixth and seventh simulation

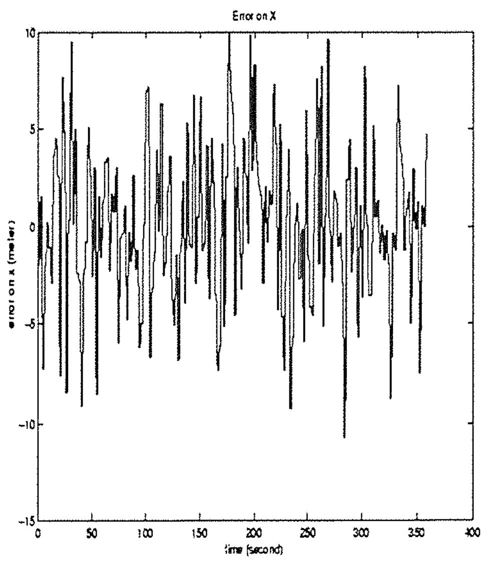
In the fourth simulation, the influence on the receiver position of the sigma values of the error independent of satellite positions is studied.

In the first experiment, the sigma of the ephemeris, clock, and noise errors are ten times greater than the common values as shown in the *Table 3-15*.

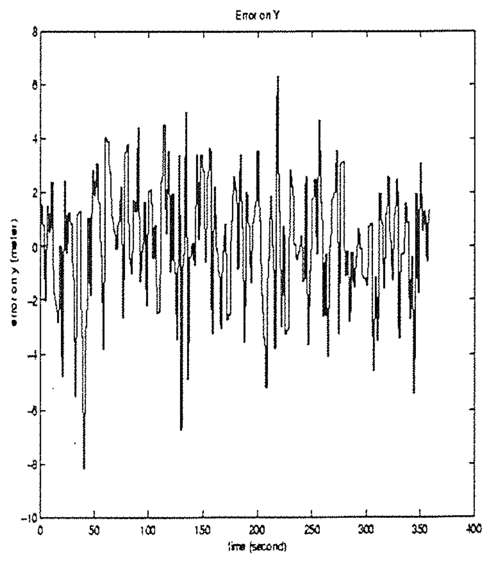
| | ephemeris | clock | ionospheric | tropospheric | multipath | noise |
|---|---|---|---|---|---|---|
| W matrix | 21.0 | 21.0 | 0.5 | 0.15 | 0.3 | 5.0 |
| Error matrix | 2.1 | 2.1 | 0.5 | 0.15 | 0.3 | 0.5 |

**Table 3-15- Weighed simulation 4-1**

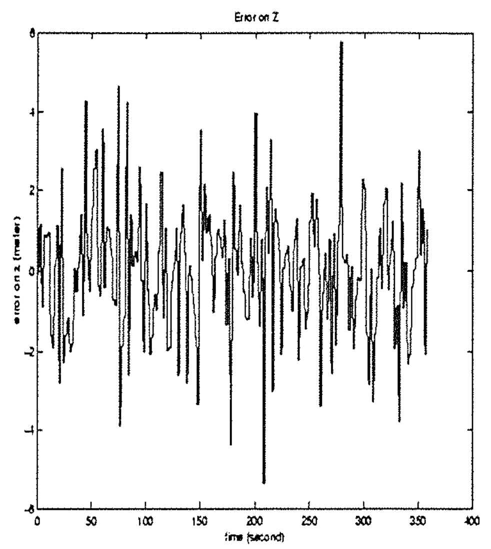
The results on the receiver position are shown in *Figure 3-33* (weighed computation).



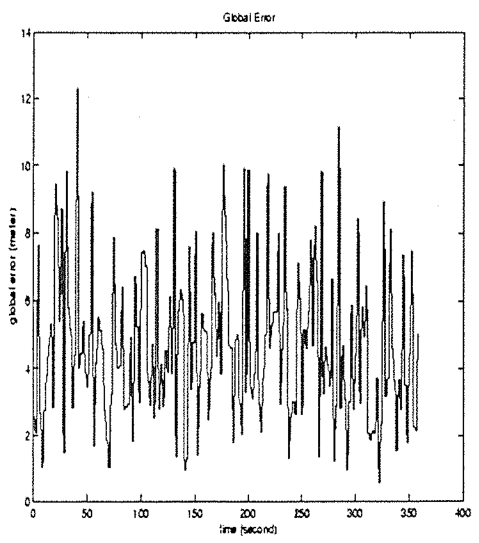
average=0.087m, standard_deviation=4.317



average=0.088m, standard_deviation=2.481



average=-0.006m, standard_deviation=1.768



average=4.701m, standard_deviation=2.414

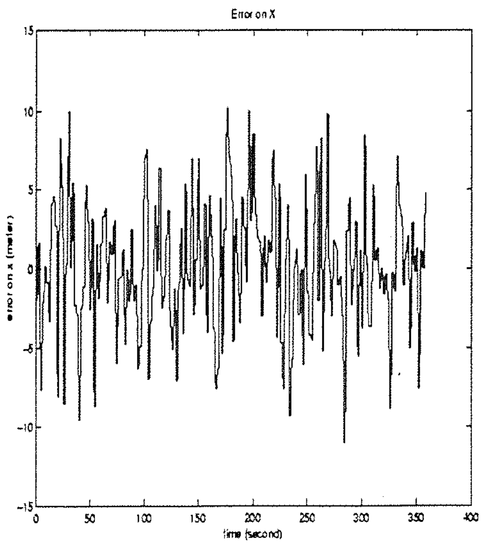
**Figure 3-33- Sigma of error independent of satellite position ten times greater in W**

In the second experiment, the sigma of the ephemeris, clock, and noise errors are one hundred times greater than the common values as shown in *Table 3-16*:

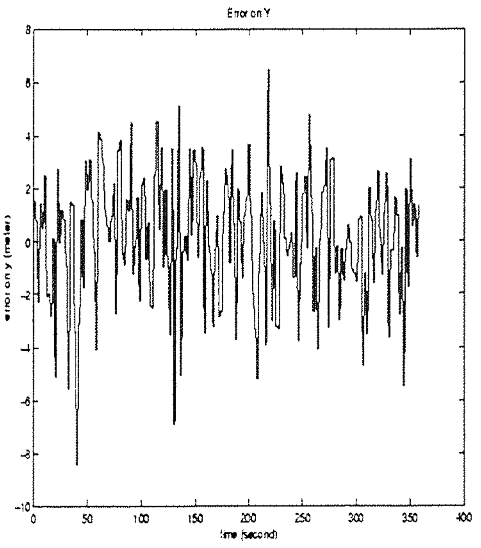
| | ephemeris | clock | ionospheric | tropospheric | multipath | noise |
|---|---|---|---|---|---|---|
| W matrix | 210.0 | 210.0 | 0.5 | 0.15 | 0.3 | 50.0 |
| Error matrix | 2.1 | 2.1 | 0.5 | 0.15 | 0.3 | 0.5 |

**Table 3-16- Weighed simulation 4-2**

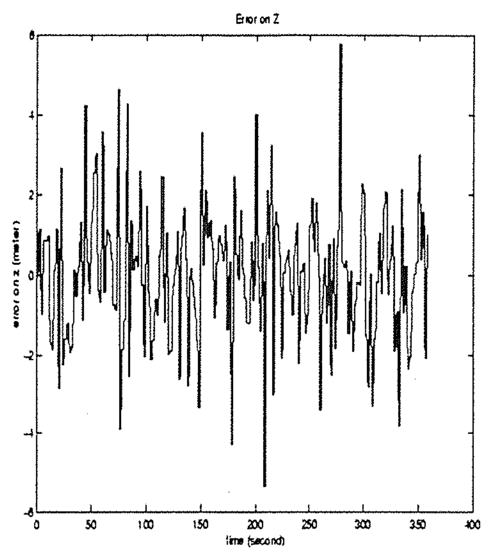
The results on the receiver position are shown in *Figure 3-34*.



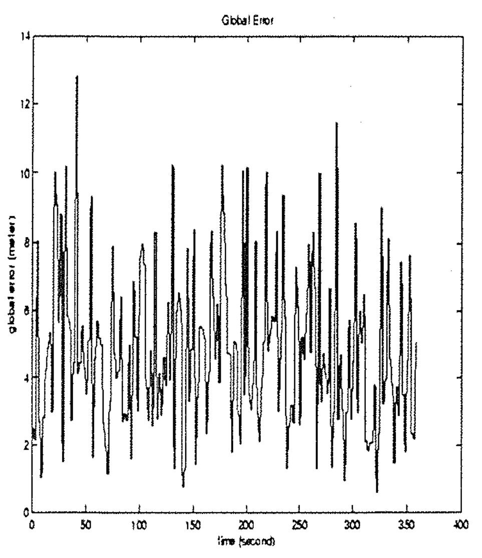
average=0.103m, standard_deviation=4.422



average=0.102m, standard_deviation=2.542



average=0.003m, standard_deviation=1.771



average=4.790m, standard_deviation=2.495

**Figure 3-34- Sigma of the error independent of satellite position one hundred times greater in W than in the error matrix**

In the two experiments, the results are very close to a non-weighed simulation.

### 3.6.4.2.5 Eighth simulation

In the eighth simulation, the influence on the receiver position of the sigma values of the error dependent from satellite positions is considered.
The sigma of the ionospheric, tropospheric and multipath errors are ten times greater than the common values as shown in the *Table 3-17*:

| | ephemeris | clock | ionospheric | tropospheric | multipath | noise |
|---|---|---|---|---|---|---|
| W matrix | 2.1 | 2.1 | 5.0 | 1.5 | 3.0 | 0.5 |
| Error matrix | 2.1 | 2.1 | 0.5 | 0.15 | 0.3 | 0.5 |

**Table 3-17- Weighed simulation 5-1**

The results on the receiver position are shown in *Figure 3-35* (weighed computation).

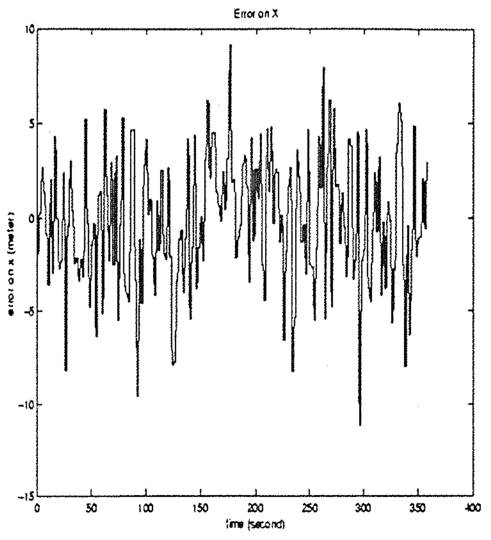
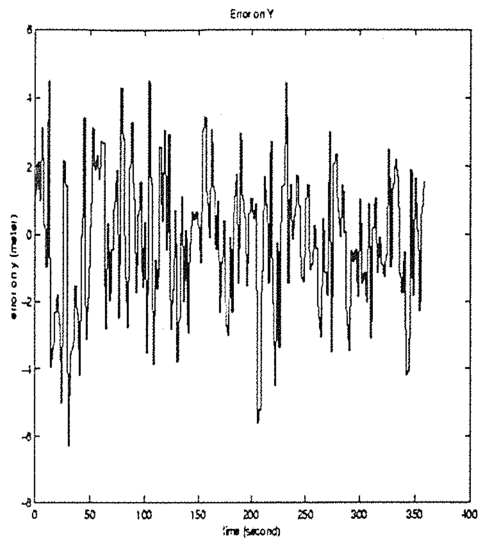
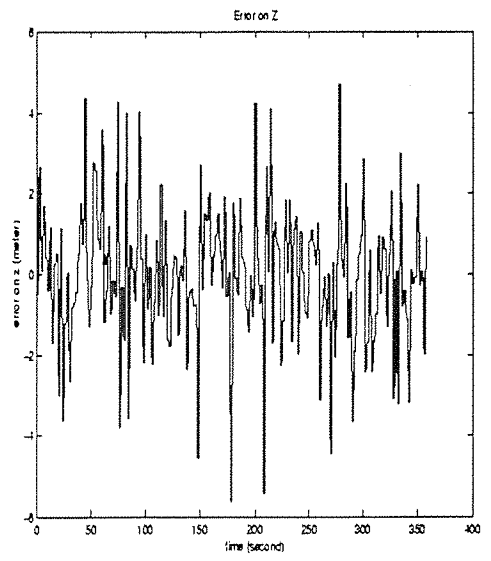
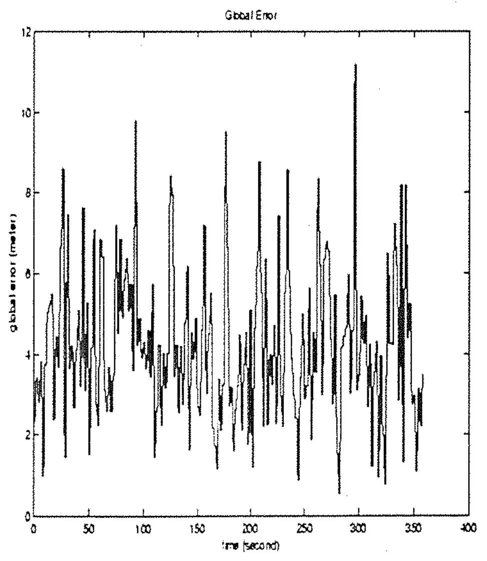
| | |
|---|---|
|  average=-0.253m, standard_deviation=3.671 |  average=-0.229m, standard_deviation=3.204 |
|  average=-0.060m, standard_deviation=1.818 |  average=4.214m, standard_deviation=1.999 |

**Figure 3-35- Sigma of error dependent from satellite position ten times greater in W than in the error matrix**

The influence on the receiver position error is nearly the same as in the fourth simulation.

If another simulation giving one hundred more important values to $\sigma_{iono}, \sigma_{tropo}, \sigma_{mupath}$ in W than in the Error matrix is done, the results are nearly the same.

#### 3.6.4.2.6 Conclusion

In order to use a precise navigation system, the weighing matrix is important: The gain on the receiver position accuracy is about 4 meters when the error on the satellite signal is well estimated. This gain can be important for users who need an accurate system, and already use P code or DGPS. These systems provide already a sharp accuracy, and the gain of using a weighing matrix is then really useful. For a classic C/A user, the gain is the same (in meter on his position), but less important in percentage of his error.

The gain due to the weighing matrix is rapidly lost if the receiver does not give a good estimation of the sigma. The weighing matrix is then only useful if the receiver is able to estimate the range of each error on the satellite signal.

The weighing matrix influences the different types of error on the satellite signal in different ways:

- W decreases the importance of the satellites with a small angle of elevation, which have a more important ionospheric, tropospheric and multipath error. Therefore, if we increase the sigma of these three types of error in W expression, the influence of the satellites with a small elevation angle will be decreased in the same proportion. But if we increase these values of sigma too much, even the satellites with a mean elevation angle will loose their importance. And the quality of the geometry of the satellites in view decreases.
- For the error independent from the satellite position (ephemeris, clock and noise), increasing or decreasing the sigma of W will influence all the satellites in the same way, and so the interest is nil.

Actually, one of the main problems concerning the weighing matrix is that the different types of error are correlated. In the following parts, the weighing matrix with the coefficients of the *Table 3-10* is used.

# 4 Ionospheric Delay models

## 4.1 Introduction

### 4.1.1 The need for improved ionospheric correction

In the previous part the simulations show that, under normal conditions, the error on the receiver position is within a range of 25 meters. The ionospheric error, in a period of low solar activity, is between 0 and 10 meters. It is the main part of the error on the satellite signal. The GPS designers tried some new systems to correct the errors on the satellite signal, and thus, the ionospheric delay. For example, the use of Differential GPS provides a good estimation of these errors using fixed ground stations, which can compute the error on the satellite signal at their position because this position is known exactly. The mobile receiver uses information about these errors to correct the satellite signal at its position. The more accurate the system that is wanted, however, the more numerous are the ground stations required and this makes the system more expensive. Moreover, a receiver moving on the sea can not use a fixed ground station.

It could seem needless to compute an accurate estimate of the error on the satellite signal in the light of the S/A degradation values: the DoD degrades the GPS satellites signal by putting a S/A error on it. The S/A degradation is most of the time lower than 100 meters, but can vary depending on the DoD decisions.
Nowadays, however, GPS receivers use additional techniques to improve the accuracy of their position. For example, the Europeans, through ESA, are trying to increase the accuracy of the GPS, by using it coupled with additional systems. These additional systems try to avoid the S/A degradation, and give a better correction of all the types of error (ephemeris, clock, ionospheric and tropospheric delay, multipath, noise). The main part of this program is called EGNOS, which uses geostationary satellites to provide corrections on the GPS satellite errors.

The aim of these improvements is to create a GNSS-II (Global Navigation Satellite System, second generation), which would integrate the GPS, GLONASS, EGNOS and regional navigation systems in order to create a very accurate navigation system. This system is planed to be used as a main navigation system, particularly in aircraft, even for precise approaches.

The future use of such systems makes the development of more accurate models to compute the errors on satellite signal, and particularly the ionospheric error needed.

### 4.1.2 General description of the ionosphere

To first order, the ionosphere is formed by the ultraviolet ionising radiation from the sun. The ionosphere is weakly ionised plasma, or gas, which can affect radio-wave propagation in various ways.
Different regions of the ionosphere are produced by different chemical species (D. Bilitza, K. Rawer, L. Bossy, T. Gulyaeva, 1993 [21]. Klobuchar, 1996 [5]. [30]):

- The D region, between 50 and 90 kilometres high, is due to cosmic radiation, solar ultraviolet radiation of hydrogen (Lyman alpha ray) and X-rays (with wavelength less than to 1 nm). This part of the ionosphere disappears during the night. It causes the absorption of radio signal at frequencies up to the low VHF band, and has no measurable effect on GPS frequencies.
- The E region between 90 and 140 kilometres, is due to solar X-rays between 1 nm and 10 nm, and to some solar ultraviolet radiation around 100 nm which ionises the oxygen. This part of the ionosphere disappears during the night. The E region produced by X-rays has a minimal effect on the GPS signal.
- The F1 region between 140 and 210 kilometres is due to the solar ultraviolet rays between 10 nm and 80 nm wavelength. This part of the ionosphere still exists during the night. The F1 region combined with the E region can account for up to 10% of ionospheric delay encountered by GPS signal.
- The F2 region between 210 and 1000 kilometres, is the densest part of the ionosphere and it has also the highest variability, causing the main part of the ionospheric delay on the GPS signal. The height of the peak

of the electron density is commonly between 250 and 400 kilometres, but can be lower or higher under extreme conditions. This part of ionosphere is mainly due to ionisation of oxygen atoms by solar ultraviolet rays.

- The protonosphere (height > 1000 km) is a region of ionised hydrogen. It is a low density region, but extends out to approximately the height of the GPS satellite orbits. It can be a significant source of ionospheric delay on GPS signal. Estimates of the contribution of the protonosphere vary from 10%of the total ionospheric delay during the day, when the electron density of the F2 region is the highest, to 50% during the night, when the F2 density is low. Electron density of the protonosphere does not change by a large amount, except during major magnetic storms. In this case it can take several days to recover the pre-storm values.

The general profile of the electron density as a function of the height is shown in *Figure 4-1* (From [31]). This experiment, in a low solar activity period, shows that the daytime density is higher than the nighttime density, but follows the same profile.

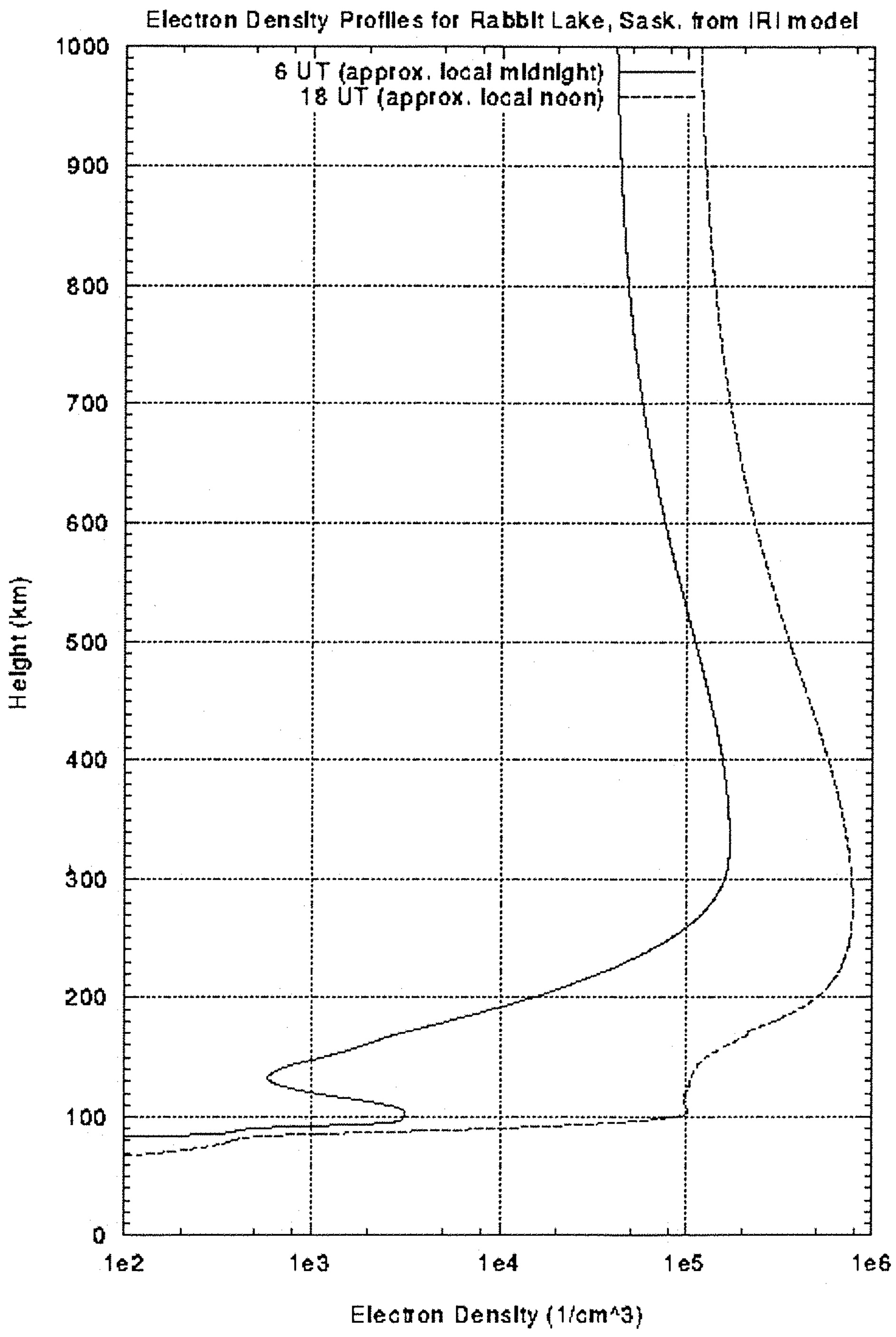


**Figure 4-1- Electron density of the ionosphere in function of the height**

The *Figure 4-2* [30] shows that the effect of ionosphere on the signal of the satellites is a function of the frequency. Almost all the perturbation of the ionosphere on the satellite signal is due to perturbations on its speed, whereas the perturbations on its trajectory are negligible. Regarding this graphic, the GPS signal, on both frequencies, is less perturbed than a signal with a lower frequency.

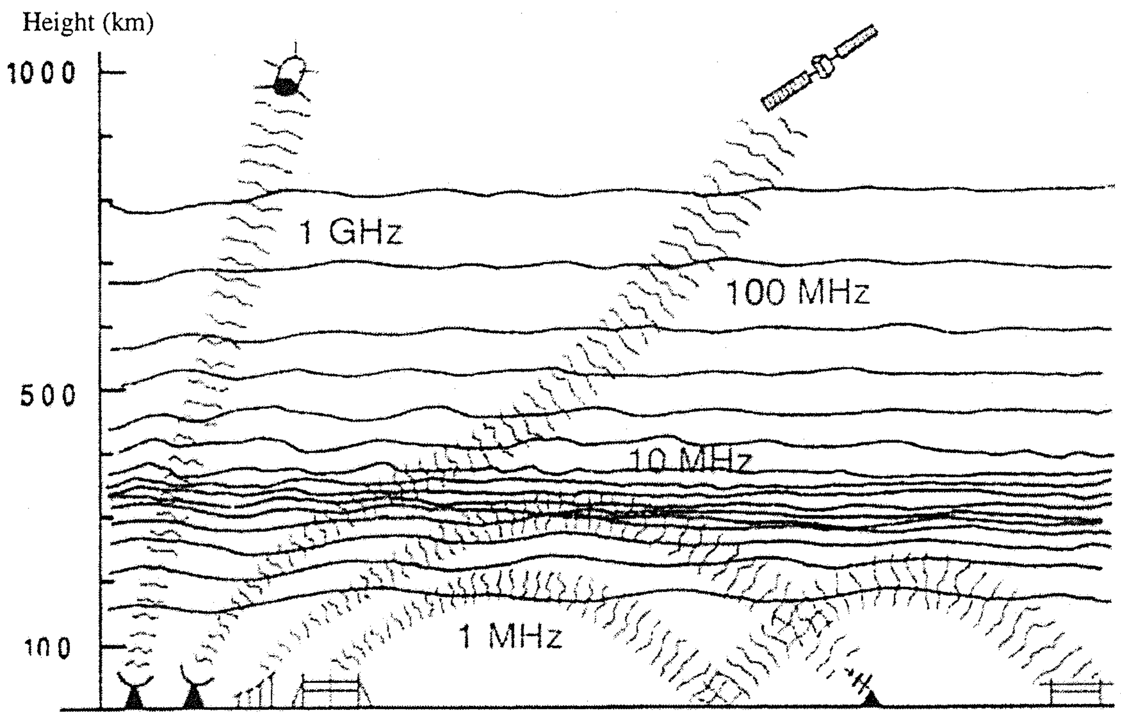


**Figure 4-2- Signal deviation due to ionosphere**

## 4.2 Ionospheric delay computation

### 4.2.1 Single frequency group delay

The group delay of the ionosphere produces range errors, which can be expressed either in units of distance or in units of time delay. The following equation describes how to compute it,

$$\Delta t = \frac{1}{c}\int (1-n)dl$$

**Equation 4-1**

where Δt is in seconds, and n is the refractive index of the ionosphere, and can be expressed as in the *Equation 4-2* (D.J. Crain, J.J. Sojka, R.W. Schunk P.H. Doherty, J.A. Klobuchar, 1993 [22]. Klobuchar, 1996 [5]):

$$n^2 = 1 - \frac{X}{1 - i*Z - \frac{Y_T^2}{2*(1-X-i*Z)} \pm \left[ \frac{Y_T^4}{4*(1-X-i*Z)^2} + Y_L^2 \right]}$$

**Equation 4-2**

where $X = \frac{Ne^2}{\varepsilon_0 m\omega^2} = \frac{f_n^2}{f^2}$

$Y_L = \frac{eB_L}{m\omega} = \frac{f_H \cos\theta}{f}$

$Y_T = \frac{eB_T}{m\omega} = \frac{f_H \sin\theta}{f}$

$Z = \frac{\upsilon}{\omega}$

$\omega = 2\Pi f$

| | | |
|---|---|---|
| with | N | local number of electron per meter on the user-satellite path. |
| | e | electron charge, -1.602 $10^{-19}$ coulomb. |
| | $\varepsilon_0$ | permittivity of the space, 8.854 $10^{-12}$ farad/m. |
| | m | rest mass of electron, 9.107 $10^{-31}$ kg. |
| | $f_n$ | plasma frequency. |
| | f | system frequency in Hz. |
| | $\theta$ | angle of the ray with respect to the Earth's magnetic field. |
| | $\upsilon$ | electron-neutral collision frequency. |
| | $f_H$ | electron gyro frequency. |

The electron gyro frequency $f_H$ is typically 1.5 MHz, the plasma frequency $f_n$ rarely exceeds 20 MHz, and the collision frequency $\upsilon$ is approximately $10^4$ Hz.
With these values, a simpler expression of n with an accuracy better than 1% compared with the value provided by *Equation 4-2* can be derived:

$$n = 1 - \frac{X}{2}$$

**Equation 4-3**

with $X = \frac{40.3}{f^2}\int Ndl$ (first approximation) and *N* the local number of electron per meter along the path from receiver to a satellite.

Therefore, using the *Equation 4-1* the ionospheric delay becomes:

$$\Delta t = \frac{40.3}{cf^2}\int Ndl = \frac{40.3}{cf^2}TEC \text{ in seconds.}$$

**Equation 4-4**

where $\int Ndl$ is the Total Electron Content (TEC) in el/m$^2$, integrated along the path from receiver to each in view satellite.

Some models of the ionosphere can compute the values of TEC. Some organisations provide the world-wide values of the TEC. For example, the Deutsche Forschungsanstalt für Luftund Raumfahrt [29] proposes some maps of hourly vertical TEC (VTEC). Having the ionospheric pierce point (defined in *Annexe I*) of each satellite in view from the receiver, the ionospheric delay can be computed from these maps (described in *Figure 4-3*). The TEC is obtained from VTEC by multiplying it by an obliquity function.

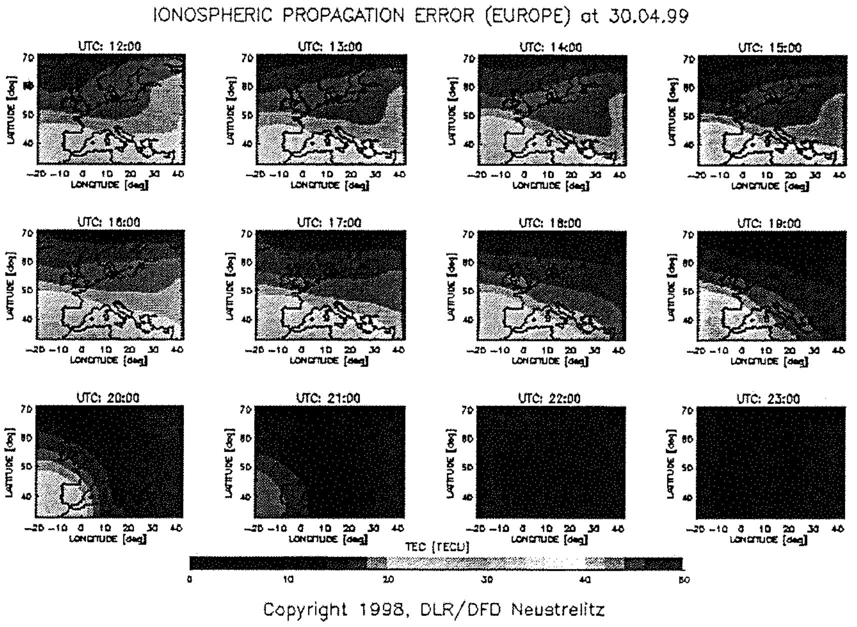


**Figure 4-3- VTEC on Western Europe**

### 4.2.2 Dual frequency group delay

By using a dual frequency receiver, the user can compute the real ionospheric error. In fact these values are an estimation of the real ionospheric delay, with less than 10% of error between the two values. The GPS satellites broadcast their signal on two frequencies L1 (1575.42 MHz) and L2 (1227.6 MHz). This error is due to the time bias between the receiver and the satellite. We can measure the difference between the two ionospheric delays, which becomes, using the *Equation 4-4* (F.K. Brunner, M. Gu, 1991 [24]. Klobuchar, 1996 [5]):

$$\delta(\Delta t) = |\Delta t_2 - \Delta t_1| = \left(\frac{40.3}{c}\right) * TEC * \left\| \left[\frac{1}{f_1^2} - \frac{1}{f_2^2}\right] \right\| = \Delta t_1 * \left(\frac{f_1^2 - f_2^2}{f_2^2}\right)$$

and we have the ionospheric delay $\Delta t_1$ as a function of $\delta(\Delta t)$

$$\Delta t_1 = \left(\frac{f_2^2}{f_1^2 - f_2^2}\right) * \delta(\Delta t)$$

**Equation 4-5**

However, it is not easy to measure the difference $\delta(\Delta t)$. It is easier to measure the difference between the two times delays. In this case, $\Delta t_1$ and $\Delta t_2$ are the total time delays, and $\delta(\Delta t)$ Is the difference between these two time delays. Therefore, $\Delta t_1$ and $\Delta t_2$ are obtained by $\Delta t_1 = \frac{\rho_1}{c}$ and $\Delta t_2 = \frac{\rho_2}{c}$. $\rho_1$ and $\rho_2$ are equal to the distance between the receiver (estimated position) and the satellite, including the errors on the satellite signal (cf *Equation 2-3*).

Therefore, $\delta(\Delta t)$ is obtained by the following equation:

$$\delta(\Delta t) = |\Delta t_2 - \Delta t_1| = \left|\frac{\rho_2}{c} - \frac{\rho_1}{c}\right|$$

And then $\Delta t_1$ is given by the Equation 4-6:

$$\Delta t_1 = \frac{|\rho_2 - \rho_1|}{c} * \frac{f_2^2}{f_2^2 - f_1^2}$$

**Equation 4-6**

In the same way, $\Delta t_2$ could be computed. But only $\Delta t_1$ is interesting, because the single frequency ionospheric models describes below use only data broadcast on L1 (frequency f$_1$). The advantage of this method is that we can directly measure $\Delta t_1$ and $\Delta t_2$ with a dual frequency receiver. But, even if most of the errors on the satellite signal are cancelled by making the difference between the two delays, the *Equation 4-6* does not represent the time delay coming from the ionosphere delay only. The main remaining error is the multipath one, which might interfere with the ionospheric delay.

## 4.3 GPS receiver ionospheric model-Model of Klobuchar

### 4.3.1 Presentation

The model of Klobuchar ([5] and [23]) is the model currently used in the GPS receiver to estimate the ionospheric delay.
This model uses some parameters broadcast by the GPS satellites. It is estimated that it provides at least a 50% reduction on the ionospheric delay (W.A. Feess, S.G. Stephens, 1987 [25]).

The algorithm is described below (J.A. Klobuchar 1987 [23], 1996 [5]).
The ionospheric delay is given by:

$$T_{IONO} = \begin{cases} F * \left[ 5.0*10^{-9} + AMP * \left( 1 - \frac{x^2}{2} + \frac{x^4}{24} \right) \right] & if|x| < 1.57 \\ F * 5.0 * 10^{-9} & if|x| \geq 1.57 \end{cases} \text{ in seconds.}$$

**Equation 4-7**

where

$$AMP = \sum_{n=0}^{3} \alpha_n * \phi_m^n \begin{cases} If \quad AMP \geq 0 \quad then \quad AMP = \sum_{n=0}^{3} \alpha_n * \phi_m^n \\ If \quad AMP < 0 \quad then \quad AMP = 0 \end{cases} \text{ in seconds.}$$

$$x = \frac{2 * \pi * (t - 50400)}{PER} \quad \text{in radians.}$$

$$PER = \sum_{n=0}^{3} \beta_n * \phi_m^n \text{ and} \begin{cases} If \quad PER < 72000 \quad then \quad PER = \sum_{n=0}^{3} \beta_n * \phi_m^n \\ If \quad PER \geq 72000 \quad then \quad PER = 72000 \end{cases} \text{ in seconds.}$$

$$F = 1.0 + 16.0 * (0.53 - E)^3$$

$$\phi_m = \phi_i + 0.064 * \cos(\lambda_i - 1.67) \quad \text{in radians within [-}\Pi,\Pi\text{].}$$

$$\lambda_i = \lambda_u + \frac{\psi * \sin(A)}{\cos(\phi_I)}$$ in radians. within [-Π,Π].

$$\phi_i = \phi_u + \psi * \cos(A) \text{ and } \begin{cases} If \quad \phi_i < -0.416 \quad then \quad \phi_i = 0.416 \\ If \quad |\phi_i| \le 0.416 \quad then \quad \phi_i = \phi_u + \psi * \cos(A) \\ If \quad \phi_i > 0.416 \quad then \quad \phi_i = 0.416 \end{cases}$$

in radians within [-Π,Π].

$$\psi = \frac{0.00137}{E + 0.11} - 0.022$$ in radians within [-Π,Π].

$$t = 4.32 * 10^4 * \lambda_i + GPS\_time \begin{cases} if \quad t < 0 \quad t = t + 86400 \\ if \quad t \ge 86400 \quad t = t - 86400 \end{cases}$$ in seconds.

The notation is summarised as follows:

- Satellite transmitted data:

| | |
|---|---|
| $\alpha_n$: | The coefficient of a cubic equation, representing the amplitude of the vertical ionospheric delay (4 coefficients, 8 bits each). |
| $\beta_n$: | The coefficient of a cubic equation, representing the period of the model (4 coefficients, 8 bits each) |

- Receiver general terms:

| | |
|---|---|
| E: | estimated elevation angle of the satellite from the receiver (radians within [-Π,Π]). |
| A: | estimated azimuth angle of the satellite from the receiver. Measured positive from the true (radians within [-Π,Π]). |
| $\phi_u$: | receiver geodetic latitude (radians within [-Π,Π]) in WGS84. |
| $\lambda_u$: | receiver geodetic longitude (radians within [-Π,Π]) in WGS84. |
| GPS_time: | receiver computed system time (in second) in the day. |

- Computed terms:

| | |
|---|---|
| x: | Phase (in radians). |
| F: | Obliquity factor (dimensionless). |
| t: | local time (in second). |
| $\phi_m$: | Geomagnetic latitude of the location looking toward the satellite (in radians). |
| $\lambda_i$: | Subionospheric longitude (radians within [-Π,Π]). |
| $\phi_i$: | Subionospheric latitude (radians within [-Π,Π]). |
| $\psi$: | Earth central angle between the receiver position and the pierce point direction (radians within [-Π,Π]). |

### 4.3.2 Simulations

In this part a comparison is done between the values of ionospheric delay from the model of Klobuchar and the real ionospheric delay provided by a dual frequency receiver (*part 4.2.2*). These data come from the dual frequency receiver of Alcatel Space Industries in Toulouse.

The parameters of the experiment are:

| Initial estimated receiver position: | | Initial true receiver position: | |
|---|---|---|---|
| X=4628000.0000 | meters | X=4629365.4041 | meters |
| Y=113000.0000 | meters | Y=112099.4237 | meters |
| Z=4372000.0000 | meters | Z=4371618.9541 | meters |
| Bias=0 | second | Bias=0.00028352948 | seconds |

The satellites data come from real data files. Nine satellites are in view during the simulation from the receiver. To see the effect of a low elevation angle on the ionospheric delay, no mask angle is used.
The elevation angles of the nine satellites from the receiver (computed with ellipsoid Earth model) are listed in the *Table 4-1*.

| satellite | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Elevation angle (radians) | 1.35753 | 0.158422 | 0.315615 | 0.204632 | 0.910624 | 0.667757 | 0.724319 | 0.006486 | 0.725082 |

**Table 4-1- Elevation angle of the satellites**

In the following graphics, the real ionospheric error (continuous line) computed with *Equation 4-6* and the ionospheric error computed with the model of Klobuchar (dotted line) are compared.

The real ionospheric delay has two types of variations:

- Some slow variations, mainly due to the solar activity variations, and to the difference of light level at each pierce point as a function of the sun and pierce point movement. These variations are temporal.
- Some fast variations, mainly due to the movement of the receiver estimated position, and to the pierce point position into the ionosphere. These variations are spatial.

It can be noticed that these variations are in a small range (commonly less than 1 meter per minute on the ionospheric error).

Considering satellite number 1 (*Figure 4-4*), which has a high angle of elevation, the real ionospheric error does not exceed 1.8 meters. The mean value of the real error is 1.24 meter.

The ionospheric error coming from the model is nearly a straight line, around 0.75 meters. It means that the model can be considered efficient for an average value of the ionospheric delay, but it does not follow the fast variations of the real error.

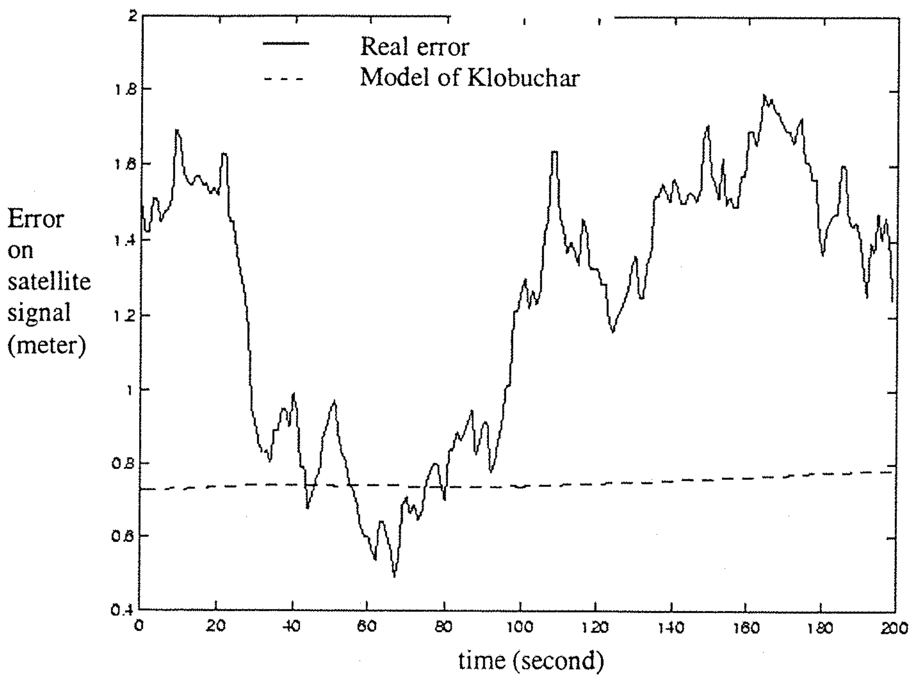


**Figure 4-4- Ionospheric error on satellite number 1**

Considering satellite number 8, which has a really small elevation angle, it can be noted that the real error varies more quickly as a function of the time. The mean value of the real error is slightly higher than for satellite number 1: it is 3.37 meters. The ionospheric delay provided by the model has a mean value of 3.18 meters and it follows the variations of the real error more closely than for the satellite number 1.

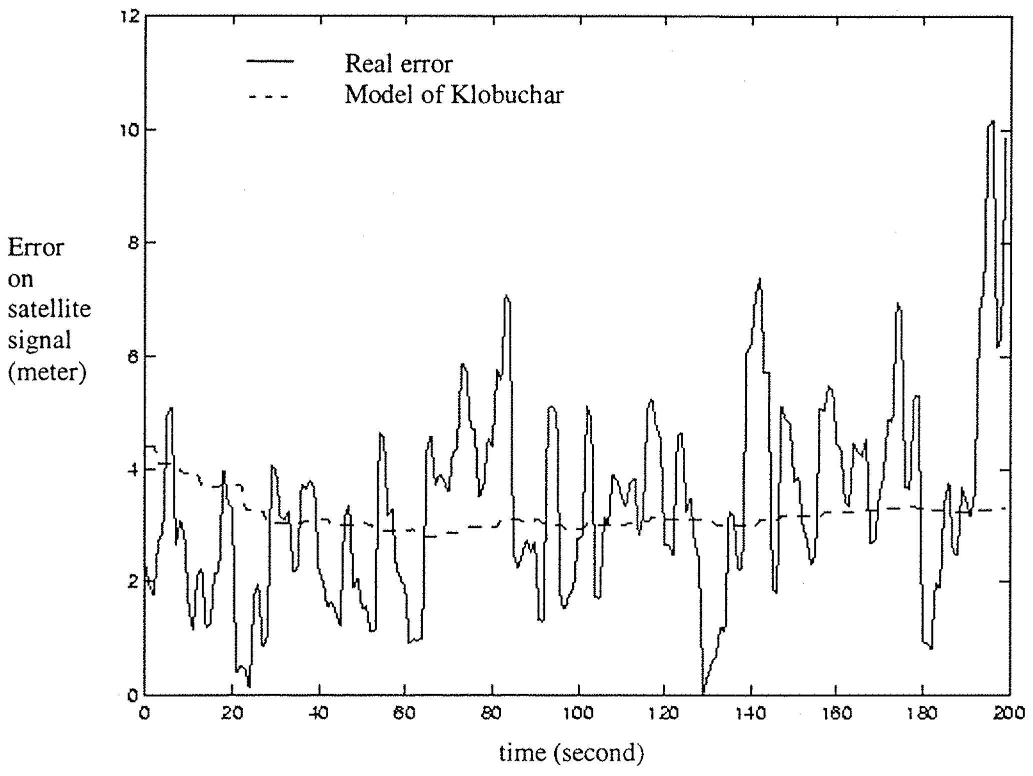


**Figure 4-5- Ionospheric error on satellite number 8**

For the satellites with an average elevation angle (satellite number 7 in *Figure 4-6*, and satellite number 9 in *Figure 4-7*), the real ionospheric error is not really important (lower than 1.8 meter), and the variations are quite slow in this range. The error coming from Klobuchar model does not follow these variations. The *Table 4-2* shows the mean values of real ionospheric delay, and modelled ionospheric delay.

| Satellite | 7 | 9 |
|---|---|---|
| Mean value of real error (meter) | 0.45 | 0.99 |
| Mean value of modelled error (meter) | 0.46 | 0.33 |

**Table 4-2- Comparison of the mean values**

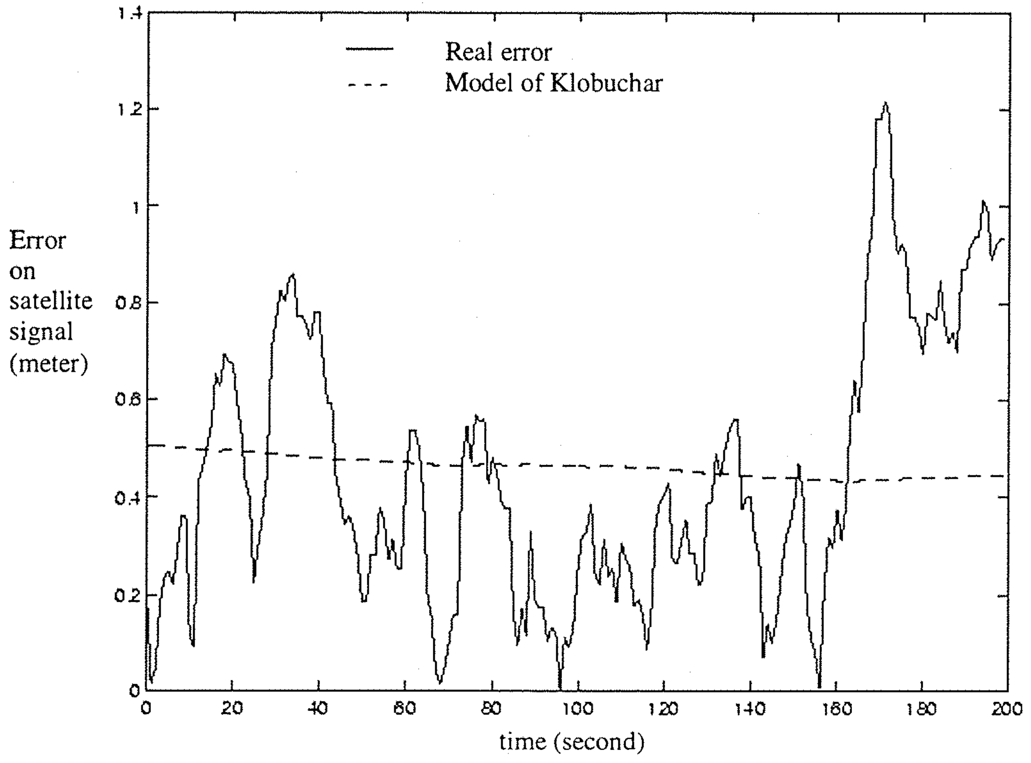


**Figure 4-6- Ionospheric error on satellite number 7**

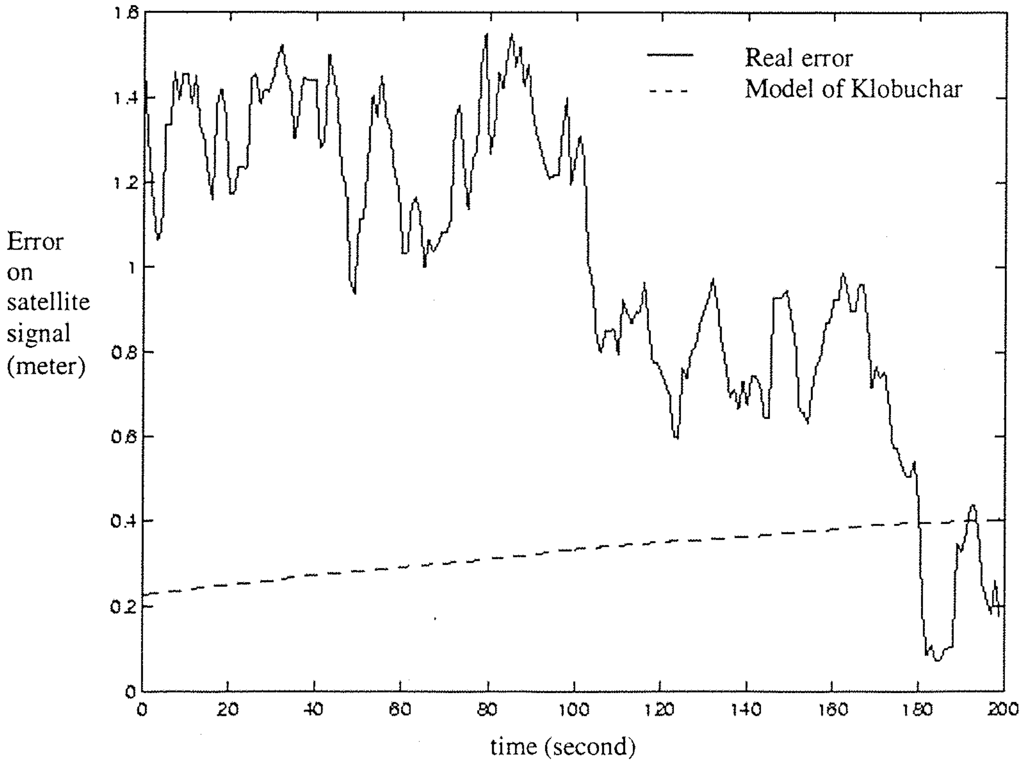


**Figure 4-7- Ionospheric error on satellite number 9**

### 4.3.3 Conclusion

The experimentation done using the model of Klobuchar does not exceed 200 seconds, which is a too short duration to make some definitive conclusions. At least, a duration of few hours would be needed to do so. This lack of long simulation is mainly due to the non availability of long real data files. However, from the previous graphics, the conclusions would be as follows.

The model commonly used today in the GPS receiver has the following characteristics:

- It provides a good approximation of the mean value of the real ionospheric error (commonly no more than 1 meter of difference between the mean real ionospheric delay and the modelled error). It performs the requested performance of correcting 50% of the ionospheric delay.
- When the real ionospheric error variations are weak, it does not take them into account. But if the ionospheric delay variations have greater amplitude, it is able to model them.
- The ionospheric model is unable to model small ionospheric variations.

The low variations of the modelled error can be explained as follows:

- The modelled delay varies as a function of the elevation and azimuth angle of the considered satellite. These two angles do not change rapidly (especially with a static receiver). But if the elevation angle is small (see satellite number 8), each little variation of it will change the value of the modelled error.
- The modelled error also changes as a function of the coefficients $\alpha_n$, and $\beta_n$ broadcast by the satellite. These coefficients are not often refreshed.

Therefore this model is accurate enough to provide a mean correction of the ionospheric error, but does not take into account the real time variations of the ionospheric delay.

*Note*: The ionospheric error given by the white noise in *part 3.6.3* (parametric study) is more important than the real ionospheric delay shown in this section. But the experiment here considered was during low solar activity, which means that the TEC (Total Electron Content) of the ionosphere was low. The parametric study with white noise was to compare the importance of a different kind of error, and so the values were just estimated ones. But in a period of high solar activity, the ionospheric delay can reach this range of values.

## 4.4 EGNOS ionospheric corrections

### 4.4.1 Presentation

#### *4.4.1.1 Generalities*

The EGNOS (European Geostationary Navigation Overlay System) is an augmentation system which eliminates the limitations of GPS and GLONASS related to accuracy and reliability and will make satellite-based radionavigation possible for the civil aviation. In only a few years, EGNOS will be used by aircraft for navigation from oceanic flight down to precision approach [11].

EGNOS network includes

- A ground segment composed of some reference and integrity monitor data processing sites.
- A space segment composed of geostationary satellites, which broadcast to the GPS/EGNOS receiver the EGNOS corrections assessment. The geostationary satellites used are some Inmarsat satellites and GEO satellites.

EGNOS improvements are on the ephemeris, clock and ionospheric error on the GPS satellites signal.

The EGNOS signal is transmitted on the L1 frequency. The EGNOS data stream is transmitted at 500 symbols/second. EGNOS network time is maintained such that the offset from GPS is less than 50 nanoseconds.
In the following parts only the ionospheric correction provided by EGNOS system is considered. EGNOS is planned to provide a correction on the ionospheric error to within 1 meter of the real ionospheric delay for satellites with an elevation angle higher than 20 degrees.

EGNOS uses the algorithm described below. The input data are the vertical ionospheric delay and their accuracy. These data are provided by a model of the ionosphere, which estimate the TEC in each grid point (e.g. [29]). The main aim of the module of DUNSS dedicated to EGNOS is to enable EGNOS designers to choose the most accurate model of ionosphere. With this new module, it is now possible to compare the ionospheric corrections provided by EGNOS using different models of the ionosphere.

The output data are the ionospheric corrections on GPS signal, and their accuracy.

#### *4.4.1.2 Ionospheric Grid Point*

EGNOS provides a vertical ionospheric delay estimate on specified ionospheric grid points (IGPs). In order to facilitate flexibility in the location of these IGPs, a fixed definition of densely spaced IGP locations is used, resulting in a large number of possible IGPs. The density of these predefined IGPs is dictated by the possible large variation of the vertical ionospheric delay during periods of high solar activity, especially at lower latitudes. The IGPs used at any time are based upon the current variation between IGP locations. Since it would be impossible to broadcast IGP delays for all possible locations, a mask is broadcast to define the IGP locations providing the most efficient model of the ionosphere at a time.

The predefined 1808 possible IGP locations, given in latitude and longitude coordinates are shown in *Figure 4-8.* These locations are denser at lower latitudes because the distance represented by degree of longitude becomes smaller at higher latitudes. The IGP grid at the equator has 5 degrees spacing, 10 degrees spacing between 10 and 55 degrees of latitude (North or South), and finally 90 degrees spacing at ±85 degrees of latitude.

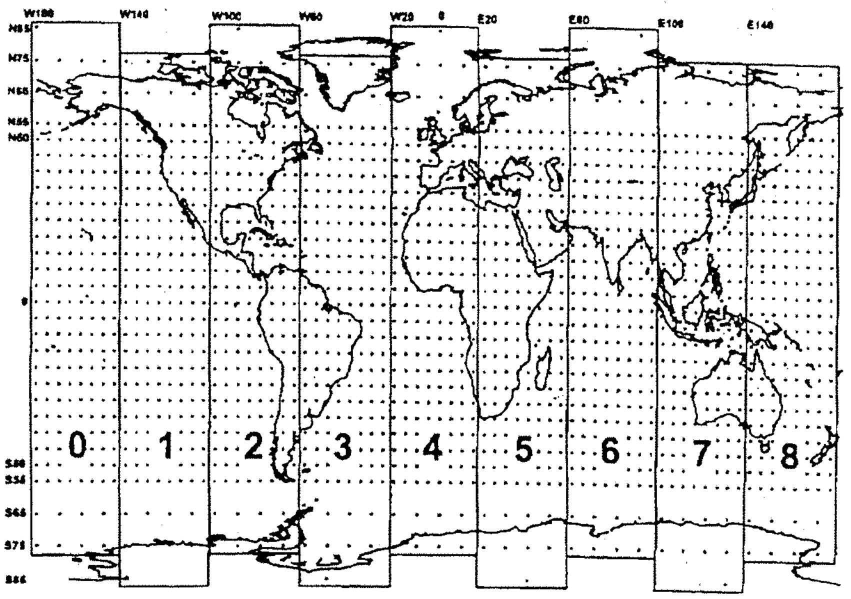


**Figure 4-8- Ionospheric Grid Points**

The total IGP grid represents too many IGPs for broadcasting them in a single message. Therefore, the grid is divided into 9 bands (numbered from 0 to 8, see *Figure 4-8*), and each message indicates the band associated with 201 possible IGPs. Each band covers 40 degrees of longitude. The user has only to collect and save the corrections for IGPs located within about ±20 degrees of his location, which would all be located in one or two bands. A given GEO satellite broadcasts the number of the band in which it provides an ionospheric correction. If the number of band is 0, no ionospheric delay corrections are provided.

Within each band, the IGPs are numbered from 1 to 201 (200 in band 9), counting up from the south west corner up each longitude column of the band (from south to north) and continuing for each column from west to east from the bottom of each column. In the mask, a bit set to one indicates that the ionospheric delay correction information is being provided for the associate IGP. If the bit is set to zero, no ionospheric correction information is provided for that IGP.

### 4.4.1.3 Ionospheric Delay Correction

EGNOS provides on the active IGPs a vertical ionospheric delay correction and their accuracy ($\sigma^2_{GIVE}$). These data are broadcast by the GEO satellite in the message number 26 under the format shown in *Table 4-3*. Each message contains a block number: the 201 IGPs of one band are divided into a maximum of 14 blocks of 15 satellites.

| Field | Field type | Data description | Example | Unit |
|---|---|---|---|---|
| 1 | $DELAY | Log header | $DELAY | |
| 2 | Week | GPS week | 1004 | |
| 3 | second | GPS second in the week | 3777263 | |
| 4 | Band number | Number of the band | 5 | |
| 5 | Block number | Number of the block | 1 | |
| 6 | IODI | Issue of data iono | 0 | |
| 7 | Delay 1 | IGP vertical delay 1 | 3.38 | m |
| ... | ... | ... | ... | ... |
| 21 | Delay 15 | IGP vertical delay 15 | 6.32 | m |

| Field | Field type | Data description | Example | Unit |
|---|---|---|---|---|
| 1 | $GIVE | Log header | $GIVE | |
| 2 | Week | GPS week | 1004 | |
| 3 | second | GPS second in the week | 3777263 | |
| 4 | Band number | Number of the band | 5 | |
| 5 | Block number | Number of the block | 1 | |
| 6 | IODI | Issue of data iono | 0 | |
| 7 | GIVE 1 | IGP correction accuracy 1 | 3.38 | m |
| ... | ... | ... | ... | ... |
| 21 | GIVE 15 | IGP correction accuracy 15 | 6.32 | m |

**Table 4-3- Format of message 26**

### 4.4.2 Algorithm

#### *4.4.2.1 Selection of Ionospheric Grid Points*

For each satellite in view, after determining the location of the pierce point (*Annexe I*), the user must select the IGPs to be used to interpolate the ionospheric correction and model variance.

The following algorithm is used (Reference 11):

1) If the latitude of the pierce point is between –55 and +55 degrees:

- If the associated bits are set to 1, four IGPs that define a 5 degrees by 5 degrees cell around the pierce point are selected, else
- If the associated bits are set to 1, three IGPs that define a 5 degrees by 5 degrees triangle that circumscribes the pierce point are selected, else
- If the associated bits are set to 1, four IGPs that define a 10 degrees by 10 degrees cell around the pierce point are selected. There are four potential 10 degrees by 10 degrees cells that must be checked, any one of which may be used, and we select one without hierarchy. else
- If the associated bits are set to 1, three IGPs that define a 10 degrees by 10 degrees triangle that circumscribes the pierce point are selected, else
- No ionospheric corrections are available

2) If the latitude of the pierce point is between –55 and -75 degrees or between +55 and +75 degrees:

- If the associated bits are set to 1, four IGPs that define a 10 degrees by 10 degrees cell around the pierce point are selected, else

- If the associated bits are set to 1, three IGPs that define a 10 degrees by 10 degrees triangle that circumscribes the pierce point are selected, else
- No ionospheric corrections are available

3) If the latitude of the pierce point is between –75 and -85 degrees or between +75 and +85 degrees:

- The two nearest IGPs at 75 degrees and the two nearest IGPs at 85 degrees are identified. Therefore, a 10 degrees by 10 degrees cell is created by linearly interpolating between the IGPs at 85 degrees to obtain virtual IGPs at longitudes equal to the longitudes of the IGPs at 75 degrees, else
- No ionospheric corrections are available

4) If the latitude of the pierce point is north of 85 degrees south of -85 degrees:

- Four satellites that define a circle that circumscribes the pierce point can be used if the associated bits are set to one, else
- Any three of the four point at 85 degrees that define a semicircle that circumscribes the pierce point can be used if the associated bits are set to one, else
- No ionospheric corrections are available.

### *4.4.2.2 Ionospheric pierce point vertical delay interpolation*

For each in view satellite, the user has to interpolate the vertical ionospheric delay at the pierce point from the IGPs vertical ionospheric delays. The IGPs are selected as described in the paragraph above.

For a four points interpolation the following formula is used:

$$\tau_{vpp}\left(\phi_{pp},\lambda_{pp}\right)=\sum_{i=1}^{4}W_i\left(x_{pp},y_{pp}\right)\tau_{vi}$$

**Equation 4-8-Four point interpolation**

where $\tau_{vi}$ are the broadcast grid point vertical delay values at four corners of the IGP grid, as shown in *Figure 4-9*. $\tau_{vppi}$ is the output value at desired pierce point pp, whose geographical coordinates are $\phi_{pp}, \lambda_{pp}$. The $W_i$ functions are defined as follows:

$$W_1(x,y) = f(x,y)$$
$$W_2(x,y) = f(1-x,y)$$
$$W_3(x,y) = f(1-x,1-y)$$
$$W_4(x,y) = f(x,1-y)$$

with
$$f(x,y) = x^2 * y^2 * (9 - 6*x - 6*y + 4*x*y)$$

The function f(x,y) is weighing polynomial which enables to cover the whole square area [11].

For a pierce point with a latitude between –75 degrees and 75 degrees

$$x_{pp} = \frac{\Delta\lambda_{pp}}{\lambda_2 - \lambda_1} \qquad y_{pp} = \frac{\Delta\phi_{pp}}{\phi_2 - \phi_1}$$

For a pierce point with a latitude north of 85 degrees or south of -85 degrees

$$x_{pp} = \frac{\lambda_{pp} - \lambda_3}{90°} * (1 - 2 * y_{pp}) + y_{pp} \qquad y_{pp} = \frac{|\phi_{pp}| - 85°}{10°}$$

with:

$$\Delta\lambda_{pp} = \lambda_{pp} - \lambda_l \qquad \Delta\phi_{pp} = \phi_{pp} - \phi_l$$

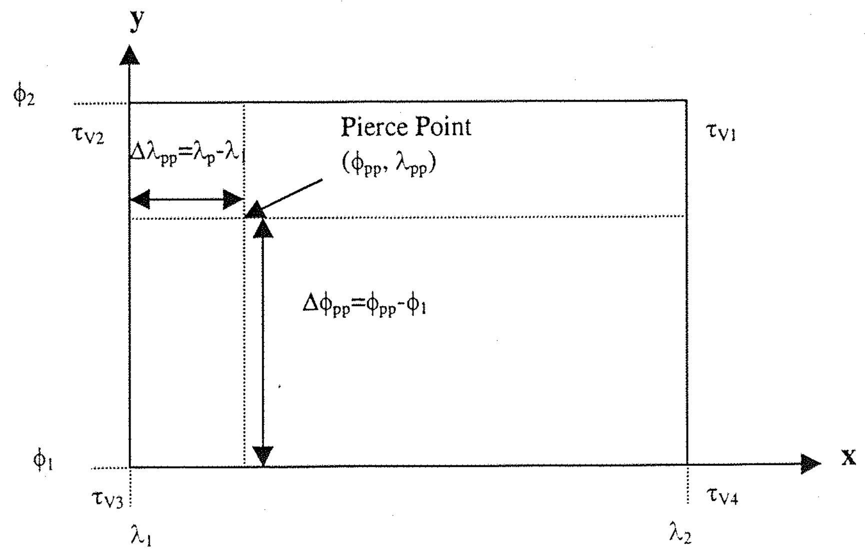


**Figure 4-9- Four points interpolation**

For a three points interpolation between –75 degrees and +75 degrees, the following formula is used:

$$\tau_{vpp}(\phi_{pp}, \lambda_{pp}) = \sum_{i=1}^{3} W_i(x_{pp}, y_{pp})\tau_{vi}$$

**Equation 4-9- Tree points interpolation**

where the terms have the same definition as for four points interpolation, except:

$$W_1(x, y) = f(1, y)$$
$$W_2(x, y) = f(1, 1 - x - y)$$
$$W_3(x, y) = f(x, 1)$$

The pierce point are numbered as shown in the *Figure 4-10*

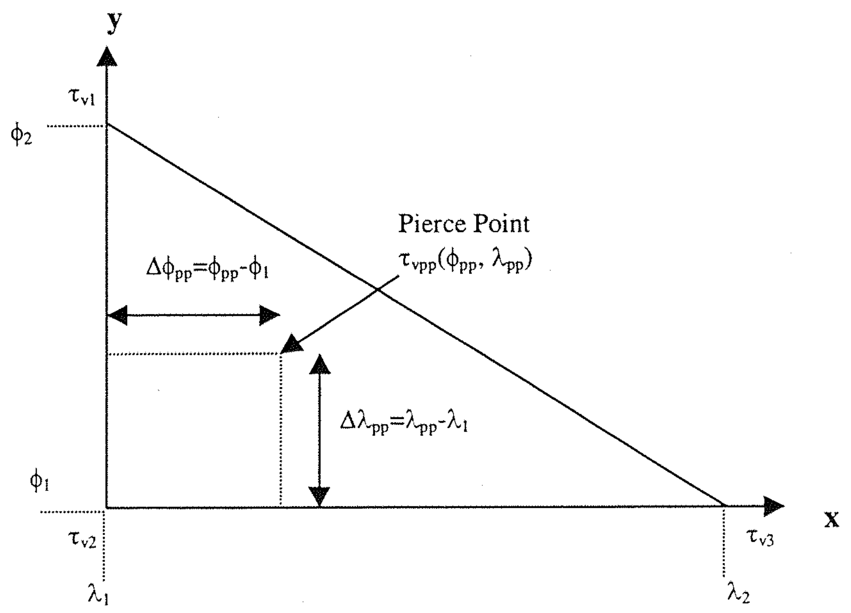

y
$\tau_{v1}$
$\phi_2$
Pierce Point
$\tau_{vpp}(\phi_{pp}, \lambda_{pp})$
$\Delta\phi_{pp}=\phi_{pp}-\phi_1$
$\Delta\lambda_{pp}=\lambda_{pp}-\lambda_1$
$\phi_1$
$\tau_{v2}$
$\lambda_1$
x
$\tau_{v3}$
$\lambda_2$

**Figure 4-10- Three points interpolation**

### 4.4.2.3 Model variance interpolation

The $\sigma^2_{UIVE}$ are interpolated by the user from the $\sigma^2_{IGP}$ defined at IGPs as follows:

$$\sigma^2_{UIVE} = \sum_{i=1}^{4} W_i(x_{pp}, y_{pp})\sigma^2_{i,GIVE}$$ for a four points interpolation.

$$\sigma^2{}_{UIVE} = \sum_{i=1}^{3} W_i(x_{pp}, y_{pp})\sigma^2_{i,GIVE}$$ for a three points interpolation.

**Equation 4-10- Variance interpolation**

#### 4.4.2.4 Slant ionospheric delay and slant model variance

$\tau_{vpp}(\phi_{pp},\lambda_{pp})$ represents the vertical ionospheric delay at the pierce point. To have the ionospheric delay following the line from the receiver to the satellite, it must be multiplied by an obliquity coefficient. The ionospheric correction applied to a satellite signal is then computable with:

$$iono\_delay = -\tau_{spp}(\phi_{pp},\lambda_{pp}) = -F_{pp} * \tau_{vpp}(\phi_{pp},\lambda_{pp})$$

**Equation 4-11- Ionospheric correction**

where $F_{pp} = \left[1-\left(\frac{R_e\cos(E)}{R_e+h_I}\right)^2\right]^{-\frac{1}{2}}$ is the obliquity factor, $R_e$ is the Earth radius, and $h_I$ the mean altitude of the ionosphere (taken equal to 350 km)

The $\sigma^2_{UIRE}$ are computed with:

$$\sigma^2_{UIRE} = F^2_{pp} * \sigma^2_{UIVE}$$

**Equation 4-12- Variance computation**

### 4.4.3 Experimentation

In this part, the values of ionospheric correction provided by EGNOS are compared with the real ionospheric delay, and if it is interesting with the values of ionospheric delay given by Klobuchar model.

The satellite data were acquired on June the 15th 1999 starting at 14:10 (local time), in Alcatel Space Industries at Toulouse. Six satellites are in view from the GPS/EGNOS receiver.

The following figures display the result on the satellite ionospheric error. Unfortunately, the same problem as in paragraph *4.3.2* is encountered: The duration of the simulation is too short to do a full analysis of EGNOS corrections.

The results obtained for satellites number 1, 2, 5, 6 are not really satisfactory. The *Figure 4-11* and *Figure 4-12* show the results for satellites 1 and 2. The EGNOS correction is around 3 meter higher than the real ionospheric delay.

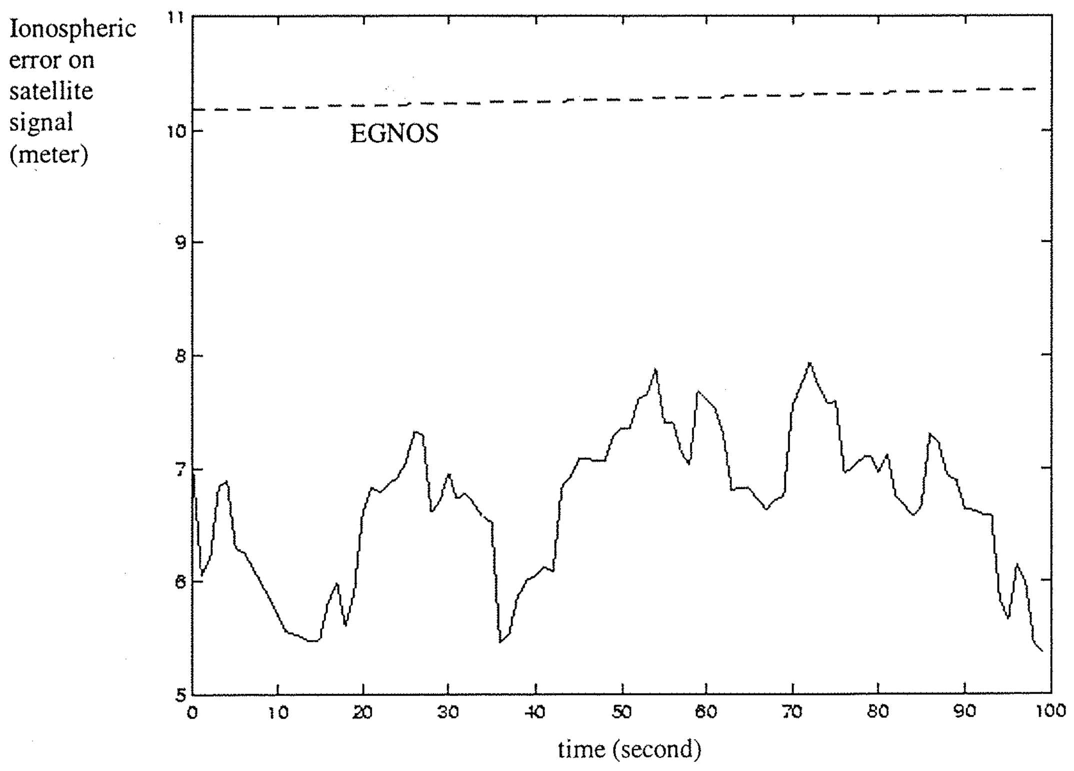


**Figure 4-11- Ionospheric delay on satellite 1**

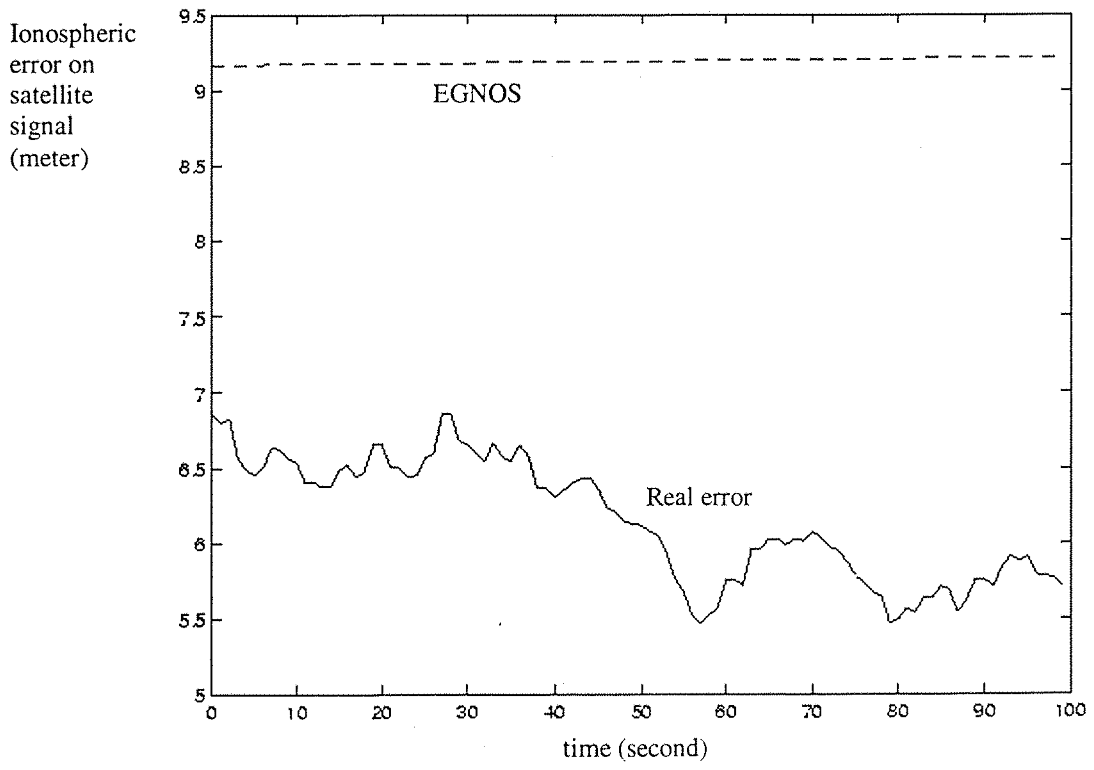


**Figure 4-12- Ionospheric delay on satellite 2**

Considering satellite number 3, we can see that the real ionospheric delay moves between EGNOS corrections and Klobuchar ones. Both of them quite close to the real delay (less than 1.5 meter): see *Figure 4-13.*

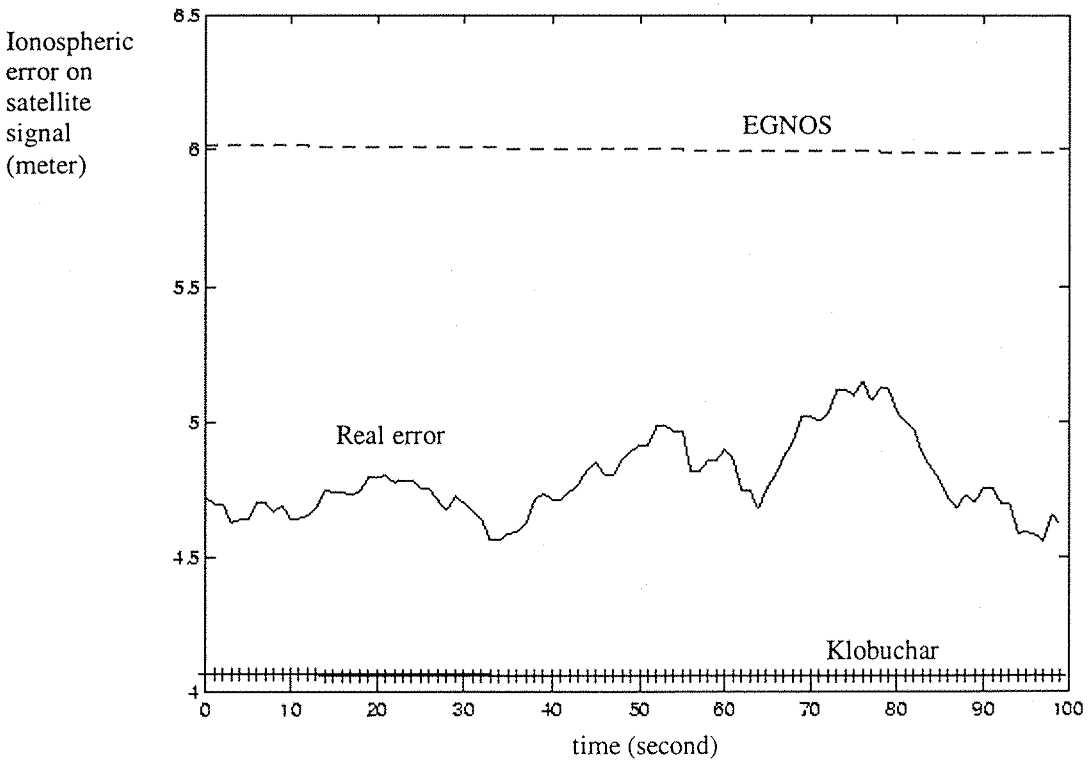


**Figure 4-13- Ionospheric delay on satellite 3**

The best results are obtained for satellite number 4: the EGNOS corrections are less than 1 meter far from the real ionospheric delay, but do not follow the variations of it (see *Figure 4-14*).

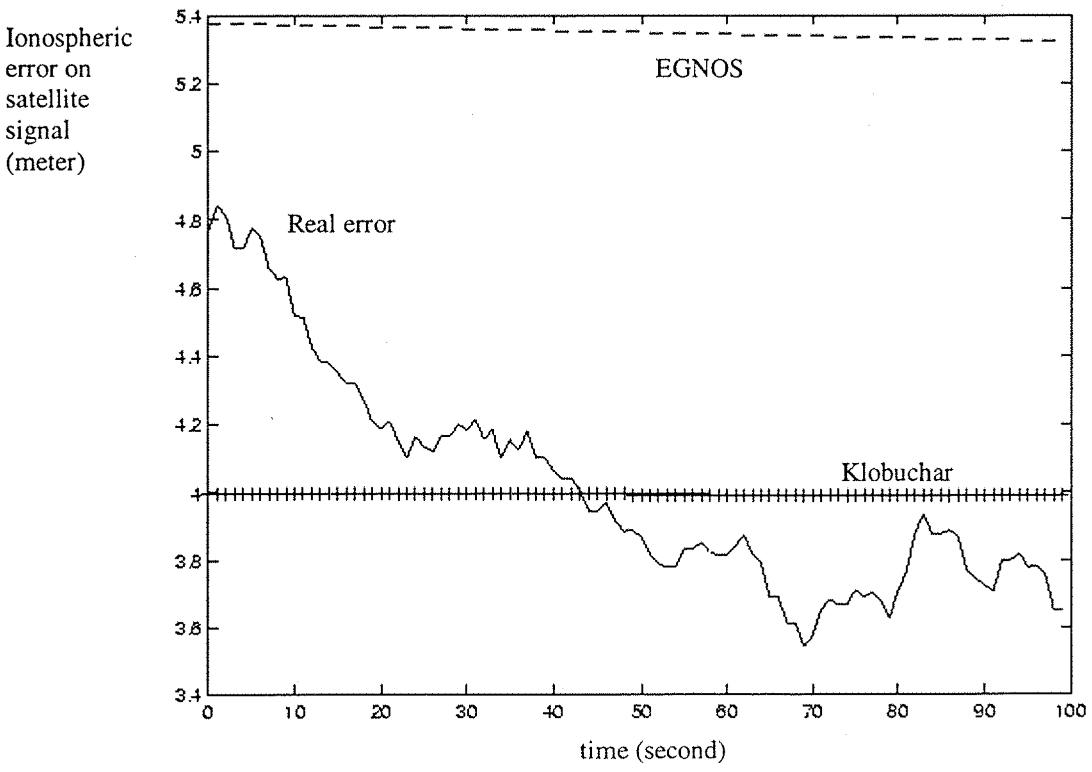


**Figure 4-14- Ionospheric delay on satellite 4**

During this simulation, the accuracy on EGNOS ionospheric correction was equal to 0.3 meters for each pierce point. As shown in the previous figures, the real ionospheric delay is not in this range $[EGNOS\_correction - 0.3, EGNOS\_correction + 0.3]$.

### 4.4.4 Conclusion

As underlined previously, the set of experiments do not last long enough to have a good evaluation of EGNOS corrections. To make some definitive conclusions concerning EGNOS, we need some simulations lasting a few hours, at several levels of solar activity. From the available graphics, we can just deduce that the EGNOS corrections do not seem to be better than corrections provided by the model of Klobuchar. The difference between the mean value of the real ionospheric delay and the mean value of EGNOS correction can be about 3 meters for some satellites.

These results could be explained as follows. EGNOS is in its validation stage. Therefore, its designers are trying different ways to compute the vertical ionospheric delays provided at each point of the ionospheric grid. To compute these values, EGNOS is using the current models of ionosphere, like those developed by CNES in France or DFLR in Germany. These models try to provide the most accurate real time vertical delay in all the points of the ionospheric grid.

The use of a grid makes EGNOS able to be more flexible than the model of Klobuchar. EGNOS can use the current most accurate model of ionosphere, and can change easily from one model to another, in order to have the most accurate values of ionospheric delay.

DUNSS can be a powerful tool to choose between the different available models and to determine which one is the most accurate. To compare two models, the user just needs to enter the vertical ionospheric delays provided by the considered model, and use the general algorithm described in *part 4.4.2* to compute the ionospheric delay on the satellite signal.

EGNOS is planned to provide a correction on the ionospheric error with an accuracy which is between 50 cm and 1 m compared with the real ionospheric delay. The values displayed in the previous simulation show that EGNOS is not yet accurate enough to perform this aim. However, the current studies carried out on this subject should improve EGNOS performances rapidly.

# 5 Conclusions

## 5.1 Future prospects

DUNSS is a basic tool simulating a GPS receiver. A lot of improvements could be done in order to study the influence of other parameters on the computed receiver position accuracy.

First, some other models of ionospheric delay can be implemented, and then compared with Klobuchar and EGNOS corrections (Suard, Lestarquit, Issler, 1997 [12]).

The other errors on satellite signal can also be modelled. EGNOS also provides some correction on ephemeris and clock error. These corrections are easy to add to DUNSS: the program already acquires it when it reads EGNOS files.
Some models of tropospheric delay are also developed.

A receiver environment can also be added to the main program. Trees or clouds can be modelled by noise, and walls or mountains by multipath.

## 5.2 Achievement

The first aim of the work was to write a program able to simulate a GPS receiver, in the most realistic way, in order to study its behaviour.

This part succeeded totally: Alcatel Space Industries has now a tool that computes the navigation solution. It uses satellite data coming from either STK files or real data files. It is able to simulate the error on the satellite signal by white noise, and two other models for the ionospheric delay. The user can change all the parameters he wants, with the aim of studying the possible improvement on a GPS receiver (or GPS/EGNOS).

Moreover, DUNSS is a base for a more elaborate simulator. The possible improvement can be other models to simulate the error on satellite signal, or virtual environment around the receiver (trees, walls, etc...).

The second aim of this thesis was to develop a new tool to test the performances of EGNOS ionospheric corrections. The author developed the algorithm of Klobuchar, and the module dedicated to EGNOS corrections and validated them.

The set of simulations displayed in this thesis does not last on a long enough period to do some definitive conclusions concerning the model of Klobuchar, and the EGNOS corrections. This limitation is mainly due to the lack of long real data files when running the simulator. To carry out a complete study of the ionospheric modelling, some simulations running for a few hours and with different level of solar activity are required.

However, from the simulations using the model of Klobuchar, it could be suggested that this model gives a good first approximation of the real ionospheric delay. The corrections provided compensate for the main part of the ionospheric delay. The model of Klobuchar, however, does not follow the real time ionospheric delay variations. It seems to be a model sufficient for common navigation requirement (leisure navigation, car navigation, aircraft cruise navigation), but not for the next generation of navigation equipment. The GNSS II program is planned to provide an accuracy of 0.5 meter on the ionospheric delay, and about 1 meter on the receiver position. The aim is to use it for very precise navigation, like aircraft landing for example. For this type of use, the model of Klobuchar is not accurate enough. This is one of the reasons that motivated the use of the EGNOS program.

From the simulations shown in this thesis, it can also be seen that the corrections provided by EGNOS are less accurate than those provided by the model of Klobuchar. These results could come from the model of the ionosphere used. This model provides a vertical ionospheric delay at each point of the EGNOS ionospheric grid. The tool required by Alcatel Space Industries and developed by the author is mainly dedicated to choose a model of the ionosphere that will provide the most accurate ionospheric correction on each satellite signal.

As already underline in this thesis, the simulations done using DUNSS could not lead to definitive conclusion concerning the error on satellite signal and especially on ionospheric delay:

- The white noises used to simulate the error on satellite signal in the *part 3.6.3* do not simulate a real case, since no one has a gaussian probability, except the noise on the receiver.
- Due to a lack of long real data files, the simulations done to study the model of Klobuchar and the EGNOS corrections are too short to have a wide survey of the ionosphere activity.

However, the program is ready to run some longer simulations, and is validated to do that. DUNSS will be used during the next months by Alcatel Space Industries in Toulouse to test and validate EGNOS ionospheric corrections. The program could contribute to the improvement of the current satellite navigation system, or to the elaboration of a new one.

# References

**Reference 1**: Logdson Tom, 1986, The NAVSTAR Global Positioning System. Van Nostrand Reinhold.

**Reference 2**: J.J. Spilker Jr, 1996, GPS navigation data. Global Positioning System: Theory and applications, Volume I, chapter 4, pages 121-149. American Institute of Aeronautics and Astronautics Inc.

**Reference 3**: Axelrad P., Brown R.G., 1996, GPS navigation algorithm. Global Positioning System: Theory and applications, Volume I, chapter 9, pages 409-433. American Institute of Aeronautics and Astronautics Inc.

**Reference 4**: Parkinson B.W., 1996, GPS error analysis. Global Positioning System: Theory and applications, Volume I, chapter 11, pages 469-483. American Institute of Aeronautics and Astronautics Inc.

**Reference 5**: Klobuchar J.A., 1996, Ionospheric effects on GPS. Global Positioning System: Theory and applications, Volume I, chapter 12, pages 485-515. American Institute of Aeronautics and Astronautics Inc.

**Reference 6**: Spilker J. J. Jr, 1996, Tropospheric effects on GPS. Global Positioning System: Theory and applications, Volume I, chapter 13, pages 517-546. American Institute of Aeronautics and Astronautics Inc

**Reference 7**: Braasch M.S., 1996, Multipath effects. Global Positioning System: Theory and applications, Volume I, chapter 14, pages 547-599. American Institute of Aeronautics and Astronautics Inc

**Reference 8**: Zumberge J. F., Bertiger W.I., 1996, Ephemeris and clock navigation message accuracy. Global Positioning System: Theory and applications, Volume I, chapter 16, pages 585-599. American Institute of Aeronautics and Astronautics Inc

**Reference 9**: Van Graas F., Braasch M.S., 1996, Selective availibility. Global Positioning System: Theory and applications, Volume I, chapter 17, pages 601-621. American Institute of Aeronautics and Astronautics Inc

**Reference 10**: Collegial work, 1995, Global Positioning System: Specification of SPS Ranging signal Characteristics, pages 40-42. Second edition.

**Reference 11**: Collegial work, 1998, EGNOS specification, Appendix A. RTCA inc.

**Reference 12**: Norbert Suard, Laurent Lestarquit, Jean Luc Issler, 1997, IONO-GPS software: Determination of the ionospheric error using only L1 frequency GPS receiver. GNSS 97, volume II, pages 643-649, Munich. The German Institute of Navigation.

**Reference 13**: Jock R.I. Christie, Bradford W. Parkinson, Per K. Enge, 1997, A proposal signal design for GNSS2 : The use of faster and longer code to provide real-time single frequency ionospheric measurements. GNSS 97 volume I, pages 163-169, Munich. The German Institute of Navigation.

**Reference 14**: S.Schluder, N. Jakowski, and A.Jungstand, 1998, Investigation of the Total Electron Content of the ionosphere during geomagnetic storms. GNSS 98 volume III, pages V-P-07 1-5, Toulouse.

**Reference 15**: Elizabeth Rooney, David Holmes, Mike Trethewey, 1998, Application of a single frequency ionosphere model for resolving GLONASS group delay biases. GNSS 98 volume III, pages V-P-01 1-7, Toulouse.

**Reference 16**: G.H. Golub, C.F. Van Loan, 1989, Matrix computations. Johns Hopkins University Press.

**Reference 17**: T.Kailath, 1977, Linear Least Square Estimation. Dowdon, Hutschinson & Ross.

**Reference 18**: D.E. Wells and al., 1986, Guide to GPS positioning. Canadian GPS associates.

**Reference 19**: P.Y.C. Hwang, R.G. Brown, 1989, GPS navigation: Combining pseudorange with continuous carrier phase using a Kalman filter. Proceedings of ION GPS-89, pages 185-190. Institute of navigation, Washington, Sept 27-29, 1989.

**Reference 20**: J.M. Goodman, J. Aarons, 1990, Ionospheric effects on Modern electronics systems. Proceedings of the IEEE, Volume 78, 1990, pages 512-528.

**Reference 21**: D. Bilitza, K. Rawer, L. Bossy, T. Gulyaeva, 1993, International reference ionosphere – past, present and future: Electron density. Advances in Space research, volume 13, n° 3, 1993, pages (3)3-(3)13.

**Reference 22**: D.J. Crain, J.J. Sojka, R.W. Schunk, P.H Doherty, J.A. Klobuchar, 1993, A first principle derivation of the high latitude total electron content distribution. Radio Science, Volume 28, n° 1, 1993, pages 49-61.

**Reference 23**: J.A. Klobuchar, 1987, Ionospheric time-delay algorithm for single frequency GPS users. IEEE Transactions on Aerospace and Electronic Systems, volume AES-23, n° 3, 1987, pages 325-331.

**Reference 24**: F.K. Brunner, M. Gu, 1991, An improved model for the dual frequency ionospheric correction of GPS observations. Manuscripta Geodaetica, volume 16, 1991, pages 205-214.

**Reference 25**: W.A. Feess, S.G. Stephens, 1987, Evaluation of GPS ionospheric time delay model. IEEE Transaction on Aerospace and Electronic Systems, volume AES-23, n° 3, 1987, pages 332-338.

**Reference 26**: E.H. Martin, 1980, GPS user equipement error models. Global Positioning System Papers, Volume I, 1980, pages 109-118. Institute of Navigation of Washington.

**Reference 27**: Tom Swan, 1995, Borland C++ 4.5. Sams Premier ltd.

**Reference 28**: Bjarne Stroustrup, 1992, The C++ language. Addison-Weisley Publishing Company.

Web sites:

**Reference 29**: http://www.nz.dlr.de/gps/gps-ion.html

**Reference 30**: http://www.oma.be/BIRA-IASB/Project_educatif/Ionosphere/

**Reference 31**: http://karlsberg.usask.ca/~andreas/thesis/

**Reference 32**: http://www.nasa.gov/

**Reference 33**: http://wwwhost.cc.utexas.edu/ftp/pub/grg/gcraft/notes/gps/gps.html

# Annexes

# ANNEXE I

## Computation of the ionospheric pierce point coordinates

The ionospheric pierce point is the point where the straight line between the satellite and the receiver cuts the ionosphere with the highest electron density. The height of the ionosphere area with the higher electron density is commonly taken to 350 km from the Earth surface.

We compute the pierce point coordinates in WGS 84 referential, in (longitude, latitude) coordinates.

First the pierce point latitude is computed as:

$$\phi_{pp} = \arcsin(\sin(\phi_u) * \cos(\psi_{pp}) + \cos(\phi_u) * \sin(\psi_{pp}) * \cos(A)) \qquad \text{in radian}$$

Where $\psi_{pp}$ is the Earth centred angle between the receiver and the satellite (see.*Figure 1*), and is computed as:

$$\psi_{pp} = \frac{\Pi}{2} - E - \arcsin\left(\frac{R_e}{R_e + h_I} * \cos(E)\right) \qquad \text{in radian}$$

The pierce point longitude is computed as:

$$\lambda_{pp} = \lambda_u + \arcsin\left(\frac{\sin(\psi_{pp}) * \sin(A)}{\cos(\phi_{pp})}\right) \qquad \text{in radian}$$

With:

| | |
|---|---|
| A | The azimuth angle of the satellite from the receiver location, in radian. |
| E | the elevation angle of the satellite from the receiver location, in radian. |
| $\phi_u$ | the receiver latitude in radian. |
| $\lambda_u$ | the receiver longitude in radian. |
| $R_e$ | the approximate radius of the earth ellipsoid, equal to 6378136.3 meters |

$h_I$ the approximate altitude of the ionospheric peak of electron density, equal to 350 km.

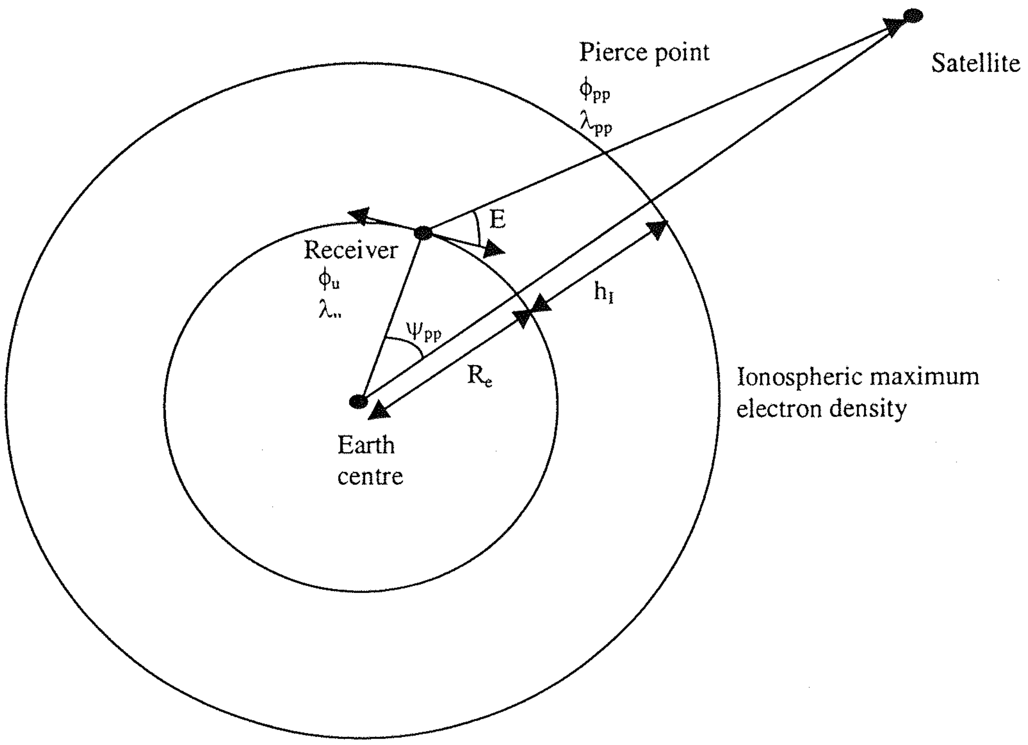


**Figure 1**

# ANNEXE II

# Earth model

## 1. Presentation of two models

To compute the elevation and azimuth angles of the in view satellites, or to compute the receiver longitude and latitude from the cartesian coordinates, we need to use an Earth representation.

The simplest model is a spherical one: The Earth is modelled as a sphere with a radius of 6378137 meters.

The earth model used in WGS 84 is an ellipsoid, in order to take into account the flattening of the Earth. The Earth is an ellipsoid with a long semi-axis a=6378137 meters and a small semi-axis b=6356752 meters (see *Figure 1*).

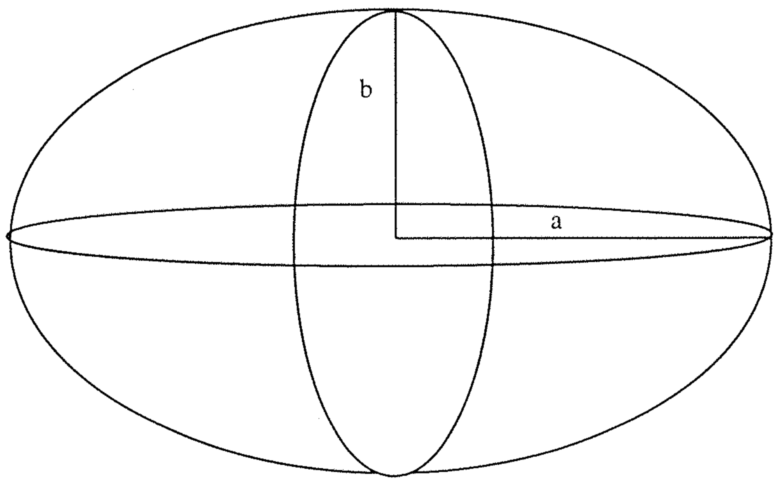
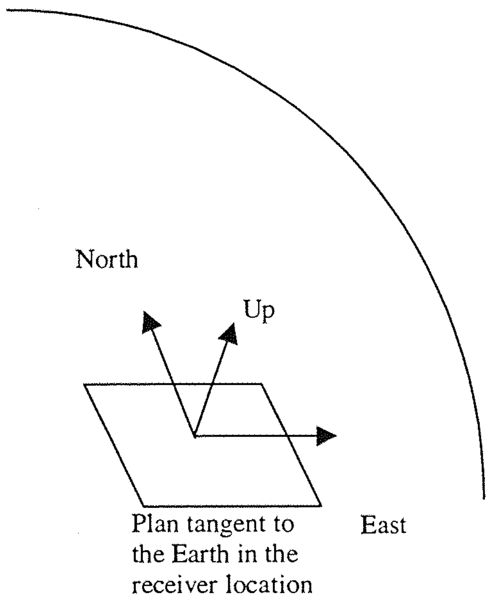


**Figure 1-Earth ellipsoid model**

## 2. Computation of the elevation and azimuth angles

To compute the elevation and azimuth angles of the in view satellites, we need to compute the local reference at the receiver location. This local receiver is defined by $\overrightarrow{North}, \overrightarrow{East}, \overrightarrow{Up}$ with:

- $\overrightarrow{North}$ unitary, oriented to the local north, in the plan tangent to the Earth.
- $\overrightarrow{East}$ unitary, oriented to the local east, in the plan tangent to the Earth.
- $\overrightarrow{Up}$ unitary, perpendicular to the plan tangent to the Earth.



The following equations describe the computation of the three vectors in both models:

In both models, the vectors are given by:

$$\overrightarrow{North}[0] = -\sin(\phi)*\cos(\lambda) \qquad \overrightarrow{East}[0] = -\sin(\phi) \qquad \overrightarrow{Up}[0] = \cos(\phi)*\cos(\lambda)$$

$$\overrightarrow{North}[1] = -\sin(\phi)*\sin(\lambda) \qquad \overrightarrow{East}[1] = \cos(\phi) \qquad \overrightarrow{Up}[1] = \cos(\phi)*\sin(\lambda)$$

$$\overrightarrow{North}[2] = \cos(\phi) \qquad \overrightarrow{East}[2] = 0 \qquad \overrightarrow{Up}[2] = \sin(\phi)$$

In spherical model, $\phi$ is the longitude, and $\lambda$ the latitude of the receiver (true position).

In ellipsoid model, $\phi$ is given by: $\arctan\left(\frac{Z_{receiver}}{(1-e^2)*\sqrt{X_{receiver}^2+Y_{receiver}^2}}\right)$

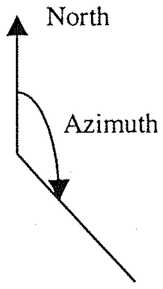
and $\lambda$ is given by: $\arctan\left(\frac{Y_{receiver}}{X_{receiver}}\right)$; e is the eccentricity of the ellipsoid: $e^2 = 1-\frac{b^2}{a^2}$

In both cases, the elevation and azimuth angles are computed with:

$$elevation = \arcsin\left(\vec{V}.\overrightarrow{Up}\right) \text{ and } azimuth = \arctan\left(\frac{\vec{V}.\overrightarrow{East}}{\vec{V}.\overrightarrow{North}}\right) \text{ modulo } 2\pi$$

with $\vec{V} = \overrightarrow{User - satellite}$

The azimuth angle is counted positively from the north vector:



The values given by the two models can be compared in the following table. The values are given in degrees, and the error on elevation and azimuth represent the absolute value of the difference between the spherical and ellipsoid values.

| n° of satellite | spherical elevation | ellipsoid elevation | error on elevation | sperical azimuth | ellipsoid azimuth | error on azimuth |
|---|---|---|---|---|---|---|
| 1 | 77.32 | 77.50 | 0.18 | 345.52 | 345.31 | 0.21 |
| 2 | 8.50 | 8.40 | 0.10 | 237.75 | 237.72 | 0.03 |
| 3 | 18.71 | 18.83 | 0.12 | 52.51 | 52.56 | 0.05 |
| 4 | 11.87 | 11.95 | 0.08 | 293.12 | 293.08 | 0.04 |
| 5 | 51.63 | 51.51 | 0.12 | 130.78 | 130.96 | 0.18 |
| 6 | 39.00 | 38.82 | 0.08 | 174.50 | 174.51 | 0.01 |
| 7 | 41.73 | 41.80 | 0.07 | 67.83 | 67.99 | 0.16 |
| 8 | 0.39 | 0.56 | 0.17 | 30.14 | 30.14 | 0.00 |
| 9 | 40.90 | 40.99 | 0.09 | 299.48 | 299.33 | 0.15 |

We can see that on this example the gain on the angles using an ellipsoid Earth model is not really important (around 0.1 degree). But this small difference can have a great influence on some computation, particularly for the computation of ionospheric delay, at low elevation angle (satellite 8).

The *Figure 2* and *Figure 3* show the difference between the spherical and ellipsoid values of elevation and azimuth angle compared with the values provided by a dual frequency GPS receiver during a simulation (simulation done by Alcatel Space Industries in Toulouse).

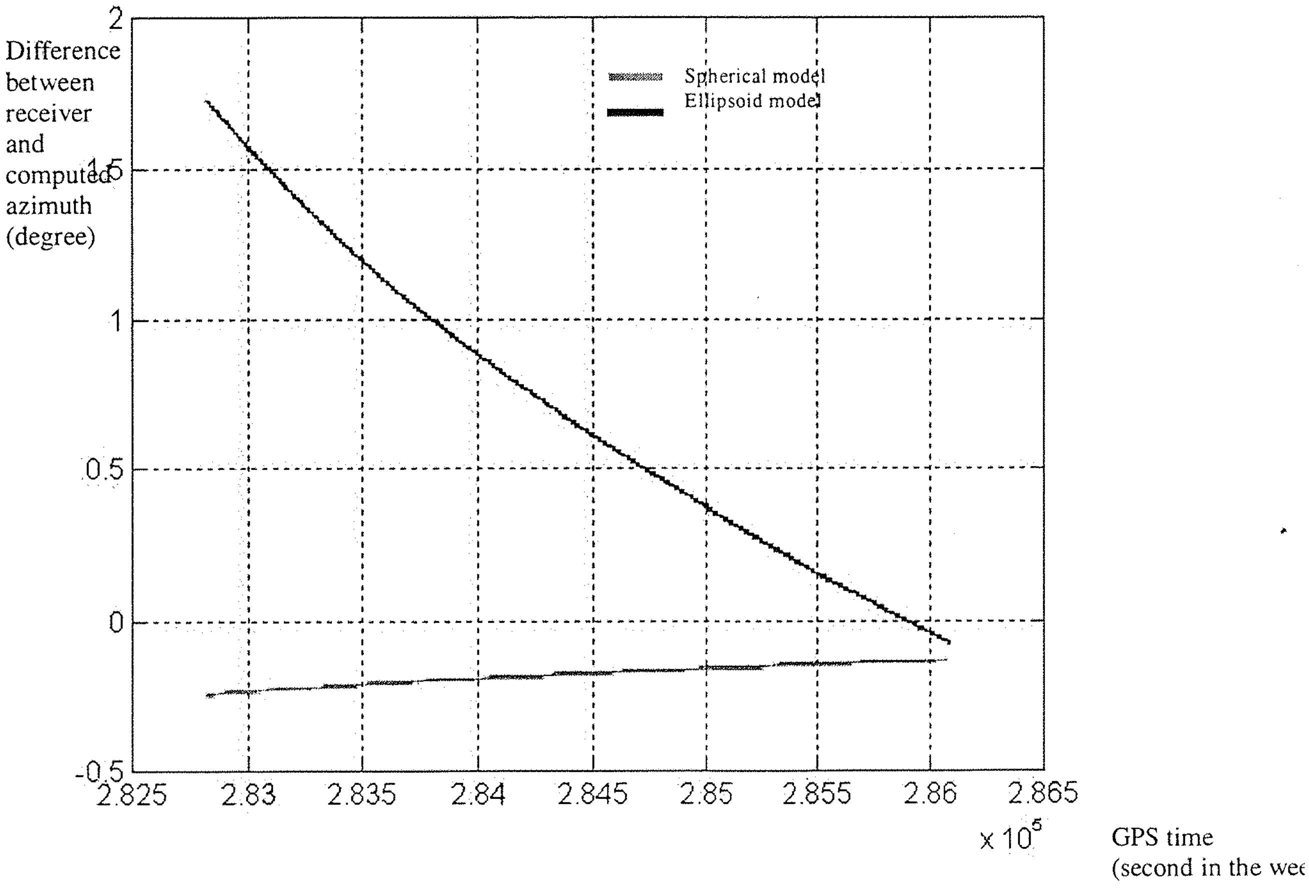


**Figure 2- Difference between receiver azimuth and computed azimuth**

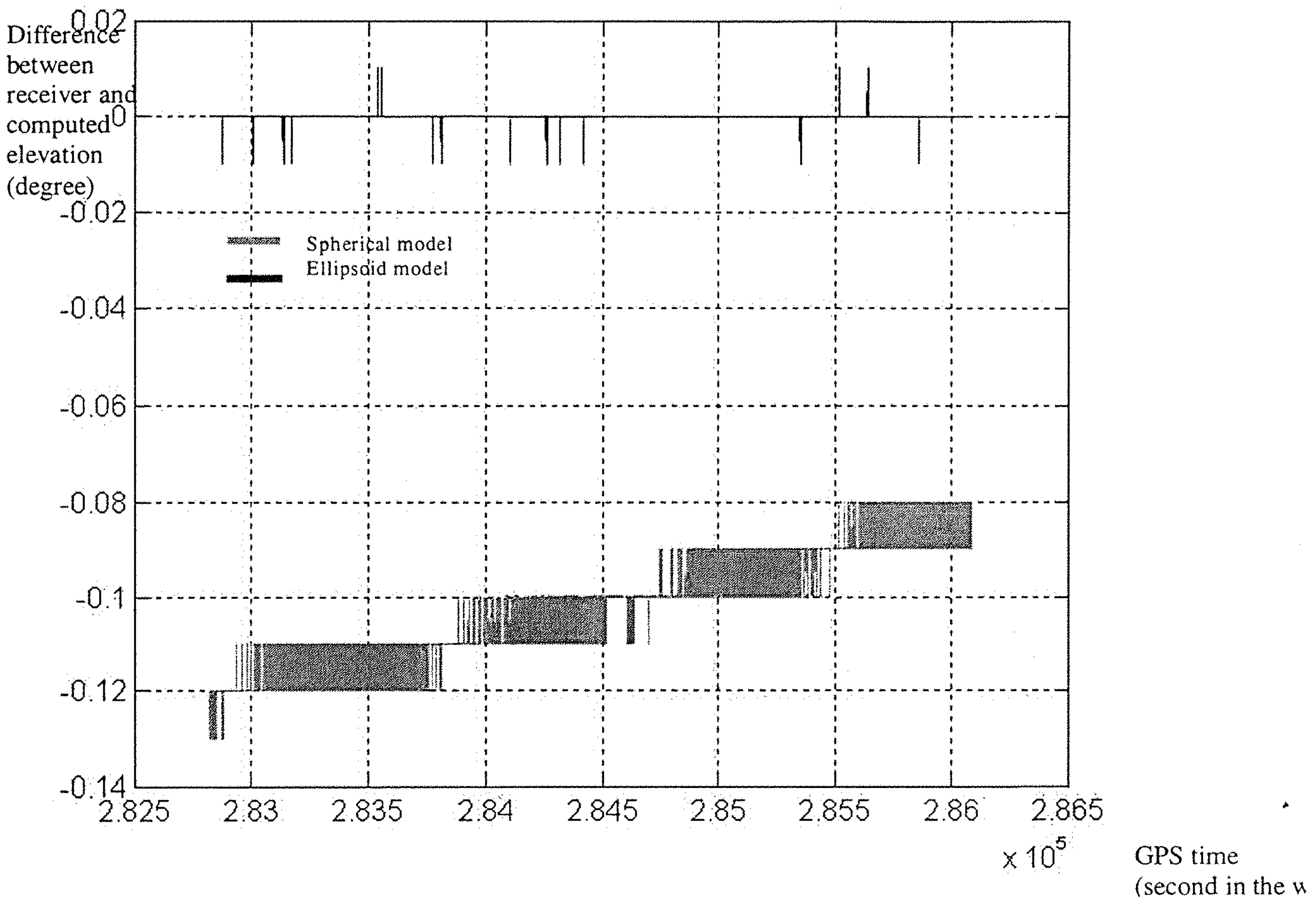


**Figure 3-Difference between receiver elevation and computed elevation**

We can see on this example, that the elevation angle provided by the ellipsoid model is closer to the value provided by the receiver than the elevation provided by the spherical model. For the azimuth, it is the contrary.

## 3. Computation of the receiver longitude and latitude

To compute the receiver longitude and latitude from the cartesian coordinates, we can use the both models.

With the spherical model, values are given by:

$$longitude = \arctan\left(\frac{Y_{receiver}}{X_{receiver}}\right) \text{ and } latitude = \left(\frac{Z_{receiver}}{\sqrt{X_{receiver}^2 + Y_{receiver}^2}}\right)$$

But if we want to use more accurate values, we can use the ellipsoid Earth model.

$$longitude = \arctan\left(\frac{Y_{receiver}}{X_{receiver}}\right) \text{ and } latitude = \arctan\left(\frac{Z_{receiver}}{(1-e^2)*\sqrt{X_{receiver}^2 + Y_{receiver}^2}}\right)$$

with:

a=6378137 meters, long semi-axis of the ellipsoid

b=6356752 meters, small semi-axis of the ellipsoid

$e^2 = 1 - \frac{b^2}{a^2}$, eccentricity of the ellipsoid

For the position of the GPS receiver located in Alcatel Space Industries in Toulouse, the coordinates in WGS 84 are: X=4629365.4041, Y=112099.4237, Z=4371618.9541.

The two models give the same value of longitude (same formula used).
We have the values of latitude given by both models:

| | spherical | ellipsoid |
|---|---|---|
| Latitude (degree) | 43.3514 | 43.5435 |

We gain 0.2 degree, which is not really important, but can be useful for really precise computation.

## 4. Conclusion

In order to have the most accurate values, we use the ellipsoid model of the Earth. The gain on the accuracy is not primordial, but it does not cost many more operation than using the spherical model.

# ANNEXE III
# PROGRAM DETAILS

## 1 Generalities

The constants used in the program DUNSS, are listed in *Table 1*.

| Name | Description | Value | Unit |
|---|---|---|---|
| NB_SAT | Maximum number of satellites used for the simulation | 70 | |
| NB_POINT | Maximum duration of the simulation | 512 | second |
| NB_ITE | Number of iterations between two acquisitions | 2 | |
| NB_FILE_STK | Maximum number of STK files used | 27 | |
| R_EARTH | Radius of the Earth | 6378140 | meter |
| R_e | Estimated radius of the Earth ellipsoid | 6378136.3 | meter |
| c | Speed of the light | 299792458 | m/s |
| pi | pi | 3.14159265359 | |
| Sin_Mask_Angle | Sinus of the minimum angle of satellite signal acquisition | 0.0871557427477 | |
| El_Ref | Angle of reference | 0.349065850399 | rad |
| L1 | Frequency 1 of GPS signal | 1575420000 | Hz |
| L2 | Frequency 2 of GPS signal | 1227600000 | Hz |
| NB_BAND | Number of band used to divide the world map for EGNOS | 9 | |
| NB_SAT_IN_BAND | Number of satellites in each EGNOS band | 201 | |
| | | | |

**Table 1- Constants used in DUNSS**

Two main classes are created for the program:

- The class CUser, which consists of the user position and time (true and estimated) and the user bias.
- The class CSatellite, which consists of the satellite position and time, and also of the "in view" parameter.

Some other classes are also used in each sub-routine.

All the floating-point values are in the format double.

All matrix used in the program (G, W, Error) are arrays [NB_SAT][NB_SAT] of double, even if most of their coefficients are nil.

## 2 Database Management function

The database Management function can be found in "in_data.C" file.
The Database management function prints the question on the screen to make the user enter the input parameters. All the input parameters are stored into general variables.

## 3 Satellite Data Acquisition function

The satellite Data Acquisition function is in "read.C" file.

### 3.1 *Satellite data coming from real data files*

The real satellite data are stored in different files:

- *Log_n.txt*: This file contains the $RGEA and $SVDA data (RGEA: pseudorange. SVDA: satellite position and error corrections). The program reads it to gain the satellite time, position, and correction on ionospheric error. DUNSS also read the pseudoranges broadcast on L1 and L2 (in RGEA) in order to compute the real ionospheric error.

- *Log_mik.ion*: This file contains the $ALMA and $IONA data (ALMA: almanach of the satellite. IONA: ionospheric coefficients used to compute the ionospheric delay using the Klobuchar model).
- *Corr03.txt*: This file contains the EGNOS corrections used to compute the ionospheric delay with EGNOS.

The structure of these files is describes in the *Annexe V.*

DUNSS reads the real data files at the beginning of the simulation and store the values into some arrays. Therefore, the Satellite Data Acquisition function is launched only once during a simulation.

## 3.2 *Satellite data coming from STK*

STK provides one file per satellite used in its simulation. The data stored in this file consists of the data of time, position and speed. These data are stored under the following format:
Time(\t)position_X(\t)position_Y(\t)position_Z(\t)speed_X(\t)speed_Y(\t)speed_Z(\t)
With \t a tabulation.

The Satellite Data Acquisition function opens one file per satellite used during the DUNSS simulation. Afterward, the values of the time, position on X, position on Y, position on Z are stored into an array [NB_POINT][NB_SAT] of class CSatellite, for all the duration of the simulation. Therefore, the Satellite Database Acquisition function is launched only once during a DUNSS simulation: just after the Database Management function. We choose to use only STK simulation with a time step of 1 minute, to avoid storing huge files. Therefore, the Satellite Database Acquisition function interpolates the satellite position and time between two minutes, and stores them in the same array. The interpolation is a linear interpolation.

The Satellite Database Acquisition function must be modified at each modification of STK simulation: It is written for a fixed number of satellites. If a new simulation use more or less satellites than the previous one, the function must be modified to open the files corresponding with these satellites, and only these ones.

## 4 Navigation Solution Computation function

### 4.1 *Co-ordinates computation*

The coordinates computation is done by a function of "func_glob.C".
*Annexe II* describes the choice of the Earth model used to compute the angles needed during the computation: receiver latitude and longitude, elevation and azimuth angles for each satellite.

### 4.2 *In view satellite computation*

The determination of the satellites in view from the receiver is done by a function of "func_glob.C".
To compute the satellites in view, we just need to compute the elevation angle of the vector $\overrightarrow{receiver\_true - satellite_j}$ and to compare it with the chosen mask angle, and particularly to compare the sinus of the two angles. The mask angle is chosen in order to avoid taking the signal from satellites with a low elevation angle, which have a more important error on their signal (ionospheric and tropospheric delay).

### 4.3 *G computation*

G is computed by a function of "compute_k.C" file.
To compute G, we use the formula:

$$G=\begin{bmatrix} \frac{-\overrightarrow{X}_1[0]}{\left\|\overrightarrow{X_1}\right\|} & \frac{-\overrightarrow{X}_{1i}[1]}{\left\|\overrightarrow{X_1}\right\|} & \frac{-\overrightarrow{X}_1[2]}{\left\|\overrightarrow{X_1}\right\|} & 1 \\ \vdots & \vdots & \vdots & \vdots \\ \frac{-\overrightarrow{X}_k[0]}{\left\|\overrightarrow{X_k}\right\|} & \frac{-\overrightarrow{X}_k[1]}{\left\|\overrightarrow{X_k}\right\|} & \frac{-\overrightarrow{X}_k[2]}{\left\|\overrightarrow{X_k}\right\|} & 1 \\ 0 & 0 & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

with $\overrightarrow{X}_i$ the vector receiver_estimated-satellite$_i$, and k the number of in view satellites.

## 4.4 *W computation*

W is computed by a function of "compute_k.C" file.

To compute W we use the formula:

$$W=\begin{bmatrix} w_1 & 0 & & \cdots & & 0 \\ 0 & \ddots & & & & \\ & & w_k & \ddots & & \vdots \\ \vdots & & \ddots & 1 & & \\ & & & & \ddots & 0 \\ 0 & & \cdots & & 0 & 1 \end{bmatrix}$$

with k the number of in view satellites, and

$$w_i=\frac{1}{\sigma_{ephemeris}^2+\sigma_{clock}^2+\sigma_{noise}^2+\sigma_{iono}^2\div\sin^2(El_i)+\sigma_{tropo}^2\div\sin(El_i)+\sigma_{mupath}^2\div\tan^2(El_i)}$$

where El$_i$ is the elevation angle of the satellite number i from the true receiver position, $i\in[1..k]$.

The sigma are the estimated sigma, which are supposed to be as close as possible from the real values.

We use another weighting matrix with EGNOS simulation, as described in *part 4-4*.

## 4.5 Error matrix computation

The matrix, error, is computed in "err_sat.C" file.

To compute the error matrix (matrix of the errors put on the satellites signal), we use the following formula:

$$Error = \begin{bmatrix} ephemeris_1 + clock_1 + iono_1 + tropo_1 + mupath_1 + noise_1 \\ \vdots \\ ephemeris_k + clock_k + iono_k + tropo_k + mupath_k + noise_k \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

with k the number of in view satellites.

The terms of the different types of error are computed with the models chosen by the user among those proposed.

## 4.6 K computation

K is computed by a function of "compute_k.C" file.

To compute the K matrix, we use the following formula:

$K = \left(G^T W G\right)^{-1} G^T W$

## 4.7 Pseudorange residual computation

The pseudorange is computed by a function of "compute_k.C" file.

We compute the pseudorange residual with the following formula:

$\Delta\rho = G\left(x\_estimated - x\_true\right) - Error$

## 4.8 Estimated receiver position

The estimated receiver position is computed by a function of "compute_k.C" file.

We first compute the estimated delta x:

$\Delta\bar{x} = K\Delta\rho$

Afterwards, we can compute the new estimated receiver position:

$\bar{x} = \bar{x} - \Delta\bar{x}$

## 5 Error Computation function

The Error Computation function is one function of "out_data.C" file.
The Error Computation function computes, at each computation step, the error between the computed receiver position and the true one. It stores this value in an array [4] for the error on X, Y, Z and time. And it returns the general geometric error ($= \sqrt{(error\_x)^2 + (error\_y)^2 + (error\_z)^2}$).

## 6 Display function

The Display function is included in "out_data.C" file.
The display function is composed of 2 main functions:

- The storage function store at each second the values of the receiver position and error on these positions (X, Y, Z, time and global geometric error) in some array [NB_VALUES]. This function is called at each computation step.
- The global storage function, which prints the following valued into some files: receiver position (on X, Y, Z), error on the receiver position (on X, Y, Z, global error), satellites in view from the receiver, number of satellites in view from the receiver, Azimuth and Elevation angle of all the satellites, error on all the in view satellites signal (6 types of error, more the total error). The values are stored so that they can be used by Matlab: The files are .m files, the values are stored into some vectors, and the command lines for Matlab are print in the file. Writing the name of one of these files under a Matlab session will draw automatically the curve of the data stored into this file in function of the time.

## 7 Main function

The main function is in "DUNSS.C" file.

The main function calls the functions of the program into two loops:

- One loop on the acquisition step (one acquisition every second).
- One loop on the iteration step, between each second.

The structure can be summarised as follows:

```
{
Database Management;
Acquisition of data from STK or real data files;
Display_init;
Loop on the acquisition
    {
    Computation of the in view satellites;
    Computation of the error put on the satellite signals;
    Loop on iteration
            {
            Computation of G;
            Computation of W;
            Computation of K;
            Computation of the new estimated receiver position;
            Computation of the error between the estimated and the true user position.
            Storage of the results;
            }
    }
    }
Global storage;
}
```

# ANNEXE IV
# User handbook

## 1. Starting DUNSS

DUNSS is written in order to be used under UNIX systems.
To start DUNSS, the user has to write the UNIX command line DUNSS.out in the directory called DUNSS.

## 2. Entering input data

The first screen presents the program. The user has to press a key to continue.

The user has to choose how he wants to enter the input data. He has two choices:

- Read a predefined scenario stored in *scenario_1.txt*, and whom the user can modify the values.
- Enter the input data now.

In both cases, the user has a warning message telling him that he has to check if the satellite data files (STK files, or real data files) are compatible with the input parameters of the simulation.

If he chooses to read *scenario_1.txt*, the pre-computing stage is ended.

If he chooses to enter the input parameters, he has to follow the printed instructions in order to enter:

- Duration of the simulation.
- True initial position of the user.
- Estimated initial position of the user.

- Equation of the receiver trajectory.
- Choice of the errors put on the satellite signals.

The pre-computing stage is then over.

## 3. Reading of the results

To read the results, the user can open the files:

- *Satel_in_view.txt* which contains the satellites in view at each step of the simulation.
- *Results.txt* which contains the values of DOP, and the average of the error on X, Y, Z, T, and global error for the receiver position.

To visualise the graphics of errors on satellite signals, satellite elevation and azimuth angles, error on receiver position, and receiver position, the user has to launch a Matlab simulation in the directory DUNSS.

Under this cession, he can:

- Type *error_satel_i* to visualise the error on the satellite number i (all the types of error will be drawn, the user has to modify the Matlab file if he wants only one or two type(s) of error).
- Type *el_az_i* to visualise the elevation and azimuth angles of the satellite i in function of the time.
- Type *error_X*, *error_Y*, *error_Z*, *error_T*, or *error_glob* to visualise the error on the receiver position on X, Y, Z, T or global error in function of the time.
- Type *posi_X*, *posi_Y*, *posi_Z* to visualise the position on X, Y, or Z of the user in function of the time.

# ANNEXE V

# Satellite data processing

## 1. Generalities

All the satellite data are stored under the format *double*, except for the data of time and of satellite number which are stored under the format of *integer.*

Each satellite broadcasts a prn number, which an identity number. The prn number are divided as follows (international convention):

- Between 1 to 37: GPS satellites.
- Between 38 to 61: GLONASS satellites.

In fact, in the simulation used, only GPS satellites (prn between 1 and 32) are used.

When the Satellite Data Acquisition function read the STK files or real data files, all the satellite data are stored in a class CSatellite, marked with the satellite prn number.

But, just before the beginning of the navigation solution computation, all these data are re-affected in class CSatellite marked from 0 to the number of satellites in view.

For example, if the satellites number 2, 15, 17, and 21 are in view, their data will be processed under the number 0, 1, 2, 3 during all the computation.

After the simulation, the user can read which satellite was in view reading the *satel_in_view.txt* file. But, he has to be careful concerning the Matlab files: they have a number between 0 and the number of satellites in view during the simulation. Therefore, the user has to make the link between this number and the real number of the satellite.

Anyway, the Matlab graphic presents in his title the prn number of the satellite, so that the confusion can be avoid.

## 2. Format of the real data files

The file *log.txt* contains two messages that interest us: the $SVDA and $RGEA messages.

The structure of each one is described below:

| Field | Field type | Data description | Example | Unit |
|---|---|---|---|---|
| 1 | $SVDA | Log header | $SVDA | |
| 2 | week | GPS week number | 766 | |
| 3 | second | GPS seconds in the week | 143860.00 | sec |
| 4 | rec clk err | Solved receiver clock error | -4.062 | m |
| 5 | obs | Number of satellites observed | 7 | |
| 6 | prn | Satellite prn number (1-32) | 20 | |
| 7 | x | Satellite x coordinate | -15044774.225 | m |
| 8 | y | Satellite y coordinate | -9666598.520 | m |
| 9 | z | Satellite z coordinate | 19499537.398 | m |
| 10 | clk corr | Satellite clock correction | 6676.013 | m |
| 11 | ion corr | Ionospheric correction | -1.657 | m |
| 12 | trop corr | Tropospheric correction | -2.662 | m |
| 13 | diff corr | Differential correction used | 16.975 | |
| 14 | rng std | Range weight std deviation | 0.674 | |
| 15… | | Next satellites… | | |
| variable | *xx | checksum | *23 | |
| variable | [CR][LF] | Sentence terminator | [CR][LF] | |

**Figure 1- $SVDA structure**

| Field | Field type | Data description | Example | Unit |
|---|---|---|---|---|
| 1 | $RGEA | Log header | $RGEA | |
| 2 | week | GPS week number | 663 | |
| 3 | second | GPS seconds in the week | 247893.30 | sec |
| 4 | obs | Number of satellites observed | 7 | |
| 5 | rec status | Receiver self test status | 000040F6 | |
| 6 | prn | Satellite prn number (1-32) | 26 | |
| 7 | per | Pseudorange measurement | 23704623.130 | m |
| 8 | psr std | Pseudorange measurement std deviation | 0.148 | m |
| 9 | adr | Carrier phase | -124567967.330 | cycle |
| 10 | adr std | Estimated carrier phase deviation | 0.010 | cycle |
| 11 | C/N0 | Signal to noise density ratio | 43.0 | |
| 12 | locktime | Number of second of continuous tracking | 2693.370 | sec |
| 13 | tr status | Phase lock, channel number and channel state | E04 | |
| 14 | mg std | Range weight std deviation | 0.674 | |
| 15-23 | | Same data at L2 | | |
| ... | ... | Next prn range measurment | | |
| variable | *xx | checksum | *73 | |
| variable | [CR][LF] | Sentence terminator | [CR][LF] | |

**Figure 2- $RGEA structure**

The *log_mik.ion* file contains only one message interesting for the simulation: the $IONA message.

| Field | Field type | Data description | Example | Unit |
|---|---|---|---|---|
| 1 | $IONA | Log header | $RGEA | |
| 2 | a0t | Alpha constant term | 1.0244548320770265e-08 | sec |
| 3 | a1t | Alpha 1st order term | 1.4901161193847656e-08 | sec/semic |
| 4 | a2t | Alpha 2nd order term | -5.960464477539061e-08 | sec/semic$^2$ |
| 5 | a3t | Alpha 3rd order term | -1.1920928995507812e-07 | sec/semic$^3$ |
| 6 | b0t | Beta constant term | 8.8064000000000017e04 | sec |
| 7 | b1t | Beta 1st order term | 3.2768000000000010e04 | sec/semic |
| 8 | b2t | Beta 2nd order term | -1.9660800000000001e05 | sec/semic$^2$ |
| 9 | b3t | Beta 3rd order term | -1.9660800000000001e05 | sec/semic$^3$ |
| 10 | *xx | checksum | *02 | |
| 11 | [CR][LF] | Sentence terminator | [CR][LF] | |

**Figure 3- $IONA structure**

# ANNEXE VI
# Files organisation

## 1. General overview

The *Figure 1* shows which header files are called by the other header files.

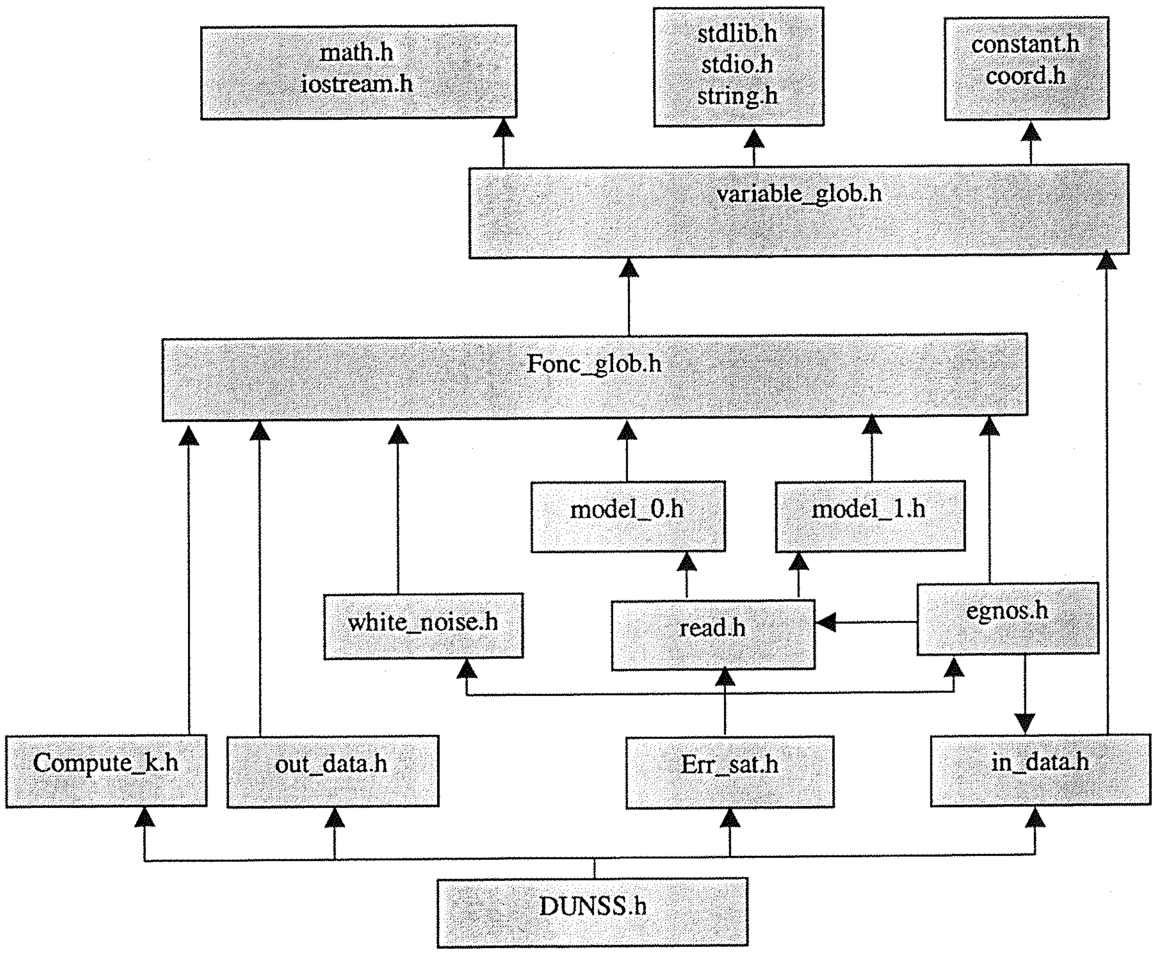


**Figure 1- Header files calling**

We now list the functions file by file.

## 2. Functions listing

| Name | Input parameters | Output parameters | Function |
|---|---|---|---|
| main | ∅ | ∅ | General computation, master function |

**Table 1- Function of "DUNSS.C"**

| Name | Input parameters | Output parameters | Function |
|---|---|---|---|
| Compute_G_0 | Integer: computation step | ∅ | Computation of matrix, G |
| Compute_W_0 | Integer: computation step | ∅ | Computation of matrix, W |
| Compute_K_0 | ∅ | ∅ | Computation of matrix, K |
| Compute_G_1 | Integer: computation step | ∅ | Computation of matrix, G |
| Compute_W_1 | Integer: computation step | ∅ | Computation of matrix, W |
| Compute_K_1 | ∅ | ∅ | Computation of matrix, K |
| uere | ∅ | Double:$\sigma_{uere}$ | Computation of a reference σ for the error on satellite signal |
| Compute_A | ∅ | ∅ | Computation of $(G^TG)^{-1}$ |
| Compute_DOP | Integer: computation step | ∅ | Computation of GDOP, PDOP, HDOP, VDOP, XDOP, YDOP, TDOP |
| Compute_expected_error | Integer: computation step | ∅ | Computation of the estimated expected error on the receiver position |

**Table 2- Functions of "compute_K.C"**

| Name | Input parameters | Output parameters | Function |
|---|---|---|---|
| Read_data | ∅ | ∅ | Read satellite data in the files "log_n.gps" and "log_mik.ion" |
| Read_stk | ∅ | ∅ | Read satellite data coming from STK simulation |
| Interpolation | Integers: time step, satellite number | ∅ | Computation of a linear interpolation of the satellite position between two minutes |

**Table 3- Functions of "read.C"**

| Name | Description |
|---|---|
| CAlma_Iona | Parameters contained in $Alma and $Iona message: GPS week and seconds, satellite prn, alpha and beta used in the model of Klobuchar to compute the ionospheric delay. Function to read this parameters: lecture. |
| CRgea_Svda | Parameters contained in $Rgea and $Svda messages: GPS week and seconds, satellite prn, number of observed satellite, the pseudorange on L1 and L2, the satellite position in WGS 84, and some corrections on the ionospheric delay, tropospheric delay and clock error. Function to read this parameters: lecture. |

**Table 4- Classes of "read.C"**

| Name | Input parameters | Output parameters | Function |
|---|---|---|---|
| Input_data | ∅ | ∅ | Acquisition of the user choices for input parameters |
| Lecture_scen1 | ∅ | ∅ | Lecture of the file "scenario_1" which contain a set of input parameters |

**Table 5- Functions of "in_data.C"**

| Name | Input parameters | Output parameters | Function |
|---|---|---|---|
| Read_data | ∅ | ∅ | Read satellite data in the files "log_n.gps" and "log_mik.ion" |
| Read_stk | ∅ | ∅ | Read satellite data coming from STK simulation |
| Interpolation | Integers: time step, satellite number | ∅ | Computation of a linear interpolation of the satellite position between two minutes |

**Table 3- Functions of "read.C"**

| Name | Description |
|---|---|
| CAlma_Iona | Parameters contained in $Alma and $Iona message: GPS week and seconds, satellite prn, alpha and beta used in the model of Klobuchar to compute the ionospheric delay.<br>Function to read this parameters: lecture. |
| CRgea_Svda | Parameters contained in $Rgea and $Svda messages: GPS week and seconds, satellite prn, number of observed satellite, the pseudorange on L1 and L2, the satellite position in WGS 84, and some corrections on the ionospheric delay, tropospheric delay and clock error.<br>Function to read this parameters: lecture. |

**Table 4- Classes of "read.C"**

| Name | Input parameters | Output parameters | Function |
|---|---|---|---|
| Input_data | ∅ | ∅ | Acquisition of the user choices for input parameters |
| Lecture_scen1 | ∅ | ∅ | Lecture of the file "scenario_1" which contain a set of input parameters |

**Table 5- Functions of "in_data.C"**

| Name | Input parameters | Output parameters | Function |
|---|---|---|---|
| Global_storage | ∅ | ∅ | Storage of the receiver position on X, Y, Z, error on X, error on Y, error on Z, and global error (in referential WGS 84) in some Matlab files |
| Write_error_ satellite | ∅ | ∅ | Storage of the error on satellite signal in Matlab files. Storage of the six types of error: ephemeris, clock, ionospheric, tropospheric, multipath and noise, more the total error |
| Storage | Integer: computation step | ∅ | Storage at each computation step of the user position on X, Y, and Z. |
| Compute_error | Integer: computation step | ∅ | Computation and storage at each time step of the error on the receiver position on X, Y, Z, T and global error. |
| Numeric_output | ∅ | ∅ | Computation and storage in text file of numeric values: Satellites in view at each computation step, average and standart deviation on the receiver position error (on X, Y, Z, T and global error), DOP at each computation step |

**Table 6- Functions of "out_data.C"**

| Name | Input parameters | Output parameters | Function |
|---|---|---|---|
| Compute_error_ iono_0 | Integer: computation step | ∅ | Computation of the dual frequency ionospheric delay |
| Compute_error_ iono_2 | Integer: computation step | ∅ | Computation of the ionospheric delay using the model of Klobuchar |
| Compute_error_ satel | Integer: computation step | ∅ | Computation of the total error on satellite signal. |

**Table 7- Functions of "er_sat.C"**

| Name | Description |
|---|---|
| CGauss | Definition and computation of a gaussian distribution. |
| CUere | Definition of the six types of error on satellite signal. |
| CUereAssess | Computation of the six types of error on satellite signal using the gaussian distribution. |

**Table 8- Classes of "white_noise.C"**

| Name | Description |
|---|---|
| CIono_0 | Definition of the pseudoranges on L1 and L2.<br>Computation of the ionospheric delay with dual frequency data. |

**Table 9- Class of "model_0.C"**

| Name | Description |
|---|---|
| CIono_1 | Definition of alpha and beta coefficients.<br>Computation of the ionospheric delay using the model of Klobuchar. |

**Table 10- Class of "model_1.C"**

| Name | Description |
|---|---|
| CEgnos | Definition of grid point, vertical ionospheric delay, and accuracy on it. |

**Table 11- Class of "egnos.C"**

| Name | Input parameters | Output parameters | Function |
|---|---|---|---|
| Compute_error_egnos | Integer: computation step | ∅ | Computation of the ionospheric delay using EGNOS data and algorithm. |
| Remplissage_egnos | ∅ | ∅ | Acquisition of EGNOS data. |

**Table 12- Functions of "egnos.C"**

| Name | Input parameters | Output parameters | Function |
|---|---|---|---|
| sqr | Double: x | Double: x*x | Square value computation of a double. |
| sqr_n | Double:x<br>Integer: n | Double: $x^n$ | Computation of a double to the power of an integer. |
| coslongi | Double: X, Y, Z | Double: longitude cosinus | Computation of the longitude cosinus of a point from its cartesian coordinates. |
| sinlongi | Double: X, Y, Z | Double: longitude sinus | Computation of the longitude sinus of a point from its cartesian coordinates. |
| coslati | Double: X, Y, Z | Double: latitude cosinus | Computation of the latitude cosinus of a point from its cartesian coordinates. |
| sinlati | Double: X, Y, Z | Double: latitude sinus | Computation of the latitude sinus of a point from its cartesian coordinates |

| | | | |
|---|---|---|---|
| longi | Double: X, Y, Z | Double: longitude | Computation of the longitude of a point from its cartesian coordinates. |
| lati | Double: X, Y, Z | Double: latitude | Computation of the latitude of a point from its cartesian coordinates. |
| transposition | Matrix NB_SAT * NB_SAT Integer: number of lines and columns | ∅ | Computation of the transposition of a square matrix NB_SAT*NB_SAT. |
| addition | 2 Matrix NB_SAT * NB_SAT Integer: number of lines and columns | ∅ | Computation of the sum of two square matrix NB_SAT*NB_SAT. |
| multiplication | 2 Matrix NB_SAT * NB_SAT Integer: number of lines and columns | ∅ | Computation of the product of two square matrix NB_SAT*NB_SAT. |
| inversion | Matrix NB_SAT * NB_SAT | ∅ | Compuation of the inverse of a square matrix NB_SAT*NB_SAT. |
| affichage | Matrix NB_SAT * NB_SAT Integer: n | ∅ | Display of a matrix NB_SAT * NB_SAT, called matrix number n. |
| Satel_in_view | Integer: k | ∅ | Test to which satellites are in view from the receiver at the |

| | | | |
|---|---|---|---|
| | | | computation step k. |
| Compute_el_az | Integer: k | ∅ | Computation of the elevation and azimuth angle of all the in view satellites at the computation step k, using a spherical Earth's model. |
| Compute_el_az_ellipt | Integer: k | ∅ | Computation of the elevation and azimuth angle of all the in view satellites at the computation step k, using an elliptic Earth's model. |
| Pierce_point | Integers: i, k | Doubles: pierce point latitude and longitude | Computation of the pierce point latitude and longitude for the satellite number i at the computation step k. |
| Compute_f | Doubles: x, y | Double: f(x, y) | Computation of a weighting function used in EGNOS (*part 4.4.2.2*). |
| Write_affectation_0 | ∅ | ∅ | Storage of the value of Δx (difference between estimated and true receiver position). |
| Write_affectation_1 | ∅ | Integer: k | Storage of the error on satellite signal in a matrix NB_SAT*NB_SAT at the computation step k. |
| Write_affectation_2 | ∅ | Integer: k | Storage of the estimated and true receiver position at the computation step k. |
| Write_affectation_3 | ∅ | Integer: k | Storage of the estimated and true pseudoranges at the computation |

| | | | |
|---|---|---|---|
| | | | step k. |
| Write_affectation_4 | ∅ | Integer: k | Re-numbering of the satellites in view. |

**Table 13- Functions of "func_glob.C"**

# ANNEXE VII

## Program codes

```
#ifndef _CONSTANT_
#define _CONSTANT_

const int NB_SAT=70;
//nb max of used satellites
const double R_EARTH=6378140;
//Earth's radius in meter
const double R_e=6378136.3;
//Earth's ellipsoid radius
const double hI=350000;
//mean height of ionosphere

const double Sin_Mask_Angle=0.0871557427477;
//sinus of the mask angle (minimum elevation angle) taken to 5
degrees
const double El_Ref=0.349065850399;
//angle of reference in radians, equal to 20 degrees

const double c=299792458;
//speed of the light in the vacuum
const double pi=3.14159265359;
//pi

const int NB_POINT=512;
//number maximum of points in the simulation
const int NB_ITE=2;
//number of iteration between two acquisitions
const int NB_FILE_STK=27;
//maximum number of STK satellites used
const int NB_BAND=9;
//number of bands in the EGNOS ionospheric grid
const int NB_SAT_IN_BAND=201;
//number of satellites in each band of EGNOS ionospheric grid

const double L1=1575420000;
//first frequency of GPS signal
const double L2=1227600000;
//second frequency of GPS signal

#endif _CONSTANT_
```

```
#ifndef _COORD_
#define _COORD_

class CUser
{
public:
      double posi_true_X;
      double posi_true_Y;
      double posi_true_Z;
      double posi_true_T;
      double bias_true;
      double posi_estimated_0_X;
      double posi_estimated_0_Y;
      double posi_estimated_0_Z;
      double posi_estimated_0_T;
      double posi_estimated_1_X;
      double posi_estimated_1_Y;
      double posi_estimated_1_Z;
      double posi_estimated_1_T;
      double bias_estimated;
      double time;
};

class CSatellite
{
public:
      double posi_X;
      double posi_Y;
      double posi_Z;
      double El;
      double Az;
      double time;
      int view;
};

#endif
```

```
#ifndef _VARIABLE_GLOB_
#define _VARIABLE_GLOB_

#include <iostream.h>
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include "coord.h"
#include "constant.h"

typedef double Square_Mat_NB_SAT[NB_SAT][NB_SAT];

typedef struct
{
double lati_pp;
double longi_pp;
}SCoord_pp;

FILE *File_Iono_0;
FILE *File_Iono_1;

int time_GPS;
int time_sec_GPS;
int duration;
int choice_iono, choice_tropo, choice_ephe,
choice_clock,choice_noise, choice_mupath;
int choice_satel_data, choice_scen;

double error_iono_0[NB_SAT][NB_POINT];
double error_iono_1[NB_SAT][NB_POINT];
double error_iono_2[NB_SAT][NB_POINT];
double error_iono_3[NB_SAT][NB_POINT];
double correction_iono[NB_SAT][NB_POINT];
double error_iono_egnos[NB_SAT][NB_POINT];
double error_tropo_0[NB_SAT][NB_POINT];
double error_ephe_0[NB_SAT][NB_POINT];
double error_clock_0[NB_SAT][NB_POINT];
double error_mupath_0[NB_SAT][NB_POINT];
double error_noise_0[NB_SAT][NB_POINT];
double error_satel_0[NB_SAT][NB_POINT];
double error_satel_1[NB_SAT][NB_POINT];

double sigma_ephemeris;
double sigma_clock;
double sigma_iono;
double sigma_mupath;
double sigma_tropo;
double sigma_noise;

int view_during_simu[NB_SAT];
int number_satel_in_view;

CUser user;
CSatellite satel[NB_SAT][NB_POINT];

double user_satel_estimated_0[NB_SAT][4];
double user_satel_estimated_0_local[NB_SAT][3];
double user_satel_estimated_1[NB_SAT][4];
double user_satel_true[NB_SAT][4];
double user_satel_true_local[NB_SAT][3];

Square_Mat_NB_SAT A, G_0, W_0, K_0, G_1, W_1, K_1, sum, product,
trans;
Square_Mat_NB_SAT error_NB_SAT_0, error_NB_SAT_1,
delta_x_estimated_0_NB_SAT, delta_x_estimated_1_NB_SAT, delta_ro_0,
delta_ro_1;

double delta_x_estimated_0[4];
double delta_x_estimated_1[4];

double posi_estimated_1_X[NB_POINT], posi_estimated_1_Y[NB_POINT],
posi_estimated_1_Z[NB_POINT], posi_estimated_0_X[NB_POINT],
posi_estimated_0_Y[NB_POINT], posi_estimated_0_Z[NB_POINT];

double error_0_x[NB_POINT], error_0_y[NB_POINT], error_0_z[NB_POINT],
error_0_t[NB_POINT], error_1_x[NB_POINT], error_1_y[NB_POINT],
error_1_z[NB_POINT], error_1_t[NB_POINT], error_global_1[NB_POINT],
error_global_0[NB_POINT];

double GDOP[NB_POINT], PDOP[NB_POINT], HDOP[NB_POINT],
VDOP[NB_POINT], XDOP[NB_POINT], YDOP[NB_POINT], TDOP[NB_POINT];

double expected_error_x[NB_POINT], expected_error_y[NB_POINT],
expected_error_z[NB_POINT], expected_error_t[NB_POINT],
expected_error_global[NB_POINT];

Square_Mat_NB_SAT Ref_true, Ref_estimated_0, Inv_Ref_true,
Inv_Ref_estimated_0;



//double duration1=200;

#endif _VARIABLE_GLOB_
```

```
#ifndef _FONC_GLOB_
#define _FONC_GLOB_

#include <iostream.h>
#include <math.h>
#include "variable_glob.h"

double sqr(double x);
double sqr_n(double x, int n);
double coslongi(double X, double Y, double Z);
double sinlongi(double X, double Y, double Z);
double coslati(double X, double Y, double Z);
double sinlati(double X, double Y, double Z);
double longi(double X, double Y, double Z);
double lati(double X, double Y, double Z);
void transposition(Square_Mat_NB_SAT M, int line, int col);
void addition(Square_Mat_NB_SAT M,Square_Mat_NB_SAT N, int line, int
col);
void multiplication(Square_Mat_NB_SAT M,Square_Mat_NB_SAT N, int
line, int col, int middle);
void inversion(Square_Mat_NB_SAT N);
void affichage(Square_Mat_NB_SAT M, int k);
void satel_in_view(int k);
void compute_el_az(int k);
SCoord_pp pierce_point(int i, int k);
double compute_f(double x, double y);
void compute_el_az_ellipt(int k);
void write_affectation_0();
void write_affectation_1(int k);
void write_affectation_2(int k);
void write_affectation_3(int k);
void write_affectation_4(int k);


#endif _FONC_GLOB_
```

```c
#include "fonc_glob.h"

double sqr(double x)
      {
      return x*x;
      }

double sqr_n(double x, int n)
      {
      int k;
      double aux=1;

      for(k=0; k<n; k++)
      aux=aux*x;

      return (aux);
      }

double coslongi(double X, double Y, double Z)
            {
            return (X/(sqrt(sqr(X)+sqr(Y))));
            }

double sinlongi(double X, double Y, double Z)
            {
            return (Y/(sqrt(sqr(X)+sqr(Y))));
            }

double coslati(double X, double Y, double Z)
            {
            return (sqrt((sqr(X)+sqr(Y))/(sqr(X)+sqr(Y)+sqr(Z))));
            }

double sinlati(double X, double Y, double Z)
            {
            return (Z/sqrt(sqr(X)+sqr(Y)+sqr(Z)));
            }

double longi(double X, double Y, double Z)
{
      long double longi;

      longi=acos(coslongi(X,Y,Z));
      sinlongi(X,Y,Z)<0 ? longi=-longi : longi=longi;

      return(longi);
}

double lati(double X, double Y, double Z)
{
      long double lati;

      lati=acos(coslati(X,Y,Z));
      sinlati(X,Y,Z)<0 ? lati=-lati : lati=lati;

      return(lati);
}
```

```c
void transposition(Square_Mat_NB_SAT M, int line, int col){
      int i,j;

      for (i=0;i<=(line-1);i++)
            for (j=0;j<=(col-1);j++)
            trans[i][j]=M[j][i];
      }

//function computing the sum of two matrix [NB_SAT][NB_SAT]
//parameters: two Square_Mat_NB_SAT stores into sum

void addition(Square_Mat_NB_SAT M,Square_Mat_NB_SAT N, int line, int
col)
      {
      int i,j;

      for (i=0;i<=(line-1);i++)
      for (j=0;j<=(col-1);j++)
      sum[i][j]=M[i][j]+N[i][j];
      }

//function computing the product of two matrix [NB_SAT][NB_SAT]
//parameters:two Square_Mat_NB_SAT        stores into product

void multiplication(Square_Mat_NB_SAT M,Square_Mat_NB_SAT N, int
line, int col, int middle)
      {
      int i,j,k;
      Square_Mat_NB_SAT vase;

      for(i=0;i<=(NB_SAT-1); i++)
            for (j=0;j<=(NB_SAT-1);j++)
                  product[i][j]=0;

      for(i=0;i<=(line-1); i++)
            {
            for (j=0;j<=(col-1);j++)
                  {
                  vase[i][j]=0;
                  for (k=0;k<=(middle-1);k++)
                        vase[i][j]=vase[i][j]+M[i][k]*N[k][j];
                  product[i][j]=vase[i][j];
                  }
            }
      }

//function computing the inverse of a matrix [4][4]
//parameter:Square_Mat_NB_SAT store into M

void inversion(Square_Mat_NB_SAT N){
      int i,j,k,l,l2,icol,irow;
      double big, temp, zaza, pivinv,aux;
      int indxc[4];
int indxr[4];
int ipiv[4];
double M[4][4];

//M receive the sub-matrix [4][4] to be inversed
for (i=0; i<=3; i++)
      for (j=0; j<=3; j++)
      M[i][j]=N[i][j];

//initial value of the ipiv
for(j=0; j<=3; j++) ipiv[j]=0;

//pivot research
for(i=0; i<=3; i++)
{
big=0.0;
for (j=0; j<=3; j++)
      if (ipiv[j] != 1) for (k=0; k<=3; k++)
                  {
                        if (ipiv[k] == 0)
                        {
                              if (fabs(M[j][k]) >= big)
                              {
                              big=fabs(M[j][k]);
                              irow=j;
                              icol=k;
                              }
                        }
                  else if (ipiv[k] > 1) cout<<"\n\nerror1\n\n";
                  }
      ++(ipiv[icol]);


if (irow != icol)
for (l=0; l<=3; l++)
{
aux=M[icol][l];
M[icol][l]=M[irow][l];
M[irow][l]=aux;
}

indxr[i]=irow;
indxc[i]=icol;

if (M[icol][icol] ==0) cout<<"\n\n error2\n\n";
pivinv=1.0/M[icol][icol];
M[icol][icol]=1.0;
for (l=0; l<=3; l++) M[icol][l] *= pivinv;

for (l2=0; l2<=3; l2++)
      if (l2 != icol)
            {
            zaza=M[l2][icol];
            M[l2][icol]=0.0;
            for (l=0; l<=3; l++) M[l2][l] -= M[icol][l]*zaza;
            }
}
```

```
      for (l=3; l>=0; l--)
            {
            if (indxr[l] != indxc[l])
            for (k=0; k<=3; k++)
                  {
                  aux=M[k][indxr[l]];
                  M[k][indxr[l]]=M[k][indxc[l]];
                  M[k][indxc[l]]=aux;
                  }
            }

      for (l=0; l<=3; l++)
            {
            ipiv[l]=0;
            indxr[l]=0;
            indxc[l]=0;
            }

      //N receive the inversed sub-matrix [4][4]
      for (i=0; i<=3; i++)
            for (j=0; j<=3; j++)
            N[i][j]=M[i][j];
}


void affichage(Square_Mat_NB_SAT M, int k){
      int i,j;

      cout<<"\n\nmatrice numero"<<k<<"\n";

      for (i=0; i<=(number_satel_in_view-1); i++)
            {
            for (j=0; j<=(number_satel_in_view-1); j++)
            cout<<"M["<<i<<"]["<<j<<"]="<<M[i][j]<<"  ";
            cout<<"\n";
            }
      cout<<"\n";
      }


void satel_in_view(int k)
      {
      int j;

      number_satel_in_view=0;

      for(j=0; j<=(NB_SAT-1); j++)
      {
      if (sinlati(user_satel_true_local[j][0],
user_satel_true_local[j][1],
user_satel_true_local[j][2])>Sin_Mask_Angle)
      {
      satel[j][k].view=1;
      satel[j][k].El=lati(user_satel_true_local[j][0],
user_satel_true_local[j][1], user_satel_true_local[j][2]);
      satel[j][k].Az=longi(user_satel_true_local[j][0],
user_satel_true_local[j][1], user_satel_true_local[j][2]);
```

```
      number_satel_in_view++;
      }
      }
      }


void compute_el_az(int k)
      {
      int j,i;
      double N1, N2, N3, aux1, aux2, aux3;
      double East[3];
      double North[3];
      double Up[3];
      double V[3];
      double elevation, azimuth;

      N1=sqrt(sqr(user.posi_estimated_0_X)+sqr(user.posi_estimated_0_
Y));
N2=sqrt(sqr(user.posi_estimated_0_X)+sqr(user.posi_estimated_0_Y)+sqr
(user.posi_estimated_0_Z));

      North[0]= -
(user.posi_estimated_0_X*user.posi_estimated_0_Z)/(N1*N2);
      North[1]= -
(user.posi_estimated_0_Y*user.posi_estimated_0_Z)/(N1*N2);
      North[2]=
sqrt(sqr(user.posi_estimated_0_X)+sqr(user.posi_estimated_0_Y))/N2;

      East[0]= -user.posi_estimated_0_Y/N1;
      East[1]= user.posi_estimated_0_X/N1;
      East[2]= 0;

      Up[0]= user.posi_estimated_0_X/N2;
      Up[1]= user.posi_estimated_0_Y/N2;
      Up[2]= user.posi_estimated_0_Z/N2;

      for(j=0; j<=(NB_SAT-1); j++)
      if(satel[j][k].view)
      {
N3=sqrt(sqr(user_satel_estimated_0[j][0])+sqr(user_satel_estimated_0[
j][1])+
sqr(user_satel_estimated_0[j][2]));

V[0]=user_satel_estimated_0[j][0]/N3;
V[1]=user_satel_estimated_0[j][1]/N3;
V[2]=user_satel_estimated_0[j][2]/N3;

aux1=(Up[0]*V[0]+Up[1]*V[1]+Up[2]*V[2]);

aux2=(North[0]*V[0]+North[1]*V[1]+North[2]*V[2]);

aux3=(East[0]*V[0]+East[1]*V[1]+East[2]*V[2]);


      elevation=asin(aux1);
      azimuth=atan2(aux3, aux2);
      if (azimuth<0) azimuth=2*pi+azimuth;
```

```
      }
      }

void compute_el_az_ellipt(int k)
      {
double          East[3];
double                North[3];
double                Up[3];
double                V[3];
double                p,R,D;
const double    a = 6378137L;
const double    b = 6356752L;
double                e2,lambda,phi;
int j;
double Elevation, Azimuth;

e2 = 1 - b*b/(a*a);

p =
sqrt(user.posi_estimated_0_X*user.posi_estimated_0_X+user.posi_estima
ted_0_Y*user.posi_estimated_0_Y);
R =
sqrt(user.posi_estimated_0_X*user.posi_estimated_0_X+user.posi_estima
ted_0_Y*user.posi_estimated_0_Y+user.posi_estimated_0_Z*user.posi_est
imated_0_Z);

lambda = atan2(user.posi_estimated_0_Y,user.posi_estimated_0_X);
phi = atan2(user.posi_estimated_0_Z , ((1-e2)*p));


Up[0] = cos(phi)*cos(lambda);
Up[1] = cos(phi)*sin(lambda);
Up[2] = sin(phi);

North[0] = -sin(phi)*cos(lambda);
North[1] = -sin(phi)*sin(lambda);
North[2] = cos(phi);

East[0] = North[1]*Up[2] - North[2]*Up[1];
East[1] = North[2]*Up[0] - North[0]*Up[2];
East[2] = North[0]*Up[1] - North[1]*Up[0];


      for(j=0; j<=(NB_SAT-1); j++)
      if(satel[j][k].view)
      {
D =
sqrt(sqr(user_satel_estimated_0[j][0])+sqr(user_satel_estimated_0[j][
1])+
sqr(user_satel_estimated_0[j][2]));

V[0] = (user_satel_estimated_0[j][0]) / D;
V[1] = (user_satel_estimated_0[j][1]) / D;
V[2] = (user_satel_estimated_0[j][2]) / D;
```

```
satel[j][k].El = asin(V[0]*Up[0]+V[1]*Up[1]+V[2]*Up[2]);
satel[j][k].Az =
atan2(East[0]*V[0]+East[1]*V[1]+East[2]*V[2],North[0]*V[0]+North[1]*V
[1]+North[2]*V[2]);

if (satel[j][k].Az<0)
     satel[j][k].Az = 2*pi + satel[j][k].Az;
      }

}



SCoord_pp pierce_point(int i, int k)
      {
      SCoord_pp result;
      double psy_pp;

psy_pp=pi/2-satel[i][k].El-
asin(R_EARTH/(R_EARTH+hI)*cos(satel[i][k].El));

result.lati_pp=asin(sinlati(user.posi_true_X, user.posi_true_Y,
user.posi_true_Z)*cos(psy_pp)+coslati(user.posi_true_X,
user.posi_true_Y, user.posi_true_Z)*sin(psy_pp)*cos(satel[i][k].Az));

result.longi_pp=longi(user.posi_true_X, user.posi_true_Y,
user.posi_true_Z)+asin(sin(psy_pp)*sin(satel[i][k].Az)/cos(result.lat
i_pp));

      return(result);
      }

double compute_f(double x, double y)
      {
      return sqr(x)*sqr(y)*(9-6*x-6*y+4*x*y);
      }

void write_affectation_0(){
      int i;

      delta_x_estimated_0[0]=user.posi_estimated_0_X-
user.posi_true_X;
      delta_x_estimated_0[1]=user.posi_estimated_0_Y-
user.posi_true_Y;
      delta_x_estimated_0[2]=user.posi_estimated_0_Z-
user.posi_true_Z;
      delta_x_estimated_0[3]=user.posi_estimated_0_T-
user.posi_true_T;
      }

void write_affectation_1(int k){
      int i;

      for (i=0; i<=(number_satel_in_view-1); i++)
      {
      error_NB_SAT_1[i][0]=-correction_iono[i][k];
      error_NB_SAT_0[i][0]=-error_satel_0[i][k];
```

```
      }
      }

void write_affectation_2(int k)
      {
      int j,u;

      for(j=0; j<=(NB_SAT-1); j++)
      {
      if (satel[j][k].view)
      {
      user_satel_estimated_0[j][0]=satel[j][k].posi_X-
user.posi_estimated_0_X;
      user_satel_estimated_0[j][1]=satel[j][k].posi_Y-
user.posi_estimated_0_Y;
      user_satel_estimated_0[j][2]=satel[j][k].posi_Z-
user.posi_estimated_0_Z;

      user_satel_true[j][0]=satel[j][k].posi_X-user.posi_true_X;
      user_satel_true[j][1]=satel[j][k].posi_Y-user.posi_true_Y;
      user_satel_true[j][2]=satel[j][k].posi_Z-user.posi_true_Z;
      }
      }
      }

void write_affectation_4(int k)
      {
      int i,j,u;

      i=0;
      for (j=0; j<=(NB_SAT-1); j++)
      if (satel[j][k].view)
      {
      for (u=0; u<=2; u++)
      {
      user_satel_estimated_0[i][u]=user_satel_estimated_0[j][u];
      user_satel_true[i][u]=user_satel_true[j][u];
      }
      satel[i][k].El=satel[j][k].El;
      satel[i][k].Az=satel[j][k].Az;
      i++;
      }
      }

void write_affectation_3(int k)
      {
      int i;

      for(i=0; i<=(number_satel_in_view-1); i++)
      {
      user_satel_estimated_0[i][0]=satel[i][k].posi_X-
user.posi_estimated_0_X;
      user_satel_estimated_0[i][1]=satel[i][k].posi_Y-
user.posi_estimated_0_Y;
      user_satel_estimated_0[i][2]=satel[i][k].posi_Z-
user.posi_estimated_0_Z;
```

```
user_satel_true[i][0]=satel[i][k].posi_X-user.posi_true_X;
user_satel_true[i][1]=satel[i][k].posi_Y-user.posi_true_Y;
user_satel_true[i][2]=satel[i][k].posi_Z-user.posi_true_Z;
}
}
```

```
#ifndef _MODEL_0_
#define _MODEL_0_

#include "fonc_glob.h"

class CIono_0
{
public:
      double ro_L1[NB_SAT][NB_POINT];
      double ro_L2[NB_SAT][NB_POINT];
      double compute_T_Iono(int j, int k);
};

#endif _MODEL_0_
```

```
#include "model_0.h"



double CIono_0::compute_T_Iono(int j, int k)
      {
      double T_io;

      T_io=fabs(sqr(L2)*(ro_L2[j][k]-ro_L1[j][k])/(c*(sqr(L2)-
sqr(L1))));
      return (T_io);
      }
```

```
#ifndef _MODEL_1_
#define _MODEL_1_

#include "fonc_glob.h"

class CIono_1
{
public:
      double T_Iono;
      float alpha[4][NB_POINT];
      float beta[4][NB_POINT];
      double compute_psy(int i, int k);
      double compute_phi(int i, int k);
      double compute_lambda(int i, int k);
      double compute_geo_lati(int i, int k);
      double compute_t(int i, int k);
      double compute_xi(int i, int k);
      double compute_T_Iono(int i, int k);
};

#endif _MODEL_1_
```

```
#include "model_1.h"

double CIono_1::compute_psy(int i, int k)
       {
       return (0.00137/(satel[i][k].El+0.11)-0.022);
       }

double CIono_1::compute_phi(int i, int k)
       {
       double phi;

phi=lati(user.posi_estimated_0_X,user.posi_estimated_0_Y,user.posi_es
timated_0_Z)+compute_psy(i, k)*cos(satel[i][k].Az);

       if (phi>0.416) phi=0.416;
       else
             {
             if (phi<-0.416) phi=-0.416;
             else phi=phi;
             }
       return phi;
       }

double CIono_1::compute_lambda(int i, int k)
       {
       return(longi(user.posi_estimated_0_X,user.posi_estimated_0_Y,us
er.posi_estimated_0_Z)+compute_psy(i,k)*sin(satel[i][k].Az)/cos(compu
te_phi(i,k)));
       }

double CIono_1::compute_geo_lati(int i, int k)
       {
       return(compute_phi(i,k)+0.064*cos(compute_lambda(i,k)-1.617));
       }

double CIono_1::compute_t(int i, int k)
       {
       double t, sec_in_day;
       int nb_day;

       nb_day=(int)time_sec_GPS/86400;
       sec_in_day=time_sec_GPS-nb_day*86400;

       t=4.32e4*compute_lambda(i, k)+ sec_in_day;

       if (t>=86400) t=t-86400;
       else
             {
             if (t<0) t=t+86400;
             else t=t;
             }
       return t;
       }

double CIono_1::compute_xi(int i,int k)
       {
       int j;
       double a=0;

       for (j=0; j<=3; j++)
       a=a+beta[j][k]*sqr_n(compute_geo_lati(i,k),j);
       if (a<72000) a=72000;
       return(2*pi*(compute_t(i, k)-50400)/a);
       }

double CIono_1::compute_T_Iono(int i,int k)
       {
       int j;
       double a, F;

F=1.0+16.0*sqr_n((0.53-satel[i][k].El),3);

       a=0;
       for (j=0; j<=3; j++)
       a=a+alpha[j][k]*sqr_n(compute_geo_lati(i,k),j);

       if (fabs(compute_xi(i,k))>1.57) T_Iono=F*(5e-9);
       else T_Iono=F*(5e-9+a*(1-
(sqr(compute_xi(i,k))/2)+(sqr_n(compute_xi(i,k),4)/24)));

       return T_Iono;

       }
```

```
#ifndef _WHITE_NOISE_
#define _WHITE_NOISE_

#include "fonc_glob.h"

class CGauss
{
public:
      int alea;
      void fcgauss();
      double gauss;
      double valer;
      void fcvaler();
};

struct CUere{
      double clock;
      double ephemeris;
      double iono;
      double tropo;
      double mupath;
      double noise;
      };


class CUereAssess
{
public:
      double t;
      double elev;
      double attenuation;
      double iono;
      double tropo;
      double ephemeris;
      double clock;
      double mupath;
      double noise;
      void computation(int i, int k);
};

#endif _WHITE_NOISE_
```

```
#include "white_noise.h"

void CGauss::fcgauss()
{
      double y;
      int j;

      gauss = 0.0;
      for (j = 1; j<=12; j++)
      {
            fcvaler();
            y = valer;
            gauss = y - gauss;
      }
}

void CGauss::fcvaler()
{
      static int ini,nex,mul,k,kon,ide;
      static double un,div;
      ini = 0;
      nex=10;
      un=1.0;

      //initialisation
      if (ini==0)
      {
            mul = 1;
            for (k=1;k<nex;k++)
                  mul = mul*2;

            kon = mul*mul;
            div = un / (double)(kon);
            ide = mul-3;
            mul = mul+3;
            ini = 1;
      }

      k = alea/ide;
      alea = alea-k*ide;
      alea = mul*alea-9*k+1;
      if (alea < 0.0)
            alea = alea+kon;
      valer = div*(double)(alea);
}


void CUereAssess::computation(int i, int k)
{
      static int init=0;
      static CUere sigma_uere;
      static CGauss bruit;

      if (init==0)
      {
      sigma_uere.clock=sigma_clock;
      sigma_uere.ephemeris=sigma_ephemeris;
```

```
      sigma_uere.iono=sigma_iono;
      sigma_uere.tropo=sigma_tropo;
      sigma_uere.mupath=sigma_mupath;
      sigma_uere.noise=sigma_noise;


      bruit.alea = 2147483647;
      bruit.fcgauss();
      init = 1;
      }
      //determination of error on each pseudorange
      bruit.fcgauss();
      clock = sigma_uere.clock*bruit.gauss;
      bruit.fcgauss();
      ephemeris = sigma_uere.ephemeris*bruit.gauss;
      bruit.fcgauss();
      iono = sigma_uere.iono*bruit.gauss/sin(satel[i][k].El);
      bruit.fcgauss();
      tropo= sigma_uere.tropo*bruit.gauss/sin(satel[i][k].El);
      bruit.fcgauss();
      mupath= sigma_uere.mupath*bruit.gauss /
(sin(satel[i][k].El)/cos(satel[i][k].El));
      bruit.fcgauss();
      noise= sigma_uere.noise*bruit.gauss;
}
```

```
#ifndef _READ_
#define _READ_

#include "model_0.h"
#include "model_1.h"

CIono_0 error_0;
CIono_1 error_1;

class CAlma_Iona
{
public:
      float week;
      float sec;
      int prn[NB_SAT];
      double alpha[4];
      double beta[4];
      void lecture(void);
      int cate;
};

class CRgea_Svda
{
public:
      float week;
      float sec;
      int nb_obs;
      int prn[NB_SAT];
      double ro_L1[NB_SAT];
      double ro_L2[NB_SAT];
      double satel_X[NB_SAT];
      double satel_Y[NB_SAT];
      double satel_Z[NB_SAT];
      double clock_correc[NB_SAT];
      double iono_correc[NB_SAT];
      double tropo_correc[NB_SAT];
      void lecture(void);
      int cate;
};

void read_data(void);

void read_stk(void);

void interpolation(int i, int k);

#endif _READ_
```

```
#include "read.h"


void CAlma_Iona::lecture(void){
      char string_1[8192];
      char string_2[8192];
      char *token;
      char debut[6]="abcde";
      char aux;
      int k,j,temp;
      double vase;

      aux=debut[1];

      fgets(string_1, 8191, File_Iono_1);

      token = strtok(string_1, ",");
      sscanf(token,"%5s",&debut);

      switch(debut[1])
      {
      case 'A':
            {
            cate=0;
            token = strtok(NULL, ",");
            sscanf(token,"%i",&temp);
            prn[temp]=temp;

            token = strtok(NULL, ",");
            token = strtok(NULL, ",");
            sscanf(token,"%f",&sec);

            token = strtok(NULL, ",");
            sscanf(token,"%f",&week);
            }
      break;

      case 'I':
            {
            cate=1;
            token = strtok(NULL, ",");
            sscanf(token,"%lf",&alpha[0]);

            token = strtok(NULL, ",");
            sscanf(token,"%lf",&alpha[1]);

            token = strtok(NULL, ",");
            sscanf(token,"%lf",&alpha[2]);

            token = strtok(NULL, ",");
            sscanf(token,"%lf",&alpha[3]);

            token = strtok(NULL, ",");
            sscanf(token,"%lf",&beta[0]);

            token = strtok(NULL, ",");
            sscanf(token,"%lf",&beta[1]);
```

```
            token = strtok(NULL, ",");
            sscanf(token,"%lf",&beta[2]);

            token = strtok(NULL, ",");
            sscanf(token,"%lf",&beta[3]);
            }
      break;

      case 'U':
            {
            cate=2;
            }
      }
      }


void CRgea_Svda::lecture(void)
      {
      char string_1[8192];
      char *token;
      char debut[6]="abcde";
      char aux;
      int k,j,temp;
      double vase;

      aux=debut[1];

      fgets(string_1, 8191, File_Iono_0);
      token = strtok(string_1, ",");
      sscanf(token,"%5s",&debut);

      cate=2;
      switch (debut[2])
      {
      case 'G' :
            {
            cate=0;

            token = strtok(NULL, ",");
            sscanf(token,"%f",&week);

            token = strtok(NULL, ",");
            sscanf(token,"%f",&sec);

            token = strtok(NULL, ",");
            sscanf(token,"%i",&nb_obs);
            token = strtok(NULL, ",");

            for (k=0; k<=(nb_obs/2-1); k++)
            {
            token = strtok(NULL, ",");
            sscanf(token,"%i",&temp);
            prn[temp-1]=temp-1;

            token = strtok(NULL, ",");
            sscanf(token,"%lf",&ro_L1[temp-1]);
```

```
        for (j=0; j<=8; j++)
        token = strtok(NULL, ",");
        sscanf(token,"%lf",&ro_L2[temp-1]);

        for (j=0; j<=6; j++)
        token = strtok(NULL, ",");
        }
        }
        break;

case 'V' :
        {
        cate=1;

        token = strtok(NULL, ",");
        sscanf(token,"%f",&week);

        token = strtok(NULL, ",");
        sscanf(token,"%f",&sec);

        token = strtok(NULL, ",");
        token = strtok(NULL, ",");
        sscanf(token,"%i",&nb_obs);

        for (k=0; k<=(nb_obs-1); k++)
        {
        token = strtok(NULL, ",");
        sscanf(token,"%i",&temp);
        prn[temp-1]=temp-1;

        token = strtok(NULL, ",");
        sscanf(token,"%lf",&satel_X[temp-1]);

        token = strtok(NULL, ",");
        sscanf(token,"%lf",&satel_Y[temp-1]);

        token = strtok(NULL, ",");
        sscanf(token,"%lf",&satel_Z[temp-1]);

        token = strtok(NULL, ",");
        sscanf(token,"%lf",&clock_correc[temp-1]);

        token = strtok(NULL, ",");
        sscanf(token,"%lf",&iono_correc[temp-1]);

        token = strtok(NULL, ",");
        sscanf(token,"%lf",&tropo_correc[temp-1]);

        token = strtok(NULL, ",");
        token = strtok(NULL, ",");
        }
        }
        break;
 }
 }
```

```
void read_data(void)
        {
        int prn,j,k,l,u, tps_init;
        CRgea_Svda rgea_svda;
        CAlma_Iona alma_iona;

        File_Iono_0=fopen("log_n.gps","r");
        rgea_svda.cate=2;
        while (rgea_svda.cate!=0) rgea_svda.lecture();
        tps_init=rgea_svda.sec;
        fclose(File_Iono_0);

        rgea_svda.sec=0;
        k=0;

        File_Iono_0=fopen("log_n.gps","r");
        File_Iono_1=fopen("log_mik.ion","r");

        alma_iona.cate=0;
        while (alma_iona.cate != 1)
        alma_iona.lecture();

        for(j=0; j<=(NB_SAT-1); j++)
        view_during_simu[j]=0;

        while (rgea_svda.sec<(tps_init+duration))
        {
        l=(int)k/2;
        rgea_svda.lecture();
        time_GPS=rgea_svda.week*604800+rgea_svda.sec;
        time_sec_GPS=rgea_svda.sec;

                if (rgea_svda.cate==0)
                {
                number_satel_in_view=0;
                for(j=0; j<=(NB_SAT-1); j++)
                {
                if (rgea_svda.prn[j] != 0)
                        {
                        view_during_simu[j]=1;
                        satel[j][l].view=1;
                        prn=rgea_svda.prn[j];
                        error_0.ro_L1[prn][l]=rgea_svda.ro_L1[prn];
                        error_0.ro_L2[prn][l]=rgea_svda.ro_L2[prn];
                        number_satel_in_view++;
                        }
                }
                k++;
                }

                if (rgea_svda.cate==1)
                {
                for(j=0; j<=(NB_SAT-1); j++)
                {
                if (rgea_svda.prn[j] != 0)
                        {
                        prn=rgea_svda.prn[j];
```

```
                    satel[prn][1].posi_X=rgea_svda.satel_X[prn];
                    satel[prn][1].posi_Y=rgea_svda.satel_Y[prn];
                    satel[prn][1].posi_Z=rgea_svda.satel_Z[prn];
                    correction_iono[prn][1]=rgea_svda.iono_correc[prn];

                    for(u=0; u<=3; u++)
                    {
                    error_1.alpha[u][1]=alma_iona.alpha[u];
                    error_1.beta[u][1]=alma_iona.beta[u];
                    }
                    }
               }
               k++;
               }
          }
          fclose(File_Iono_0);
          fclose(File_Iono_1);
          }

void read_stk(void)
          {
          FILE *infile[NB_FILE_STK];     //array of satellites' data files
          int i,k,j,aux;
          float tps,x,y,z,vx,vy,vz;
          double vase[3];
          double vase1;

          infile[0]=fopen("gps_2-01.sa","r");
          infile[1]=fopen("gps_2-02.sa","r");
          infile[2]=fopen("gps_2-03.sa","r");
          infile[3]=fopen("gps_2-04.sa","r");
          infile[4]=fopen("gps_2-05.sa","r");
          infile[5]=fopen("gps_2-06.sa","r");
          infile[6]=fopen("gps_2-07.sa","r");
          infile[7]=fopen("gps_2-08.sa","r");
          infile[8]=fopen("gps_2-09.sa","r");
          infile[9]=fopen("gps_2-10.sa","r");
          infile[10]=fopen("gps_2-11.sa","r");
          infile[11]=fopen("gps_2-12.sa","r");
          infile[12]=fopen("gps_2-13.sa","r");
          infile[13]=fopen("gps_2-14.sa","r");
          infile[14]=fopen("gps_2-15.sa","r");
          infile[15]=fopen("gps_2-16.sa","r");
          infile[16]=fopen("gps_2-17.sa","r");
          infile[17]=fopen("gps_2-18.sa","r");
          infile[18]=fopen("gps_2-19.sa","r");
          infile[19]=fopen("gps_2-20.sa","r");
          infile[20]=fopen("gps_2-21.sa","r");
          infile[21]=fopen("gps_2-22.sa","r");
          infile[22]=fopen("gps_2-23.sa","r");
          infile[23]=fopen("gps_2-24.sa","r");
          infile[24]=fopen("gps_2-25.sa","r");
          infile[25]=fopen("gps_2-26.sa","r");
          infile[26]=fopen("gps_2-27.sa","r");

          aux=(int)(duration/60+1);
```

```
          for (i=0; i<=aux; i++)
          {
          for (k=0; k<=(NB_FILE_STK-1); k++)
          {
          fscanf(infile[k],"%f\t%f\t%f\t%f\t%f\t%f\t%f\n",
                 &tps,
                 &x,
                 &y,
                 &z,
                 &vx,
                 &vy,
                 &vz);

          satel[k][i*60].posi_X=(double)x;
          cout<<"satel["<<k<<"]["<<i*60<<"].posi_X="<<satel[k][i*60].posi
_X<<"\n";
          satel[k][i*60].posi_Y=(double)y;
          satel[k][i*60].posi_Z=(double)z;
          satel[k][i*60].time=i*60;
          }
          }

          for (k=0; k<=(NB_FILE_STK-1); k++)
          fclose(infile[k]);

          for (i=0; i<=(aux-1); i++)
          for (k=0; k<=(NB_FILE_STK-1); k++)
          interpolation(i,k);

          for(j=0; j<=(duration-1); j++)
          {
          i=0;
          for(k=0; k<=(NB_FILE_STK-1); k++)
                 {
                 cout<<"i="<<i<<"\n";
                 vase[0]=satel[k][j].posi_X-user.posi_true_X;
                 vase[1]=satel[k][j].posi_Y-user.posi_true_Y;
                 vase[2]=satel[k][j].posi_Z-user.posi_true_Z;
vase1=(vase[0]*user.posi_true_X+vase[1]*user.posi_true_Y+vase[2]*user
.posi_true_Z)/(sqrt(sqr(vase[0])+sqr(vase[1])+sqr(vase[2]))*sqrt(sqr(
user.posi_true_X)+sqr(user.posi_true_Y)+sqr(user.posi_true_Z)));

                 if (vase1>Sin_Mask_Angle)
                 {
                 satel[k][j].view=1;
                 satel[i][j].posi_X=satel[k][j].posi_X;
                 satel[i][j].posi_Y=satel[k][j].posi_Y;
                 satel[i][j].posi_Z=satel[k][j].posi_Z;
                 satel[i][j].time=satel[k][j].time;
                 i++;
                 }
                 }
          }
          number_satel_in_view=i;
          }

void interpolation(int i, int k)
```

```
        {
        int j;

        for (j=i; j<=i+59; j++)
        {
        satel[k][j].posi_X=satel[k][i*60].posi_X +
(satel[k][60*(i+1)].posi_X-satel[k][i*60].posi_X)/60;
        satel[k][j].posi_Y=satel[k][i*60].posi_Y +
(satel[k][60*(i+1)].posi_Y-satel[k][i*60].posi_Y)/60;
        satel[k][j].posi_Z=satel[k][i*60].posi_Z +
(satel[k][60*(i+1)].posi_Z-satel[k][i*60].posi_Z)/60;
        satel[k][j].time=60*i+j;
        }
        }
```

```
#ifndef _EGNOS_
#define _EGNOS_

#include "fonc_glob.h"
#include "read.h"
#include "in_data.h"

class CEgnos
{
public:
      double lati;
      double longi;
      int grid_point;
      double iono_delay;
      double iono_accuracy;
      int active_grid_point;
      int active_delay;
      int active_accuracy;
};

CEgnos IGP[NB_POINT][NB_BAND][NB_SAT_IN_BAND];


void compute_error_egnos(int k);

void remplissage_egnos(void);

#endif _EGNOS_
```

```
#include "egnos.h"


void remplissage_egnos(void)
      {
      int i,j;

      for(j=0; j<=(duration-1); j++)
      {
      for(i=18; i<=26; i++)
      IGP[j][4][i].grid_point=1;
      for(i=43; i<=49; i++)
      IGP[j][4][i].grid_point=1;
      for(i=68; i<=76; i++)
      IGP[j][4][i].grid_point=1;
      for(i=93; i<=99; i++)
      IGP[j][4][i].grid_point=1;
      for(i=118; i<=126; i++)
      IGP[j][4][i].grid_point=1;
      for(i=144; i<=150; i++)
      IGP[j][4][i].grid_point=1;
      for(i=169; i<=177; i++)
      IGP[j][4][i].grid_point=1;
      for(i=194; i<=200; i++)
      IGP[j][4][i].grid_point=1;

      for(i=18; i<=26; i++)
      IGP[j][5][i].grid_point=1;
      for(i=43; i<=49; i++)
      IGP[j][5][i].grid_point=1;
      for(i=68; i<=76; i++)
      IGP[j][5][i].grid_point=1;
      for(i=93; i<=99; i++)
      IGP[j][5][i].grid_point=1;
      for(i=119; i<=128; i++)
      IGP[j][5][i].grid_point=1;
      for(i=144; i<=150; i++)
      IGP[j][5][i].grid_point=1;

      IGP[j][4][18].iono_delay=2.38;
      IGP[j][4][19].iono_delay=1.62;
      IGP[j][4][20].iono_delay=1.50;
      IGP[j][4][21].iono_delay=3.12;
      IGP[j][4][22].iono_delay=5.75;
      IGP[j][4][23].iono_delay=8.00;
      IGP[j][4][24].iono_delay=8.38;
      IGP[j][4][25].iono_delay=2.12;
      IGP[j][4][26].iono_delay=63.75;
      IGP[j][4][43].iono_delay=2.38;
      IGP[j][4][44].iono_delay=1.50;
      IGP[j][4][45].iono_delay=1.38;
      IGP[j][4][46].iono_delay=3.50;
      IGP[j][4][47].iono_delay=5.75;
      IGP[j][4][48].iono_delay=7.88;

      IGP[j][4][49].iono_delay=9.62;
```

```
      IGP[j][4][68].iono_delay=2.00;
      IGP[j][4][69].iono_delay=1.62;
      IGP[j][4][70].iono_delay=1.38;
      IGP[j][4][71].iono_delay=3.50;
      IGP[j][4][72].iono_delay=5.88;
      IGP[j][4][73].iono_delay=7.88;
      IGP[j][4][74].iono_delay=8.62;
      IGP[j][4][75].iono_delay=2.25;
      IGP[j][4][76].iono_delay=63.75;
      IGP[j][4][93].iono_delay=3.12;
      IGP[j][4][94].iono_delay=2.12;
      IGP[j][4][95].iono_delay=1.12;
      IGP[j][4][96].iono_delay=4.62;
      IGP[j][4][97].iono_delay=6.50;

      IGP[j][4][98].iono_delay=8.00;
      IGP[j][4][99].iono_delay=7.75;
      IGP[j][4][118].iono_delay=3.88;
      IGP[j][4][119].iono_delay=3.00;
      IGP[j][4][120].iono_delay=1.75;
      IGP[j][4][121].iono_delay=4.38;
      IGP[j][4][122].iono_delay=6.75;
      IGP[j][4][123].iono_delay=7.88;
      IGP[j][4][124].iono_delay=7.75;
      IGP[j][4][125].iono_delay=2.25;
      IGP[j][4][126].iono_delay=63.75;
      IGP[j][4][144].iono_delay=2.88;
      IGP[j][4][145].iono_delay=3.12;
      IGP[j][4][146].iono_delay=2.12;
      IGP[j][4][147].iono_delay=4.12;

      IGP[j][4][148].iono_delay=6.50;
      IGP[j][4][149].iono_delay=7.75;
      IGP[j][4][150].iono_delay=5.12;
      IGP[j][4][169].iono_delay=2.12;
      IGP[j][4][170].iono_delay=3.25;
      IGP[j][4][171].iono_delay=2.50;
      IGP[j][4][172].iono_delay=3.88;
      IGP[j][4][173].iono_delay=6.25;
      IGP[j][4][174].iono_delay=7.12;
      IGP[j][4][175].iono_delay=5.50;
      IGP[j][4][176].iono_delay=1.50;
      IGP[j][4][177].iono_delay=63.75;
      IGP[j][4][194].iono_delay=1.38;
      IGP[j][4][195].iono_delay=3.75;
      IGP[j][4][196].iono_delay=2.75;

      IGP[j][4][197].iono_delay=3.75;
      IGP[j][4][198].iono_delay=5.25;
      IGP[j][4][199].iono_delay=6.38;
      IGP[j][4][200].iono_delay=4.88;


      IGP[j][5][18].iono_delay=5.62;
      IGP[j][5][19].iono_delay=4.12;
      IGP[j][5][20].iono_delay=3.00;
      IGP[j][5][21].iono_delay=2.88;
```

```
        IGP[j][5][22].iono_delay=4.12;
        IGP[j][5][23].iono_delay=5.50;
        IGP[j][5][24].iono_delay=4.62;
        IGP[j][5][25].iono_delay=2.38;
        IGP[j][5][26].iono_delay=63.75;
        IGP[j][5][43].iono_delay=6.50;
        IGP[j][5][44].iono_delay=3.50;
        IGP[j][5][45].iono_delay=2.88;
        IGP[j][5][46].iono_delay=2.75;
        IGP[j][5][47].iono_delay=3.75;
        IGP[j][5][48].iono_delay=5.75;

        IGP[j][5][49].iono_delay=4.75;
        IGP[j][5][68].iono_delay=5.00;
        IGP[j][5][69].iono_delay=3.75;
        IGP[j][5][70].iono_delay=3.25;
        IGP[j][5][71].iono_delay=2.38;
        IGP[j][5][72].iono_delay=4.00;
        IGP[j][5][73].iono_delay=5.50;
        IGP[j][5][74].iono_delay=4.50;
        IGP[j][5][75].iono_delay=3.25;
        IGP[j][5][76].iono_delay=63.75;
        IGP[j][5][93].iono_delay=4.75;
        IGP[j][5][94].iono_delay=4.50;
        IGP[j][5][95].iono_delay=3.62;
        IGP[j][5][96].iono_delay=2.50;
        IGP[j][5][97].iono_delay=3.88;

        IGP[j][5][98].iono_delay=63.75;
        IGP[j][5][99].iono_delay=63.75;
        IGP[j][5][119].iono_delay=63.75;
        IGP[j][5][120].iono_delay=5.25;
        IGP[j][5][121].iono_delay=7.50;
        IGP[j][5][122].iono_delay=4.62;
        IGP[j][5][123].iono_delay=3.25;
        IGP[j][5][124].iono_delay=63.75;
        IGP[j][5][125].iono_delay=63.75;
        IGP[j][5][126].iono_delay=63.75;
        IGP[j][5][127].iono_delay=63.75;
        IGP[j][5][144].iono_delay=63.75;
        IGP[j][5][145].iono_delay=63.75;
        IGP[j][5][146].iono_delay=63.75;
        IGP[j][5][147].iono_delay=63.75;

        IGP[j][5][148].iono_delay=63.75;
        IGP[j][5][149].iono_delay=63.75;
        IGP[j][5][125].iono_delay=63.75;
        }
        }

void compute_error_egnos(int k)
        {
        int i,j, num_col_right, num_col_left, num_line_high,
num_line_low;
        double aux, aux1, aux_lati, aux_longi;
        int pt_1, pt_2, pt_3, pt_4;
        int cas, band;
```

```
double vase1, vase2, vase3, vase4;
double lati_low, lati_high, longi_left, longi_right;
double x_pp, y_pp;


for(i=0; i<=(number_satel_in_view-1); i++)
{
num_col_right=0;
num_line_high=0;

aux_longi=pierce_point(i, k).longi_pp*180/pi;
if ((aux_longi<15) && (aux_longi)>-20)
{
band=4;
aux1=-20;
while(aux1<aux_longi)
        {
        aux1=aux1+5;
        num_col_right++;
        }
num_col_left=num_col_right-1;
longi_right=aux1;
longi_left=aux1-5;

aux_lati=pierce_point(i, k).lati_pp*180/pi;
if(abs(aux_lati)<55)
        {
        num_line_high=2;
        aux1=-55;
        while(aux1<aux_lati)
        {
        aux1=aux1+5;
        num_line_high++;
        }
        lati_high=aux1;
        lati_low=aux1-5;
        }
else
        {
        if(abs(aux_lati)<75)
        {
        if (aux_lati>0)
                {
                num_line_high=24;
                aux1=55;
                while(aux1<aux_lati)
                {
                aux1=aux1+10;
                num_line_high++;
                }
                }
        else
                {
                num_line_high=0;
                aux1=-75;
                while(aux1<aux_lati)
                {
                    aux1=aux1+10;
                    num_line_high++;
                    }
                    }
              lati_high=aux1;
              lati_low=aux1-10;
              }
              else
                    cout<<"pierce point too high\n";
              }
              num_line_low=num_line_high-1;

        if(abs(aux_lati)<55)
        {
        if ((num_col_left==0) || (num_col_left==2) ||
(num_col_left==4))
              {
              pt_1=num_col_left/2*50;
              pt_1=pt_1+num_line_low;
              }
        if ((num_col_left==1) || (num_col_left==3))
              {
              aux=(int)(num_col_left/2);
              pt_1=aux*50+27;
              pt_1=pt_1+num_line_low-2;
              }
        if (num_col_left==6)
              {
              pt_1=3*50+1;
              pt_1=pt_1+num_line_low;
              }
        if ((num_col_left==5) || (num_col_left==7))
              {
              aux=(int)(num_col_left/2);
              pt_1=aux*50+27+1;
              pt_1=pt_1+num_line_low-2;
              }
        pt_2=pt_1+1;

        if ((num_col_left==0) || (num_col_left==2) || (num_col_left==6)
|| (num_col_left==1) || (num_col_left==3) || (num_col_left==5) ||
(num_col_left==7))
              {
              pt_3=pt_1+25;
              pt_4=pt_1+26;
              }

        if ((num_col_left==4))
              {
              pt_3=pt_1+26;
              pt_4=pt_1+27;
              }

        if(IGP[k][band][pt_1].grid_point &&
IGP[k][band][pt_2].grid_point && IGP[k][band][pt_3].grid_point &&
IGP[k][band][pt_4].grid_point)
              {
```

```
                    cas=1;
                    vase1=IGP[k][band][pt_1].iono_delay;
                    vase2=IGP[k][band][pt_2].iono_delay;
                    vase3=IGP[k][band][pt_3].iono_delay;
                    vase4=IGP[k][band][pt_4].iono_delay;
                    }

              if (cas == 1)
                    {
                    x_pp=(aux_longi-longi_left)/(longi_right-longi_left);
                    y_pp=(aux_lati-lati_low)/(lati_high-lati_low);

        error_iono_egnos[i][k]=1/sqrt(1-
        sqr(R_e*cos(satel[i][k].El)/(R_e+hI))) * (compute_f(x_pp, y_pp)*vase3
        + compute_f((1-x_pp), y_pp)*vase2 + compute_f((1-x_pp), (1-
        y_pp))*vase1 + compute_f(x_pp, (1-y_pp))*vase4);

        error_iono_0[i][k]=error_iono_egnos[i][k];
                    }
              }
              }

              if ((aux_longi<55) && (aux_longi)>20)
              {
              band=5;
              aux1=20;
              while(aux1<aux_longi)
                    {
                    aux1=aux1+5;
                    num_col_right++;
                    }
              num_col_left=num_col_right-1;
              longi_right=aux1;
              longi_left=aux1-5;

              aux_lati=pierce_point(i, k).lati_pp*180/pi;
              if(abs(aux_lati)<55)
                    {
                    num_line_high=2;
                    aux1=-55;
                    while(aux1<aux_lati)
                    {
                    aux1=aux1+5;
                    num_line_high++;
                    }
                    lati_high=aux1;
                    lati_low=aux1-5;
                    }
              else
                    {
                    if(abs(aux_lati)<75)
                    {
                    if (aux_lati>0)
                          {
                          num_line_high=24;
                          aux1=55;
                          while(aux1<aux_lati)
```

```
                {
                aux1=aux1+10;
                num_line_high++;
                }
                }
          else
                {
                num_line_high=0;
                aux1=-75;
                while(aux1<aux_lati)
                {
                aux1=aux1+10;
                num_line_high++;
                }
                }
          lati_high=aux1;
          lati_low=aux1-10;
          }
          else
                cout<<"pierce point too high\n";
          }
          num_line_low=num_line_high-1;

     if(abs(aux_lati)<55)
     {
     if ((num_col_left==0) || (num_col_left==2) ||
(num_col_left==4))
          {
          pt_1=num_col_left/2*50;
          pt_1=pt_1+num_line_low;
          }
     if ((num_col_left==1) || (num_col_left==3))
          {
          aux=(int)(num_col_left/2);
          pt_1=aux*50+27;
          pt_1=pt_1+num_line_low-2;
          }
     if (num_col_left==6)
          {
          pt_1=3*50+1;
          pt_1=pt_1+num_line_low;
          }
     if ((num_col_left==5) || (num_col_left==7))
          {
          aux=(int)(num_col_left/2);
          pt_1=aux*50+27+1;
          pt_1=pt_1+num_line_low-2;
          }
     pt_2=pt_1+1;

     if ((num_col_left==0) || (num_col_left==2) || (num_col_left==6)
|| (num_col_left==1) || (num_col_left==3) || (num_col_left==5) ||
(num_col_left==7))
          {
          pt_3=pt_1+25;
          pt_4=pt_1+26;
          }
```

```
     if ((num_col_left==4))
          {
          pt_3=pt_1+26;
          pt_4=pt_1+27;
          }


     if(IGP[k][band][pt_1].grid_point &&
IGP[k][band][pt_2].grid_point && IGP[k][band][pt_3].grid_point &&
IGP[k][band][pt_4].grid_point)
          {
          cas=1;
          vase1=IGP[k][band][pt_1].iono_delay;
          vase2=IGP[k][band][pt_2].iono_delay;
          vase3=IGP[k][band][pt_3].iono_delay;
          vase4=IGP[k][band][pt_4].iono_delay;
          }

     if (cas == 1)
          {
          x_pp=(aux_longi-longi_left)/(longi_right-longi_left);
          y_pp=(aux_lati-lati_low)/(lati_high-lati_low);

          cout<<"x_pp="<<x_pp<<"\t";
          cout<<"y_pp="<<y_pp<<"\n";

error_iono_egnos[i][k]=1/sqrt(1-
sqr(R_e*cos(satel[i][k].El)/(R_e+hI))) * (compute_f(x_pp, y_pp)*vase3
+ compute_f((1-x_pp), y_pp)*vase2 + compute_f((1-x_pp), (1-
y_pp))*vase1 + compute_f(x_pp, (1-y_pp))*vase4);

error_iono_0[i][k]=error_iono_egnos[i][k];


          }
     }
     }


     if ((aux_longi<20) && (aux_longi)>15)
     {
     num_col_right=0;
     num_col_left=7;
     longi_right=20;
     longi_left=15;

     aux_lati=pierce_point(i, k).lati_pp*180/pi;
     if(abs(aux_lati)<55)
          {
          num_line_high=2;
          aux1=-55;
          while(aux1<aux_lati)
          {
          aux1=aux1+5;
          num_line_high++;
          }
```

```
        lati_high=aux1;
        lati_low=aux1-5;
        }
    else
        {
        if(abs(aux_lati)<75)
        {
        if (aux_lati>0)
            {
            num_line_high=24;
            aux1=55;
            while(aux1<aux_lati)
            {
            aux1=aux1+10;
            num_line_high++;
            }
            }
        else
            {
            num_line_high=0;
            aux1=-75;
            while(aux1<aux_lati)
            {
            aux1=aux1+10;
            num_line_high++;
            }
            }
        lati_high=aux1;
        lati_low=aux1-10;
        }
        else
            cout<<"pierce point too high\n";
        }
        num_line_low=num_line_high-1;


    pt_1=178+num_line_low-2;
    pt_2=pt_1+1;

    pt_3=num_line_low;
    pt_4=num_line_low+1;


    if(IGP[k][4][pt_1].grid_point && IGP[k][4][pt_2].grid_point &&
IGP[k][5][pt_3].grid_point && IGP[k][5][pt_4].grid_point)
        {
        cas=1;
        vase1=IGP[k][4][pt_1].iono_delay;
        cout<<"vase1="<<vase1<<"\n";
        vase2=IGP[k][4][pt_2].iono_delay;
        vase3=IGP[k][5][pt_3].iono_delay;
        vase4=IGP[k][5][pt_4].iono_delay;
        }

    if (cas == 1)
        {
```

```
                x_pp=(aux_longi-longi_left)/(longi_right-longi_left);
                y_pp=(aux_lati-lati_low)/(lati_high-lati_low);


error_iono_egnos[i][k]=1/sqrt(1-
sqr(R_e*cos(satel[i][k].El)/(R_e+hI))) * (compute_f(1,
y_pp)*IGP[k][5][22].iono_delay + compute_f(1, 1-x_pp-
y_pp)*IGP[k][4][200].iono_delay + compute_f(x_pp,
1)*IGP[k][5][21].iono_delay);

error_iono_0[i][k]=error_iono_egnos[i][k];
                }

            }
            }
            }
```

```
#ifndef _COMPUTE_K_
#define _COMPUTE_K_

#include "variable_glob.h"
#include "fonc_glob.h"

void compute_G_0(int k);
void compute_W_0(int k);
void compute_K_0(void);

void compute_G_1(int k);
void compute_W_1(int k);
void compute_K_1(void);

double uere(void);
void compute_A(void);
void compute_DOP(int k);
void compute_expected_error(int k);

#endif _COMPUTE_K_
```

```
#include "compute_K.h"

void compute_G_0(int k){
      int i,j;
      FILE *out_file;
      double norme_user_satel_estimated_0[NB_SAT];

      for (j=0; j<=(NB_SAT-1); j++)
      for (i=0; i<=(NB_SAT-1); i++)
      G_0[j][i]=0;

      for (i=0; i<=(number_satel_in_view-1); i++)
      {
      norme_user_satel_estimated_0[i]=sqrt(sqr(user_satel_estimated_0[i][0])+sqr
(user_satel_estimated_0[i][1])+sqr(user_satel_estimated_0[i][2]));

      G_0[i][0]=-user_satel_estimated_0[i][0]/norme_user_satel_estimated_0[i];
      G_0[i][1]=-user_satel_estimated_0[i][1]/norme_user_satel_estimated_0[i];
      G_0[i][2]=-user_satel_estimated_0[i][2]/norme_user_satel_estimated_0[i];
      G_0[i][3]=1;
      }
      }

void compute_G_1(int k){
      int i,j;
      double norme_user_satel_estimated_1[NB_SAT];

      for (i=0; i<=(NB_SAT-1); i++)
      for (j=0; j<=(NB_SAT-1); j++)
      G_1[j][i]=0;


      for (i=0; i<=(number_satel_in_view-1); i++)
      {
norme_user_satel_estimated_1[i]=sqrt(sqr(user_satel_estimated_1[i][0])+sqr(user_
satel_estimated_1[i][1])+sqr(user_satel_estimated_1[i][2]));

      G_1[i][0]=-user_satel_estimated_1[i][0]/norme_user_satel_estimated_1[i];
      G_1[i][1]=-user_satel_estimated_1[i][1]/norme_user_satel_estimated_1[i];
      G_1[i][2]=-user_satel_estimated_1[i][2]/norme_user_satel_estimated_1[i];
      G_1[i][3]=1;
      }
      }


void compute_W_0(int k)
      {
      int j;
      for (j=0; j<=(NB_SAT-1); j++)
      W_0[j][j]=1;


      for (j=0; j<=(number_satel_in_view-1); j++)
      {
W_0[j][j]=1/(sqr(sigma_ephemeris)+sqr(sigma_clock)+sqr(sigma_noise)+sqr(sigma_io
no/sin(satel[j][k].El))+sqr(sigma_tropo/sin(satel[j][k].El))+sqr(sigma_mupath/(s
in(satel[j][k].El)/cos(satel[j][k].El))));
      }

      }

void compute_W_1(int k)
      {
      int j;
      for (j=0; j<=(NB_SAT-1); j++)
      W_1[j][j]=1;


      for (j=0; j<=(number_satel_in_view-1); j++)
      {
W_1[j][j]=1/(sqr(sigma_ephemeris)+sqr(sigma_clock)+sqr(sigma_noise)+sqr(sigma_io
no/sin(satel[j][k].El))+sqr(sigma_tropo/sin(satel[j][k].El))+sqr(sigma_mupath/(s
in(satel[j][k].El)/cos(satel[j][k].El))));
      }

      }


void compute_K_0()
      {
      int i,j;
      Square_Mat_NB_SAT aux;

      transposition(G_0, number_satel_in_view, 4);

      multiplication(trans,W_0,4, number_satel_in_view, number_satel_in_view);
      for (i=0; i<=(NB_SAT-1); i++)
      for (j=0; j<=(NB_SAT-1); j++)
      aux[i][j]=product[i][j];

      multiplication(aux,G_0,4, number_satel_in_view, number_satel_in_view);
      for (i=0; i<=(NB_SAT-1); i++)
      for (j=0; j<=(NB_SAT-1); j++)
      aux[i][j]=product[i][j];

      inversion(aux);

      multiplication(aux,trans, 4, number_satel_in_view, 4);
      for (i=0; i<=(NB_SAT-1); i++)
      for (j=0; j<=(NB_SAT-1); j++)
      aux[i][j]=product[i][j];

      multiplication(aux,W_0, 4, number_satel_in_view, number_satel_in_view);
      for (i=0; i<=(NB_SAT-1); i++)
      for (j=0; j<=(NB_SAT-1); j++)
      K_0[i][j]=product[i][j];
      }

void compute_K_1()
      {
      int i,j;
      Square_Mat_NB_SAT aux;

      transposition(G_1, number_satel_in_view, 4);
```

```
          multiplication(trans,W_1,4, number_satel_in_view, number_satel_in_view);
          for (i=0; i<=(NB_SAT-1); i++)
          for (j=0; j<=(NB_SAT-1); j++)
          aux[i][j]=product[i][j];

          multiplication(aux,G_1,4, number_satel_in_view, number_satel_in_view);
          for (i=0; i<=(NB_SAT-1); i++)
          for (j=0; j<=(NB_SAT-1); j++)
          aux[i][j]=product[i][j];

          inversion(aux);

          multiplication(aux,trans, 4, number_satel_in_view, 4);
          for (i=0; i<=(NB_SAT-1); i++)
          for (j=0; j<=(NB_SAT-1); j++)
          aux[i][j]=product[i][j];

          multiplication(aux,W_1, 4, number_satel_in_view, number_satel_in_view);
          for (i=0; i<=(NB_SAT-1); i++)
          for (j=0; j<=(NB_SAT-1); j++)
          K_1[i][j]=product[i][j];
          }

double uere(void)
          {
return (sigma_ephemeris + sigma_clock + sigma_noise + sigma_iono / sin(El_Ref) +
sigma_tropo / sin(El_Ref) + sigma_mupath / tan(El_Ref));
          }

void compute_A(void)
          {
          int i,j;
          Square_Mat_NB_SAT aux;

          transposition(G_0, number_satel_in_view, 4);

          multiplication(trans,G_0, number_satel_in_view, number_satel_in_view,4);
          for (i=0; i<=(NB_SAT-1); i++)
          for (j=0; j<=(NB_SAT-1); j++)
          aux[i][j]=product[i][j];
          inversion(aux);
          for (i=0; i<=3; i++)
          for (j=0; j<=3; j++)
          A[i][j]=aux[i][j];
          }


void compute_DOP(int k)
          {
          int i, j;

          GDOP[k]=sqrt(A[0][0]+A[1][1]+A[2][2]+A[3][3]);
          PDOP[k]=sqrt(A[0][0]+A[1][1]+A[2][2]);
          HDOP[k]=sqrt(A[0][0]+A[1][1]);
          VDOP[k]=sqrt(A[2][2]);
          XDOP[k]=sqrt(A[0][0]);
          YDOP[k]=sqrt(A[1][1]);
          TDOP[k]=sqrt(A[3][3]);
          }


void compute_expected_error(int k)
          {
          expected_error_x[k]=XDOP[k]*uere();
          expected_error_y[k]=YDOP[k]*uere();
          expected_error_z[k]=VDOP[k]*uere();
          expected_error_t[k]=TDOP[k]*uere();
          expected_error_global[k]=GDOP[k]*uere();
          }
```

```c
#ifndef _COMPUTE_K_
#define _COMPUTE_K_

#include "variable_glob.h"
#include "fonc_glob.h"

void compute_G_0(int k);
void compute_W_0(int k);
void compute_K_0(void);

void compute_G_1(int k);
void compute_W_1(int k);
void compute_K_1(void);

double uere(void);
void compute_A(void);
void compute_DOP(int k);
void compute_expected_error(int k);

#endif _COMPUTE_K_
```

```
#ifndef _OUT_DATA_
#define _OUT_DATA_

#include "fonc_glob.h"

void global_storage(void);
void write_error_satellite(void);
void storage(int k);
void compute_error(int k);
void numeric_output(void);

#endif _OUT_DATA_
```

```
#include "out_data.h"


void global_storage(void){
      int i,j,l,aux;
      float vase;
      FILE *out_file;
      char *name[NB_SAT];
      char *name1[NB_SAT];

      aux=(int)(NB_ITE*duration/50);

      vase=user.posi_true_X;
      out_file=fopen("posi_x.m","w");

      for (l=0; l<=(aux-1); l++)
            {
            fprintf(out_file,"x%i=[",l);
            for (i=l*50; i<=(l*50+49); i++)
            fprintf(out_file,"%i\t",i);
            fprintf(out_file,"]\n");
            }
      fprintf(out_file,"x%i=[",aux);
      for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
      fprintf(out_file,"%i\t",i);
      fprintf(out_file,"]\n");

      fprintf(out_file,"x=[");
      for(l=0; l<=aux; l++)
      fprintf(out_file,"x%i\t",l);
      fprintf(out_file,"]\n");

      for (l=0; l<=(aux-1); l++)
            {
            fprintf(out_file,"y%i=[",l);
            for (i=l*50; i<=(l*50+49); i++)
            fprintf(out_file,"%lf\t",posi_estimated_0_X[i]);
            fprintf(out_file,"]\n");
            }
      fprintf(out_file,"y%i=[",aux);
      for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
      fprintf(out_file,"%lf\t",posi_estimated_0_X[i]);
      fprintf(out_file,"]\n");

      fprintf(out_file,"y=[");
      for(l=0; l<=aux; l++)
      fprintf(out_file,"y%i\t",l);
      fprintf(out_file,"]\n");

      for (l=0; l<=(aux-1); l++)
            {
            fprintf(out_file,"z%i=[",l);
            for (i=l*50; i<=(l*50+49); i++)
            fprintf(out_file,"%lf\t",vase);
            fprintf(out_file,"]\n");
            }
      fprintf(out_file,"z%i=[",aux);
```

```
      for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
      fprintf(out_file,"%lf\t",vase);
      fprintf(out_file,"]\n");

      fprintf(out_file,"z=[");
      for(l=0; l<=aux; l++)
      fprintf(out_file,"z%i\t",l);
      fprintf(out_file,"]\n");

      fprintf(out_file,"plot(x,y,'k-',x,z,'k--')\n");
      fprintf(out_file,"title('position on X')\n");
      fprintf(out_file,"xlabel('time (seconds)')\n");
      fprintf(out_file,"ylabel('Position on x (meter)')\n");
      fclose(out_file);



      vase=user.posi_true_Y;
      out_file=fopen("posi_y.m","w");

      for (l=0; l<=(aux-1); l++)
            {
            fprintf(out_file,"x%i=[",l);
            for (i=l*50; i<=(l*50+49); i++)
            fprintf(out_file,"%i\t",i);
            fprintf(out_file,"]\n");
            }
      fprintf(out_file,"x%i=[",aux);
      for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
      fprintf(out_file,"%i\t",i);
      fprintf(out_file,"]\n");

      fprintf(out_file,"x=[");
      for(l=0; l<=aux; l++)
      fprintf(out_file,"x%i\t",l);
      fprintf(out_file,"]\n");

      for (l=0; l<=(aux-1); l++)
            {
            fprintf(out_file,"y%i=[",l);
            for (i=l*50; i<=(l*50+49); i++)
            fprintf(out_file,"%lf\t",posi_estimated_0_Y[i]);
            fprintf(out_file,"]\n");
            }
      fprintf(out_file,"y%i=[",aux);
      for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
      fprintf(out_file,"%lf\t",posi_estimated_0_Y[i]);
      fprintf(out_file,"]\n");

      fprintf(out_file,"y=[");
      for(l=0; l<=aux; l++)
      fprintf(out_file,"y%i\t",l);
      fprintf(out_file,"]\n");

      for (l=0; l<=(aux-1); l++)
            {
            fprintf(out_file,"z%i=[",l);
```

```
          for (i=l*50; i<=(l*50+49); i++)
          fprintf(out_file,"%lf\t",vase);
          fprintf(out_file,"]\n");
          }
     fprintf(out_file,"z%i=[",aux);
     for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
     fprintf(out_file,"%lf\t",vase);
     fprintf(out_file,"]\n");

     fprintf(out_file,"z=[");
     for(l=0; l<=aux; l++)
     fprintf(out_file,"z%i\t",l);
     fprintf(out_file,"]\n");

     fprintf(out_file,"plot(x,y,'k-',x,z,'k--')\n");
     fprintf(out_file,"title('position on Y')\n");
     fprintf(out_file,"xlabel('time (seconds)')\n");
     fprintf(out_file,"ylabel('Position on y (meter)')\n");

     fclose(out_file);



     vase=user.posi_true_Z;
     out_file=fopen("posi_z.m","w");
     for (l=0; l<=(aux-1); l++)
          {
          fprintf(out_file,"x%i=[",l);
          for (i=l*50; i<=(l*50+49); i++)
          fprintf(out_file,"%i\t",i);
          fprintf(out_file,"]\n");
          }
     fprintf(out_file,"x%i=[",aux);
     for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
     fprintf(out_file,"%i\t",i);
     fprintf(out_file,"]\n");

     fprintf(out_file,"x=[");
     for(l=0; l<=aux; l++)
     fprintf(out_file,"x%i\t",l);
     fprintf(out_file,"]\n");

     for (l=0; l<=(aux-1); l++)
          {
          fprintf(out_file,"y%i=[",l);
          for (i=l*50; i<=(l*50+49); i++)
          fprintf(out_file,"%lf\t",posi_estimated_0_Z[i]);
          fprintf(out_file,"]\n");
          }
     fprintf(out_file,"y%i=[",aux);
     for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
     fprintf(out_file,"%lf\t",posi_estimated_0_Z[i]);
     fprintf(out_file,"]\n");

     fprintf(out_file,"y=[");
     for(l=0; l<=aux; l++)
     fprintf(out_file,"y%i\t",l);
```

```
     fprintf(out_file,"]\n");

     for (l=0; l<=(aux-1); l++)
          {
          fprintf(out_file,"z%i=[",l);
          for (i=l*50; i<=(l*50+49); i++)
          fprintf(out_file,"%lf\t",vase);
          fprintf(out_file,"]\n");
          }
     fprintf(out_file,"z%i=[",aux);
     for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
     fprintf(out_file,"%lf\t",vase);
     fprintf(out_file,"]\n");

     fprintf(out_file,"z=[");
     for(l=0; l<=aux; l++)
     fprintf(out_file,"z%i\t",l);
     fprintf(out_file,"]\n");

     fprintf(out_file,"plot(x,y,'k-',x,z,'k--')\n");
     fprintf(out_file,"title('position on Z')\n");
     fprintf(out_file,"xlabel('time (seconds)')\n");
     fprintf(out_file,"ylabel('Position on z (meter)')\n");

     fclose(out_file);



     out_file=fopen("error_x.m","w");

     for (l=0; l<=(aux-1); l++)
          {
          fprintf(out_file,"x%i=[",l);
          for (i=l*50; i<=(l*50+49); i++)
          fprintf(out_file,"%i\t",i);
          fprintf(out_file,"]\n");
          }
     fprintf(out_file,"x%i=[",aux);
     for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
     fprintf(out_file,"%i\t",i);
     fprintf(out_file,"]\n");

     fprintf(out_file,"x=[");
     for(l=0; l<=aux; l++)
     fprintf(out_file,"x%i\t",l);
     fprintf(out_file,"]\n");

     for (l=0; l<=(aux-1); l++)
          {
          fprintf(out_file,"y%i=[",l);
          for (i=l*50; i<=(l*50+49); i++)
          fprintf(out_file,"%lf\t",error_0_x[i]);
          fprintf(out_file,"]\n");
          }
     fprintf(out_file,"y%i=[",aux);
     for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
     fprintf(out_file,"%lf\t",error_0_x[i]);
```

```
fprintf(out_file,"]\n");

fprintf(out_file,"y=[");
for(l=0; l<=aux; l++)
fprintf(out_file,"y%i\t",l);
fprintf(out_file,"]\n");

for (l=0; l<=(aux-1); l++)
      {
      fprintf(out_file,"w%i=[",l);
      for (i=l*50; i<=(l*50+49); i++)
      fprintf(out_file,"%lf\t",expected_error_x[i]);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"w%i=[",aux);
for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
fprintf(out_file,"%lf\t",expected_error_x[i]);
fprintf(out_file,"]\n");

fprintf(out_file,"w=[");
for(l=0; l<=aux; l++)
fprintf(out_file,"w%i\t",l);
fprintf(out_file,"]\n");

for (l=0; l<=(aux-1); l++)
      {
      fprintf(out_file,"v%i=[",l);
      for (i=l*50; i<=(l*50+49); i++)
      fprintf(out_file,"%lf\t",-expected_error_x[i]);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"v%i=[",aux);
for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
fprintf(out_file,"%lf\t",-expected_error_x[i]);
fprintf(out_file,"]\n");

fprintf(out_file,"v=[");
for(l=0; l<=aux; l++)
fprintf(out_file,"v%i\t",l);
fprintf(out_file,"]\n");

fprintf(out_file,"plot(x,y,'k',x,v,'k--',x,w,'k--')\n");
fprintf(out_file,"title('Error on X')\n");
fprintf(out_file,"xlabel('time (second)')\n");
fprintf(out_file,"ylabel('error on x (meter)')\n");

fclose(out_file);


out_file=fopen("error_y.m","w");

for (l=0; l<=(aux-1); l++)
      {
      fprintf(out_file,"x%i=[",l);
      for (i=l*50; i<=(l*50+49); i++)
      fprintf(out_file,"%i\t",i);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"x%i=[",aux);
for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
fprintf(out_file,"%i\t",i);
fprintf(out_file,"]\n");

fprintf(out_file,"x=[");
for(l=0; l<=aux; l++)
fprintf(out_file,"x%i\t",l);
fprintf(out_file,"]\n");

for (l=0; l<=(aux-1); l++)
      {
      fprintf(out_file,"y%i=[",l);
      for (i=l*50; i<=(l*50+49); i++)
      fprintf(out_file,"%lf\t",error_0_y[i]);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"y%i=[",aux);
for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
fprintf(out_file,"%lf\t",error_0_y[i]);
fprintf(out_file,"]\n");

fprintf(out_file,"y=[");
for(l=0; l<=aux; l++)
fprintf(out_file,"y%i\t",l);
fprintf(out_file,"]\n");

for (l=0; l<=(aux-1); l++)
      {
      fprintf(out_file,"w%i=[",l);
      for (i=l*50; i<=(l*50+49); i++)
      fprintf(out_file,"%lf\t",expected_error_y[i]);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"w%i=[",aux);
for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
fprintf(out_file,"%lf\t",expected_error_y[i]);
fprintf(out_file,"]\n");

fprintf(out_file,"w=[");
for(l=0; l<=aux; l++)
fprintf(out_file,"w%i\t",l);
fprintf(out_file,"]\n");

for (l=0; l<=(aux-1); l++)
      {
      fprintf(out_file,"v%i=[",l);
      for (i=l*50; i<=(l*50+49); i++)
      fprintf(out_file,"%lf\t",-expected_error_y[i]);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"v%i=[",aux);
for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
fprintf(out_file,"%lf\t",-expected_error_y[i]);
fprintf(out_file,"]\n");
fprintf(out_file,"v=[");
for(l=0; l<=aux; l++)
fprintf(out_file,"v%i\t",l);
fprintf(out_file,"]\n");

fprintf(out_file,"plot(x,y,'k',x,v,'k--',x,w,'k--')\n");
fprintf(out_file,"title('Error on Y')\n");
fprintf(out_file,"xlabel('time (second)')\n");
fprintf(out_file,"ylabel('error on y (meter)')\n");

fclose(out_file);


out_file=fopen("error_z.m","w");

for (l=0; l<=(aux-1); l++)
      {
      fprintf(out_file,"x%i=[",l);
      for (i=l*50; i<=(l*50+49); i++)
      fprintf(out_file,"%i\t",i);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"x%i=[",aux);
for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
fprintf(out_file,"%i\t",i);
fprintf(out_file,"]\n");

fprintf(out_file,"x=[");
for(l=0; l<=aux; l++)
fprintf(out_file,"x%i\t",l);
fprintf(out_file,"]\n");

for (l=0; l<=(aux-1); l++)
      {
      fprintf(out_file,"y%i=[",l);
      for (i=l*50; i<=(l*50+49); i++)
      fprintf(out_file,"%lf\t",error_0_z[i]);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"y%i=[",aux);
for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
fprintf(out_file,"%lf\t",error_0_z[i]);
fprintf(out_file,"]\n");

fprintf(out_file,"y=[");
for(l=0; l<=aux; l++)
fprintf(out_file,"y%i\t",l);
fprintf(out_file,"]\n");

for (l=0; l<=(aux-1); l++)
      {
      fprintf(out_file,"w%i=[",l);
      for (i=l*50; i<=(l*50+49); i++)
      fprintf(out_file,"%lf\t",expected_error_z[i]);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"w%i=[",aux);
```

```
for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
fprintf(out_file,"%lf\t",expected_error_z[i]);
fprintf(out_file,"]\n");

fprintf(out_file,"w=[");
for(l=0; l<=aux; l++)
fprintf(out_file,"w%i\t",l);
fprintf(out_file,"]\n");

for (l=0; l<=(aux-1); l++)
      {
      fprintf(out_file,"v%i=[",l);
      for (i=l*50; i<=(l*50+49); i++)
      fprintf(out_file,"%lf\t",-expected_error_z[i]);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"v%i=[",aux);
for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
fprintf(out_file,"%lf\t",-expected_error_z[i]);
fprintf(out_file,"]\n");

fprintf(out_file,"v=[");
for(l=0; l<=aux; l++)
fprintf(out_file,"v%i\t",l);
fprintf(out_file,"]\n");

fprintf(out_file,"plot(x,y,'k',x,v,'k--',x,w,'k--')\n");
fprintf(out_file,"title('Error on Z')\n");
fprintf(out_file,"xlabel('time (second)')\n");
fprintf(out_file,"ylabel('error on z (meter)')\n");

fclose(out_file);

out_file=fopen("error_global.m","w");

for (l=0; l<=(aux-1); l++)
      {
      fprintf(out_file,"x%i=[",l);
      for (i=l*50; i<=(l*50+49); i++)
      fprintf(out_file,"%i\t",i);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"x%i=[",aux);
for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
fprintf(out_file,"%i\t",i);
fprintf(out_file,"]\n");

fprintf(out_file,"x=[");
for(l=0; l<=aux; l++)
fprintf(out_file,"x%i\t",l);
fprintf(out_file,"]\n");

for (l=0; l<=(aux-1); l++)
      {
      fprintf(out_file,"y%i=[",l);
      for (i=l*50; i<=(l*50+49); i++)
      fprintf(out_file,"%lf\t",error_global_0[i]);
```

```
          fprintf(out_file,"]\n");
          }
      fprintf(out_file,"y%i=[",aux);
      for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
      fprintf(out_file,"%lf\t",error_global_0[i]);
      fprintf(out_file,"]\n");

      fprintf(out_file,"y=[");
      for(l=0; l<=aux; l++)
      fprintf(out_file,"y%i\t",l);
      fprintf(out_file,"]\n");

      for (l=0; l<=(aux-1); l++)
          {
          fprintf(out_file,"w%i=[",l);
          for (i=l*50; i<=(l*50+49); i++)
          fprintf(out_file,"%lf\t",expected_error_global[i]);
          fprintf(out_file,"]\n");
          }
      fprintf(out_file,"w%i=[",aux);
      for (i=aux*50; i<=((duration-1)*NB_ITE); i++)
      fprintf(out_file,"%lf\t",expected_error_global[i]);
      fprintf(out_file,"]\n");

      fprintf(out_file,"w=[");
      for(l=0; l<=aux; l++)
      fprintf(out_file,"w%i\t",l);
      fprintf(out_file,"]\n");


      fprintf(out_file,"plot(x,y,'k',x,w,'k--')\n");
      fprintf(out_file,"title('Global Error')\n");
      fprintf(out_file,"xlabel('time (second)')\n");
      fprintf(out_file,"ylabel('global error (meter)')\n");

      fclose(out_file);
      }



void write_error_satellite(void)
      {
      int l, i,j, aux;
      FILE *out_file;
      char *name[NB_SAT];

      name[0]="error_satel_0.m";
      name[1]="error_satel_1.m";
      name[2]="error_satel_2.m";
      name[3]="error_satel_3.m";
      name[4]="error_satel_4.m";
      name[5]="error_satel_5.m";
      name[6]="error_satel_6.m";
      name[7]="error_satel_7.m";
      name[8]="error_satel_8.m";
      name[9]="error_satel_9.m";
      name[10]="error_satel_10.m";
```

```
      name[11]="error_satel_11.m";
      name[12]="error_satel_12.m";
      name[13]="error_satel_13.m";
      name[14]="error_satel_14.m";

      aux=(int)(duration/50);

      for(l=0; l<=(number_satel_in_view-1); l++)
      {
      out_file=fopen(name[l],"w");

      for (i=0; i<=(aux-1); i++)
          {
          fprintf(out_file,"x%i=[",i);
          for (j=i*50; j<=(i*50+49); j++)
          fprintf(out_file,"%i\t",j);
          fprintf(out_file,"]\n");
          }
      fprintf(out_file,"x%i=[",aux);
      for (j=aux*50; j<=(duration-1); j++)
      fprintf(out_file,"%i\t",j);
      fprintf(out_file,"]\n");
      fprintf(out_file,"x=[");
      for(i=0; i<=aux; i++)
      fprintf(out_file,"x%i\t",i);
      fprintf(out_file,"]\n");


      for (i=0; i<=(aux-1); i++)
          {
          fprintf(out_file,"a%i=[",i);
          for (j=i*50; j<=(i*50+49); j++)
          fprintf(out_file,"%lf\t",error_ephe_0[l][j]);
          fprintf(out_file,"]\n");
          }
      fprintf(out_file,"a%i=[",aux);
      for (j=aux*50; j<=(duration-1); j++)
      fprintf(out_file,"%lf\t",error_ephe_0[l][j]);
      fprintf(out_file,"]\n");
      fprintf(out_file,"yephe=[");
      for(i=0; i<=aux; i++)
      fprintf(out_file,"a%i\t",i);
      fprintf(out_file,"]\n");

      for (i=0; i<=(aux-1); i++)
          {
          fprintf(out_file,"b%i=[",i);
          for (j=i*50; j<=(i*50+49); j++)
          fprintf(out_file,"%lf\t",error_clock_0[l][j]);
          fprintf(out_file,"]\n");
          }
      fprintf(out_file,"b%i=[",aux);
      for (j=aux*50; j<=(duration-1); j++)
      fprintf(out_file,"%lf\t",error_clock_0[l][j]);
      fprintf(out_file,"]\n");
      fprintf(out_file,"yclock=[");
      for(i=0; i<=aux; i++)
```

```
fprintf(out_file,"b%i\t",i);
fprintf(out_file,"]\n");

for (i=0; i<=(aux-1); i++)
      {
      fprintf(out_file,"c%i=[",i);
      for (j=i*50; j<=(i*50+49); j++)
      fprintf(out_file,"%lf\t",error_iono_0[l][j]);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"c%i=[",aux);
for (j=aux*50; j<=(duration-1); j++)
fprintf(out_file,"%lf\t",error_iono_0[l][j]);
fprintf(out_file,"]\n");
fprintf(out_file,"yiono=[");
for(i=0; i<=aux; i++)
fprintf(out_file,"c%i\t",i);
fprintf(out_file,"]\n");

for (i=0; i<=(aux-1); i++)
      {
      fprintf(out_file,"d%i=[",i);
      for (j=i*50; j<=(i*50+49); j++)
      fprintf(out_file,"%lf\t",error_tropo_0[l][j]);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"d%i=[",aux);
for (j=aux*50; j<=(duration-1); j++)
fprintf(out_file,"%lf\t",error_tropo_0[l][j]);
fprintf(out_file,"]\n");
fprintf(out_file,"ytropo=[");
for(i=0; i<=aux; i++)
fprintf(out_file,"d%i\t",i);
fprintf(out_file,"]\n");

for (i=0; i<=(aux-1); i++)
      {
      fprintf(out_file,"e%i=[",i);
      for (j=i*50; j<=(i*50+49); j++)
      fprintf(out_file,"%lf\t",error_mupath_0[l][j]);
      fprintf(out_file,"]\n");
      }
fprintf(out_file,"e%i=[",aux);
for (j=aux*50; j<=(duration-1); j++)
fprintf(out_file,"%lf\t",error_mupath_0[l][j]);
fprintf(out_file,"]\n");
fprintf(out_file,"ymupath=[");
for(i=0; i<=aux; i++)
fprintf(out_file,"e%i\t",i);
fprintf(out_file,"]\n");

for (i=0; i<=(aux-1); i++)
      {
      fprintf(out_file,"f%i=[",i);
      for (j=i*50; j<=(i*50+49); j++)
      fprintf(out_file,"%lf\t",error_noise_0[l][j]);
      fprintf(out_file,"]\n");
```

```
            }
      fprintf(out_file,"f%i=[",aux);
      for (j=aux*50; j<=(duration-1); j++)
      fprintf(out_file,"%lf\t",error_noise_0[l][j]);
      fprintf(out_file,"]\n");
      fprintf(out_file,"ynoise=[");
      for(i=0; i<=aux; i++)
      fprintf(out_file,"f%i\t",i);
      fprintf(out_file,"]\n");


      for (i=0; i<=(aux-1); i++)
            {
            fprintf(out_file,"y%i=[",i);
            for (j=i*50; j<=(i*50+49); j++)
            fprintf(out_file,"%lf\t",error_satel_0[l][j]);
            fprintf(out_file,"]\n");
            }
      fprintf(out_file,"y%i=[",aux);
      for (j=aux*50; j<=(duration-1); j++)
      fprintf(out_file,"%lf\t",error_satel_0[l][j]);
      fprintf(out_file,"]\n");
      fprintf(out_file,"ytotal=[");
      for(i=0; i<=aux; i++)
      fprintf(out_file,"y%i\t",i);
      fprintf(out_file,"]\n");

      fprintf(out_file,"plot(x,yephe,'k:',x,yclock,'k-.',x,yiono,'k--
',x,ytropo,'k-x',x,ymupath,'k-o',x,ynoise,'k-*',x,ytotal,'k')\n");

      fprintf(out_file,"n=[");
      fprintf(out_file,"%i",(l+1));
      fprintf(out_file,"]\n");
      fprintf(out_file,"title(['error on satèllite number
',num2str(n)])\n");
      fprintf(out_file,"xlabel('acquisition')\n");
      fprintf(out_file,"ylabel('error (meter)')\n");
//    fprintf(out_file,"print -dbmp256 error_sat_%i",(l+1));
      fclose(out_file);
      }
}

void storage(int k)
      {
      posi_estimated_0_X[k]=user.posi_estimated_0_X;
      posi_estimated_0_Y[k]=user.posi_estimated_0_Y;
      posi_estimated_0_Z[k]=user.posi_estimated_0_Z;
      }

void compute_error(int k)
      {
      error_0_x[k]=user.posi_estimated_0_X-user.posi_true_X;
      error_0_y[k]=user.posi_estimated_0_Y-user.posi_true_Y;
      error_0_z[k]=user.posi_estimated_0_Z-user.posi_true_Z;
      error_0_t[k]=user.posi_estimated_0_T-user.posi_true_T;
error_global_0[k]=sqrt(sqr(error_0_x[k])+sqr(error_0_y[k])+sqr(error_
0_z[k]));
```

```
      }

void numeric_output(void)
      {
      int i,j,k;
      double vase1, vase2, vase3, vase4, vase5;
      double vase1bis, vase2bis, vase3bis, vase4bis, vase5bis;
      double aux1, aux2, aux3, aux4, aux5;
      FILE *numeric_data;
      double average_error_x,average_error_y, average_error_z,
average_error_t, average_error_global, ET_x, ET_y, ET_z, ET_t,
ET_global;

      vase1=vase2=vase3=vase4=vase5=0;
      for (i=0; i<=((duration-1)*NB_ITE); i++)
            {
            vase1=vase1+error_0_x[i];
            vase2=vase2+error_0_y[i];
            vase3=vase3+error_0_z[i];
            vase4=vase4+error_0_t[i];
            vase5=vase5+error_global_0[i];
            }

      average_error_x=vase1/((duration-1)*NB_ITE);
      average_error_y=vase2/((duration-1)*NB_ITE);
      average_error_z=vase3/((duration-1)*NB_ITE);
      average_error_t=vase4/((duration-1)*NB_ITE);
      average_error_global=vase5/((duration-1)*NB_ITE);

      for (i=0; i<=((duration-1)*NB_ITE); i++)
            {
            vase1bis=vase1bis+sqr(error_0_x[i]-average_error_x);
            vase2bis=vase2bis+sqr(error_0_y[i]-average_error_y);
            vase3bis=vase3bis+sqr(error_0_z[i]-average_error_z);
            vase4bis=vase4bis+sqr(error_0_t[i]-average_error_t);
            vase5bis=vase5bis+sqr(error_global_0[i]-
average_error_global);
            }

      ET_x=sqrt(vase1bis/((duration-1)*NB_ITE-1));
      ET_y=sqrt(vase2bis/((duration-1)*NB_ITE-1));
      ET_z=sqrt(vase3bis/((duration-1)*NB_ITE-1));
      ET_t=sqrt(vase4bis/((duration-1)*NB_ITE-1));
      ET_global=sqrt(vase5bis/((duration-1)*NB_ITE-1));

      numeric_data=fopen("numeric_data.txt","w");
      fprintf(numeric_data,"\tOutput data from the simulation\n");
      fprintf(numeric_data,"\t-------------------------------\n\n");

      for (k=0; k<=(duration-1); k++)
      {
      fprintf(numeric_data,"\tacquisition number\t%i\n",k);
      for (j=0; j<=(NB_SAT-1); j++)
      {
      fprintf(numeric_data,"satellite number\t");
      fprintf(numeric_data,"%i\t",j);
      if (satel[j][k].view) fprintf(numeric_data,"in view\n");
```

```
      else fprintf(numeric_data,"not in view\n");
      }
      fprintf(numeric_data,"\n\n");
      }

      fprintf(numeric_data,"\t\t Error average\n\n");
      fprintf(numeric_data,"\t\taverage error\t\tstandart
deviation\n");
      fprintf(numeric_data,"x\t\t%lf\t\t%lf\n",average_error_x,ET_x);
      fprintf(numeric_data,"y\t\t%lf\t\t%lf\n",average_error_y,ET_y);
      fprintf(numeric_data,"z\t\t%lf\t\t%lf\n",average_error_z,ET_z);
      fprintf(numeric_data,"t\t\t%lf\t\t%lf\n",average_error_t,ET_t);
fprintf(numeric_data,"global\t\t%lf\t\t%lf\n",average_error_global,ET
_global);


      fprintf(numeric_data,"\n\n\t\t\t\t DOP\n\n");
      fprintf(numeric_data,"    XDOP      YDOP\t  VDOP\t
HDOP\tTDOP\t  PDOP\t\tGDOP\n");
      for (k=0; k<=((duration-1)*NB_ITE); k++)
            {
fprintf(numeric_data,"+-----------+-----------+-----------+--
---------+-----------+-----------+\n");
fprintf(numeric_data,"|%f  |%f  |%f  |%f  |%f  |%f  |%f
|\n",XDOP[k], YDOP[k], VDOP[k], HDOP[k], TDOP[k], PDOP[k], GDOP[k]);
            }
fprintf(numeric_data,"+-----------+-----------+-----------+--
---------+-----------+-----------+\n");
      }
```

```
#ifndef _ER_SAT_
#define _ER_SAT_

#include "read.h"
#include "white_noise.h"
#include "egnos.h"

void compute_error_iono_0(int k);
void compute_error_iono_2(int k);
void compute_error_satel(int k);

#endif _ER_SAT_
```

```
#include "err_sat.h"

//-----------------------------------------------------
//computation of ionospheric delay by dual frequency data
//-----------------------------------------------------

void compute_error_iono_0(int k)
      {
      int i,j;

      i=0;
      for(j=0; j<=(NB_SAT-1); j++)
      if (satel[j][k].view)
      {
      error_iono_0[i][k]=c*error_0.compute_T_Iono(j,k);
      i++;
      }
      }

//-----------------------------------------------
//computation of ionospheric delay using Klobuchar
//-----------------------------------------------

void compute_error_iono_2(int k)
      {
      int i,j;

      i=0;
      for(j=0; j<=(NB_SAT-1); j++)
      if (satel[j][k].view)
      {
      correction_iono[i][k]=correction_iono[j][k];
      i++;
      }

      for(i=0; i<=(number_satel_in_view-1); i++)
      {
      error_iono_0[i][k]=c*error_1.compute_T_Iono(i,k);
      }
      }



//-----------------------------------------
//compuation of the error on satellite signal
//-----------------------------------------

void compute_error_satel(int k)
      {
      int j;
      CUereAssess signal_error;

      if ((choice_iono==2) || (choice_tropo==2) || (choice_ephe==2)
|| (choice_clock==2) || (choice_mupath==2) || (choice_noise==2))
      for (j=0; j<=(number_satel_in_view-1); j++)
      {
      signal_error.computation(j,k);
```

```
      if (choice_iono==2) error_iono_0[j][k]=signal_error.iono;
      if (choice_tropo==2) error_tropo_0[j][k]=signal_error.tropo;
      if (choice_ephe==2) error_ephe_0[j][k]=signal_error.ephemeris;
      if (choice_clock==2) error_clock_0[j][k]=signal_error.clock;
      if (choice_mupath==2) error_mupath_0[j][k]=signal_error.mupath;
      if (choice_noise==2) error_noise_0[j][k]=signal_error.noise;
      }

      switch (choice_iono){
            case 1:
      for (j=0; j<=(number_satel_in_view-1); j++)
error_iono_0[j][k]=0;
            break;
            case 3: compute_error_iono_2(k);
            break;
            case 4: compute_error_egnos(k);
            break;
            case 5: compute_error_iono_0(k);
            break;
            }

      if (choice_tropo==1)
      for (j=0; j<=(number_satel_in_view-1); j++)
      error_tropo_0[j][k]=0;

      if (choice_ephe==1)
      for (j=0; j<=(number_satel_in_view-1); j++)
      error_ephe_0[j][k]=0;

      if (choice_clock==1)
      for (j=0; j<=(number_satel_in_view-1); j++)
      error_clock_0[j][k]=0;

      if (choice_mupath==1)
      for (j=0; j<=(number_satel_in_view-1); j++)
      error_mupath_0[j][k]=0;

      if (choice_noise==1)
      for (j=0; j<=(number_satel_in_view-1); j++)
      error_noise_0[j][k]=0;


      for (j=0; j<=(number_satel_in_view-1); j++)
      {
      error_satel_0[j][k]=error_iono_0[j][k] + error_tropo_0[j][k] +
error_ephe_0[j][k] + error_clock_0[j][k] +  error_mupath_0[j][k]
+error_noise_0[j][k];
      }

      }
```

```
#include "in_data.h"


void input_data(void)
      {
      int coordinates_choice;
      double longitude0,latitude0,longitudeT,latitudeT;

      cout<<"DO YOU WANT:\n";
      cout<<"TO ENTER YOUR INPUT DATA                  1\n";
      cout<<"TO READ THE INPUT DATA FROM SCENARIO_1         2\n";
      cin>>choice_scen;


      switch (choice_scen) {

      case 1:
      {
      cout<< "WHICH TYPE OF SATELLITES DATA WOULD YOU LIKE TO USE FOR
TE SIMULATION\n";
      cout<< "DATA FROM STK FILES           1\n";
      cout<< "DATA FROM REAL DATA FILES     2\n";
      cin>> choice_satel_data;

      cout<<"\nTHE STEP TIME OF THE SIMULATION IS 1 SECOND, AND THE
PROGRAM MAKES TWO ITERATIONS PER SECOND. MAKE SUR THAT YOUR DATA
FILES ARE COMPATIBLE WITH THE PERIODE OF YOUR SIMULATION.\n";
      cout<<"\nENTER THE DURATION OF THE SIMULATION IN SECONDS\t";
      cin>>duration;

      cout<<"\nENTER THE INITIAL POSITION OF THE RECEIVER";
      cout<<"\nFIRST ENTER THE TYPE OF CO-ORDINATES CHOSEN:\n";
      cout<<"(X,Y,Z) IN THE TERRESTRIAL REFERENCE:                1";
      cout<<"\n(longitude,latitude) IN THE TERRESTRIAL REFERENCE:
2\t";
      cin>>coordinates_choice;

      switch (coordinates_choice){
            case 1 : {
                  cout<<"\nENTER X0 IN METERS\t";
                  cin>>user.posi_estimated_0_X;
                  user.posi_estimated_1_X=user.posi_estimated_0_X;
                  cout<<"\nENTER Y0 IN METERS\t";
                  cin>>user.posi_estimated_0_Y;
                  user.posi_estimated_1_Y=user.posi_estimated_0_Y;
                  cout<<"\nENTER Z0 IN METERS\t";
                  cin>>user.posi_estimated_0_Z;
                  user.posi_estimated_1_Z=user.posi_estimated_0_Z;
                  cout<<"\nENTER BIAS IN SECONDS\t";
                  cin>>user.bias_estimated;
                  }
                  break;
            case 2 : {
                  cout<<"\nENTER THE LONGITUDE IN RADIANS\t";
                  cin>>longitude0;
                  cout<<"\nENTER THE LATITUDE IN RADIANS\t";
                  cin>>latitude0;
```

```
      user.posi_estimated_0_X=R_EARTH*cos(latitude0)*cos(longitude0);

      user.posi_estimated_0_Y=R_EARTH*cos(latitude0)*sin(longitude0);
            user.posi_estimated_0_Z=R_EARTH*sin(latitude0);

      user.posi_estimated_1_X=R_EARTH*cos(latitude0)*cos(longitude0);

      user.posi_estimated_1_Y=R_EARTH*cos(latitude0)*sin(longitude0);
            user.posi_estimated_1_Z=R_EARTH*sin(latitude0);
            cout<<"\nENTER CLOCK BIAS IN SECONDS\t";
                  cin>>user.bias_estimated;
                  }
            }
      //true position parameters
      cout<<"\nENTER THE TRUE POSITION OF THE RECEIVER";
      cout<<"\nFIRST ENTER THE TYPE OF CO-ORDINATES CHOSEN:\n";
      cout<<"(X,Y,Z) IN THE TERRESTRIAL REFERENCE:               1";
      cout<<"\n(longitude,latitude) IN THE TERRESTRIAL REFERENCE:
2\t";
      cin>>coordinates_choice;

      switch (coordinates_choice){
            case 1 : {
                  cout<<"\nENTER XT IN METERS\t";
                  cin>>user.posi_true_X;
                  cout<<"\nENTER YT IN METERS\t";
                  cin>>user.posi_true_Y;
                  cout<<"\nENTER ZT IN METERS\t";
                  cin>>user.posi_true_Z;
                  cout<<"\nENTER BIAS IN SECONDS\t";
                  cin>>user.bias_true;
                  }
                  break;
            case 2 : {
                  cout<<"\nENTER THE LONGITUDE IN RADIANS\t";
                  cin>>longitudeT;
                  cout<<"\nENTER THE LATITUDE IN RADIANS\t";
                  cin>>latitudeT;
            user.posi_true_X=R_EARTH*cos(latitudeT)*cos(longitudeT);
            user.posi_true_Y=R_EARTH*cos(latitudeT)*sin(longitudeT);
            user.posi_true_Z=R_EARTH*sin(latitudeT);
            cout<<"\nENTER CLOCK BIAS IN SECONDS\t";
                  cin>>user.bias_true;
                  }
                  break;
            }
      cout<<"\n";

      cout<<"ENTER THE TYPE OF ERROR YOU WANT TO USE TO DETERIORATE
THE SATELLITES' SIGNALS\n\n";
      cout<<"\tIONOSPHERIC DELAY\n";
      cout<<"NO ERROR                    1\n";
      cout<<"WHITE NOISE                 2\n";
      cout<<"KLOBUCHAR CORRECTION        3\n";
      cout<<"EGNOS CORRECTION            4\n";
      cout<<"REAL IONOSPHERIC ERROR        5\n";
```

```
        cout<<"\nWARNING: the solutions 3, 4 and 5 require the use of
real data files.\n\n";
        cin>>choice_iono;

        cout<<"\tTROPOSPHERIC DELAY\n";
        cout<<"NO ERROR                    1\n";
        cout<<"WHITE NOISE                 2\n";
        cin>>choice_tropo;

        cout<<"\tEPHEMERIS ERROR\n";
        cout<<"NO ERROR                    1\n";
        cout<<"WHITE NOISE                 2\n";
        cin>>choice_ephe;

        cout<<"\tCLOCK ERROR\n";
        cout<<"NO ERROR                    1\n";
        cout<<"WHITE NOISE                 2\n";
        cin>>choice_clock;

        cout<<"\tMULTIPATH\n";
        cout<<"NO ERROR                    1\n";
        cout<<"WHITE NOISE                 2\n";
        cin>>choice_mupath;

        cout<<"\tNOISE\n";
        cout<<"NO ERROR                    1\n";
        cout<<"WHITE NOISE                 2\n";
        cin>>choice_noise;

        cout<<"ENTER THE VALUES OF SIGMA:";
        cout<<"\nEPHEMERIS\t";
        cin>>sigma_ephemeris;
        cout<<"\nCLOCK\t";
        cin>>sigma_clock;
        cout<<"\nIONOSPHERIC\t";
        cin>>sigma_iono;
        cout<<"\nTROPOSPHERIC\t";
        cin>>sigma_tropo;
        cout<<"\nMULTIPATH\t";
        cin>>sigma_mupath;
        cout<<"\nNOISE\t";
        cin>>sigma_noise;

        }
        break;

        case 2:
        {
        lecture_scen1();
        }
        break;
        }

        }

void lecture_scen1(void)
        {
```

```
        FILE *input;
        char string_1[8192];
        char *token;

        input=fopen("scenario1.txt","r");

        fgets(string_1, 8191, input);
        token = strtok(string_1, ":");
        token = strtok(NULL, ":");
        sscanf(token,"%i",&choice_satel_data);

        fgets(string_1, 8191, input);

        fgets(string_1, 8191, input);
        token = strtok(string_1, ":");
        token = strtok(NULL, ":");
        sscanf(token,"%i",&duration);

        fgets(string_1, 8191, input);
        fgets(string_1, 8191, input);
        fgets(string_1, 8191, input);

        token = strtok(string_1, ":");
        token = strtok(NULL, ":");
        sscanf(token,"%lf",&user.posi_estimated_0_X);
        sscanf(token,"%lf",&user.posi_estimated_1_X);

        fgets(string_1, 8191, input);
        token = strtok(string_1, ":");
        token = strtok(NULL, ":");
        sscanf(token,"%lf",&user.posi_estimated_0_Y);
        sscanf(token,"%lf",&user.posi_estimated_1_Y);

        fgets(string_1, 8191, input);
        token = strtok(string_1, ":");
        token = strtok(NULL, ":");
        sscanf(token,"%lf",&user.posi_estimated_0_Z);
        sscanf(token,"%lf",&user.posi_estimated_1_Z);

        fgets(string_1, 8191, input);
        token = strtok(string_1, ":");
        token = strtok(NULL, ":");
        sscanf(token,"%lf",&user.bias_estimated);
        user.posi_estimated_0_T=c*user.bias_estimated;
        user.posi_estimated_1_T=c*user.bias_estimated;

        fgets(string_1, 8191, input);
        fgets(string_1, 8191, input);
        fgets(string_1, 8191, input);

        token = strtok(string_1, ":");
        token = strtok(NULL, ":");
        sscanf(token,"%lf",&user.posi_true_X);

        fgets(string_1, 8191, input);
        token = strtok(string_1, ":");
        token = strtok(NULL, ":");
```

```
sscanf(token,"%lf",&user.posi_true_Y);

fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%lf",&user.posi_true_Z);

fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%lf",&user.bias_true);
user.posi_true_T=c*user.bias_true;

fgets(string_1, 8191, input);
fgets(string_1, 8191, input);
fgets(string_1, 8191, input);

fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%i",&choice_iono);

fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%i",&choice_tropo);

fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%i",&choice_ephe);

fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%i",&choice_clock);

fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%i",&choice_mupath);

fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%i",&choice_noise);

fgets(string_1, 8191, input);
fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%lf",&sigma_ephemeris);

fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%lf",&sigma_clock);
```

```
fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%lf",&sigma_iono);

fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%lf",&sigma_tropo);

fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%lf",&sigma_mupath);

fgets(string_1, 8191, input);
token = strtok(string_1, ":");
token = strtok(NULL, ":");
sscanf(token,"%lf",&sigma_noise);

fclose(input);

}
```

```
#ifndef _IN_DATA_
#define _IN_DATA_

#include <math.h>
#include "variable_glob.h"

void input_data(void);
void lecture_scen1(void);

#endif _IN_DATA_
```

```
#ifndef _DUNSS_
#define _DUNSS_

#include "compute_K.h"
#include "err_sat.h"
#include "in_data.h"
#include "out_data.h"

#endif _DUNSS_
```

```
#include "DUNSS.h"

int main(void)
      {
      int k, i, ite;

      input_data();
      write_affectation_0();

      if(choice_satel_data==2 || choice_iono==3 || choice_iono==4 ||
choice_iono==5) read_data();
      if(choice_satel_data==1) read_stk();
      if (choice_iono==4) remplissage_egnos();

      for(k=0; k<=(duration-1); k++)
      {
      write_affectation_2(k);
      compute_el_az_ellipt(k);
      write_affectation_4(k);
      compute_error_satel(k);
      write_affectation_1(k);

      for(ite=0; ite<=(NB_ITE-1);ite++)
            {
            write_affectation_0();

            compute_G_0(k);
            compute_W_0(k);
            compute_K_0();

            compute_A();
            compute_DOP(NB_ITE*k+ite);
            compute_expected_error(NB_ITE*k+ite);

            for (i=0; i<=3; i++)
      delta_x_estimated_0_NB_SAT[i][0]=delta_x_estimated_0[i];

multiplication(G_0,delta_x_estimated_0_NB_SAT, number_satel_in_view,
1, 4);

            addition(product,error_NB_SAT_0, number_satel_in_view,
1);
            for (i=0; i<=(number_satel_in_view-1); i++)
            delta_ro_0[i][0]=sum[i][0];

            multiplication(K_0,delta_ro_0, 4, 1,
number_satel_in_view);
            for (i=0; i<=3; i++)
            delta_x_estimated_0[i]=product[i][0];

      user.posi_estimated_0_X=user.posi_estimated_0_X-
delta_x_estimated_0[0];
      user.posi_estimated_0_Y=user.posi_estimated_0_Y-
delta_x_estimated_0[1];
      user.posi_estimated_0_Z=user.posi_estimated_0_Z-
delta_x_estimated_0[2];
      user.posi_estimated_0_T=user.posi_estimated_0_T-
delta_x_estimated_0[3];

            storage(NB_ITE*k+ite);
            compute_error(NB_ITE*k+ite);
            }
      }

      global_storage();
      write_error_satellite();
      numeric_output();

      return 0;
      }
```