Processing and Analysis of Positional Time-Series

Nuno Alexandre Branco Rosa Pedro

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática
(2º ciclo de estudos)

Orientador: Prof. Rui Fernandes
Co-orientador: Prof. Paul Crocker

Junho de 2024

Declaração de Integridade

Eu, Nuno Alexandre Branco Rosa Pedro, que abaixo assino, estudante com o número de inscrição m11570 do Mestrado em Engenharia Informática da Faculdade de Engenharia, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o Código de Integridades da Universidade da Beira Interior.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 11/06/2024

Assinado por: **Nuno Alexandre Branco Rosa Pedro**
Num. de Identificação: 14496875
Data: 2024.06.11 15:23:09+01'00'

Acknowledgements

Em primeiro lugar preciso agradecer obrigatoriamente aos meus pais, Luís e Magda, por sempre me incentivarem e me ajudarem a atingir todos os meus objetivos pessoais, eles em conjunto com a minha família e amigos são a maior e a mais fundamental razão para o meu sucesso.

Agradecer a todos os professores e colegas que me acompanharam ao longo da licenciatura e mestrado, em especial ao Professor Rui Fernandes por acreditar em mim e me ajudar desde a licenciatura a concretizar os meus projetos pessoais.

Também é fundamental agradecer a todas as pessoas que de alguma forma estiveram envolvidos nos anos que passei no SEGAL, foram fundamentais para o meu crescimento enquanto pessoa e profissional.

Resumo

Durante as últimas três décadas, diferentes redes de estações Global Navigation Satellite System (GNSS) têm recolhido informação que permitem obter séries temporais posicionais que podem ser utilizadas em diversas aplicações, como a monitorização da deformação da crusta da terra. A velocidade secular devido ao movimento das placas tectónicas é o principal parâmetro a ser estimado através da série temporal. Componentes como variações sazonais, ruído, *offsets* (devidos a processos físicos como terramotos ou a factores antropogénicos), são componentes adicionais usualmente estimadas através da análise da série temporal, as quais podem complicar o processo de estimativa da velocidade secular. Atualmente existem algumas aplicações computacionais para o processamento de séries temporais que foram desenvolvidos para lidar com a estimação de todos essas componentes, sendo o software Hector um deles, o qual vai ser o alvo deste estudo.

Este estudo concentra-se em duas componentes principais. A primeira componente visa a análise de otimização do Hector, sendo o objetivo primário de estimar e analisar os parâmetros ideais de execução. A segunda componente deste estudo dedica-se na implementação do Hector como um serviço online que permita o seu acesso por Command Line Interface (CLI) e por Graphical User Interface (GUI).

Devido à proliferação de redes de estações GNSS, o Hector tem se tornado um software fundamental e bastante requisitado para o processamento de dados. . A sua otimização através deste estudo surge como uma necessidade tendo em conta que o número de estações tende a aumentar no tempo, bem como a duração das suas séries temporais. Como consequência é imperativo otimizar o tempo de processamento de cada estação.

Palavras-chave

GNSS Time-Series, Hector, GUI, CLI

Resumo Alargado

Nas últimas três décadas as redes GNSS têm adquirido dados que permitem calcular séries temporais das posições das suas estações. A velocidade secular devido ao movimento das placas tectónicas é um dos principais parâmetros a ser estimados através do processamento dos dados recolhidos.

Hector é um pacote de software usado para calcular a velocidade secular através da análise das séries temporais de posições diárias. Após a realização do processamento, são obtidas séries temporais com três componentes de deslocamento. Na série temporal há componentes como variações sazonais, ruído, compensações, esses componentes podem complicar o processo de estimativa da velocidade secular, pelo que, surge a necessidade do uso de software especifico para esse cálculo.

A utilização do Hector é feita em larga escala, sendo alto o número de séries temporais que necessitam de ser processadas em muitos centros de análise, pelo que a economia de tempo se torna fundamental. Com vista a otimizar esse tempo é possível alterar e combinar vários parâmetros. Este estudo sugere algumas alterações que permitem além de uma grande economia de tempo, otimizar este software.

Executar o Hector exige a seleção de diversas opções para diversos parâmetros tendo sido desenvolvida uma aplicação de modo a facilitar a interação com este software.

Esta aplicação possui dois componentes de interação com o Hector. O primeiro é um componente Application Programming Interface (API) que permite a realização de pedidos e a obtenção de respostas através da linha de comando, facilitando a integração em ambientes preexistentes e o processamento de grandes volumes de estações. O segundo componente uma aplicação web, que também possibilita a realização de requisições ao Hector e a obtenção de resultados, além de fornecer as informações necessárias para a utilização da aplicação via linha de comando.

Todas estas ferramentas foram implementadas na aplicação TimeSeries Analysis by SEGAL (TSA), desenvolvida em Hypertext Preprocessor (PHP), JavaScript e HyperText Markup Language (HTML), interagindo com scripts Python e Tcsh que permitem então utilizar uma base de dados Structured Query Language (SQL) em conjunto com a aplicação que completa todas as funcionalidades do aplicativo.

Summary

In the last three decades, GNSS networks have acquired data that allows positional time series of their stations to be calculated. The secular velocity due to the movement of tectonic plates is one of the main parameters to be estimated by processing the collected data, which is normally daily positions of the GNSS stations

Hector is a software package that calculates the secular velocity and other associated parameters using such data. Solutions for the parameters of interest are obtained by analyzing the three components separately. Additional parameters can be seasonal variations, noise, and offsets; these parameters can complicate the process of estimating the secular velocity. Therefore, there is a need to use specific software for this calculation.

Hector is used on a large scale, meaning the number of time series processed can be large. Therefore, time savings become essential. To optimize this time, changing and combining several parameters is possible. This study suggests some changes that allow, in addition to great time savings, to optimize this software.

Running Hector requires the creation of specific configuration files that are sometimes difficult to do by an inexperienced user using commands on a Linux terminal. Therefore, an application was developed to facilitate interaction with Hector.

This application has two interaction components with Hector. The first is an API component that allows a user to make requests and receive responses through a command line interface making it easier to create requests in existing environments with a large numbers of stations. The second component is a GUI, where it is also possible to make requests to Hector and obtain the results.

All these tools were implemented in the TSA application, developed in PHP, JavaScript and HTML, interacting with Python and TC-Shell scripts that then allow the use of a SQL database together with the application that completes all the functionalities of the application.

Abstract

During the last three decades, different networks of GNSS stations have collected information that allows positional time series to be obtained for various applications, such as monitoring the deformation of the Earth crust. The secular velocity due to the movement of tectonic plates is the main parameter to be estimated through the time series. Components such as seasonal variations, noise, offsets (due to physical processes such as earthquakes or anthropogenic factors) are additional components usually estimated through time series analysis, which can complicate the secular velocity estimation process. Currently, some computational applications for processing time series have been developed to estimate all these components, the Hector software being one of them, which will be the target of this study.

This study focuses has two main components. The first component aims to analyze possible Hector optimizations, with the primary objective being to estimate and analyze the ideal execution parameters. The second component of this study is dedicated to implementing Hector as an online service that allows access via a CLI and also via a GUI.

Due to the proliferation of GNSS station networks, Hector has become an essential and highly in demand software for data processing. Therefore optimization of this software emerges as a pressing need, considering the continuous increase in the number of stations and the extension of their time series. Consequently, it is imperative to optimize the processing time of each station.

Keywords

GNSS Time-Series, Hector, GUI, CLI

Contents

Acknowledgements v

Resumo vii

Resumo Alargado ix

Summary xi

Abstract xiii

Contents xv

List of Figures xix

List of Tables xxi

Acronyms and Abbreviations xxiii

1 Introduction 1

1.1 Problem Statement 1

1.2 Motivation 1

1.3 Objectives 2

1.4 Contributions 3

2 Positional Time-Series: Concepts and Backgrounds 5

2.1 Introduction 5

2.2 CORS (Continuously Operating Reference Stations) 5

2.3 Solution Files 5

2.4 Positional Time-Series 6

2.5 Time-Series Analysis 7

2.6 Time-Series Analysis Software 8

2.6.1 Hector 8

2.6.2 CATS 9
2.6.3 LANGBEIN 9
2.6.4 MIDAS 10
2.7 Conclusion 10

3 Hector **11**
3.1 Introduction 11
3.2 Hector Work Flow 11
3.3 Ubuntu vs CentOS 12
3.4 Likelihood Method 13
3.5 Parallelization Efficiency 16
3.6 Python vs TcShell Scripts 17
3.7 MultiThreading in the AmmarGrag Method 19
3.8 Efficiency and Improvements 19
3.9 Conclusion 21

4 TSA Application **23**
4.1 Introduction 23
4.2 Software Engineering 23
4.2.1 System Architecture 23
4.2.2 Application Requirements 24
4.2.3 Use Cases 25
4.3 Software Design 29
4.3.1 System Work Flow 29
4.3.2 Front End Design 29
4.3.3 Back-End Design 30
4.4 Software Implementation 33
4.4.1 Web application 34
4.4.2 CLI 38
4.5 Conclusion 40

5 Implementation and Testing **41**

5.1 Introduction 41

5.2 Implementation 41

5.3 Results File 41

5.4 Testing 43

5.5 Conclusion 50

6 Conclusions and Future Work **51**

6.1 Conclusions 51

6.2 Future Work 51

Bibliography **53**

A Appendix **57**

A.1 JavaScript Use on Application 57

A.2 Apache Configuration File 59

A.3 DataBase Description 60

A.4 Upload Page 63

A.5 Server Application Code 68

A.6 Client Application Code 71

A.7 IQQE.North.result 75

A.8 ENU File Format 76

A.9 First 30 lines of file East.Pre.Mom 77

A.10 First 30 lines of file East.Out.Mom 77

List of Figures

2.1 Example of an unprocessed positional time series 6

2.2 Example of a Positional Time-Series of IQQE station 7

2.3 Example of various signals that a time series can contain 8

3.1 Figure explains the normal operation of Hector 12

3.2 Comparison between the AmmarGrag and FullCov for different percentages of data gaps using synthetic data 14

3.3 Comparison between the AmmarGrag and FullCov for different percentages of data gaps using real data 15

3.4 Comparison between the AmmarGrag and FullCov for different percentages of data gaps using 258 stations 16

3.5 Parallelization percentage improvement 17

3.6 Difference Time Between Python and Shell 18

3.7 Time of execution with and without Hector improvements by percentage of gaps 20

3.8 Execution gain in percentage 21

4.1 This figure describe the application architecture 24

4.2 Use Case Home Page 26

4.3 User Case Request Page 27

4.4 User Case Profile Page 28

4.5 The entire system operation, from entering a new request to returning the results to the user 29

4.6 Database schema 32

4.7 Sign Up Page 34

4.8 Login Page 35

4.9 About Page 35

4.10 Page that the user must use to place a new order 36

4.11 This page permit to user view all requests they made 37

4.12 This page permits the user to view this profile 38

5.1 Contents of the tar file with results 42

5.2 List of all files in the MOM folder 43

5.3 Shows the results obtained using the Selenium test to Signup and Sign In . 44

5.4 Filled out page of a new order for the IQQE station 45

5.5 Database entries for the request for the IQQE station 46

5.6 Checking of stored Sol file for the IQQE station 46

5.7 Request ID 49 results successfully created 47

5.8 Example of the File token.id contains the user key 47

5.9 Example of a Hector configuration file 48

5.10 Figure shows how to make a request and get the results by command line . 48

5.11 Figure shows the server side of receive a request and return results for the user 49

5.12 This error occurred when the user put a wrong parameter; in this case its an error in the Interpolate parameter 49

5.13 Error in missing of epochs for estimation of post-seismic deformation . . . 50

5.14 File with the time series does not have enough daily solutions to be processed in Hector 50

List of Tables

3.1 Ubuntu vs CentOS Times Comparison 13

3.2 Python vs Shell Times Comparison 18

3.3 Comparison between the number of Threads used by AmmarGrag and the average analysis time 19

3.4 Gain from Hector improvements 21

4.1 Description of Use Case Home Page 26

4.2 Description of Use Case Request Page 27

4.3 Description of Use Case Profile Page 28

4.4 Table to create the relationship between the folder path and its description 33

Acronyms

TSA TimeSeries Analysis by SEGAL

UBI Universidade da Beira Interior

SEGAL Space & Earth Geodetic Analysis Laboratory

SQL Structured Query Language

HTML HyperText Markup Language

PHP Hypertext Preprocessor

CSS Cascading Style Sheets

GNSS Global Navigation Satellite System

GUI Graphical User Interface

CORS Continuously Operating Reference Stations

CLI Command Line Interface

CATS Create and Analyse Time Series

API Application Programming Interface

SSE Slow Slip Events

PSD Post-Seismic Deformation

ARFIMA Auto Regressive Fractionally Integrated Moving Average

RHEL Red Hat Enterprise Linux

FIFO First In First Out

HTTPS Hyper Text Transfer Protocol Secure

ENU East North Up

MJD Modified Julian Date

MLE Maximum Likelihood Estimation

Chapter 1

Introduction

1.1 Problem Statement

Natural hazards such as volcanic eruptions, earthquakes, tsunamis, and landslides are among mankind's main concerns. Due to their complicated mechanisms, they are still impossible to predict. However, a streamlined way to study them is to monitor their primary sources of occurrence. High-resolution crustal deformation monitoring is a tool to study these natural phenomena and search for the root causes. Networks of GNSS Continuously Operating Reference Stations (CORS) have been used worldwide to monitor crustal deformation rates, with more than 1600 stations in Europe [1]. Optimally processing the ever-growing GNSS datasets requires developing state-of-the-art tools to interact with them.

Currently, there are several software packages for processing data obtained from a station to estimate the motion of a GNSS station. In this study, we will focus on Hector [2], a tool created in the Space & Earth Geodetic Analysis Laboratory (SEGAL) group and being used worldwide.

Analyzing the time-series and obtaining the best estimate possible has become imperative for a geoscientist. Despite all technological evolution there are still stations that can take up to one hour of processing time, so achieving the best possible performance in this software is imperative. This study includes the first part, which will study how to achieve high levels of performance in data processing in Hector.

This study aims to develop one application based on a GUI and CLI to process the positional time series from a station. The application intends to simplify the interaction with the Hector software, thus facilitating the station's velocity calculation. The project is quite captivating to be carried out because it improves knowledge of the development of full applications and creates solid communication between the computer and earth sciences.

1.2 Motivation

This project combines geophysics with information technology, making it a project with a direct impact on society and people's lives. Because of this, the motivation to carry out this project is easy to acquire and comes from the impact it can have on our lives.

1.3 Objectives

The objectives of this thesis are as follows:

1. The first objective of this work is to create a fully operational application that can use Hector to process the files introduced by users and allow a simple and intuitive interaction with the application. The application can be divided into two parts: a web application and a CLI application, which allows entering files for processing. Also, part of the study was to analyze Hector and its implementation for better software performance. In addition to this main objective, other objectives were also addressed.

    - Back to the main objective is to have a web application with a graphical interface that allows the user to create a profile and obtain help/information on how to use the GUI and CLI when making requests to Hector. This application must be easy and intuitive to use and allow the user to easily understand how both the web and command line applications work
    - Secondly, software was developed that allows users to place requests via the application command line, allowing them to make requests and receive responses as soon as the request is processed in Hector.
    - Both the GUI and CLI applications share the same databaseand processing on the back office. Another objective is to combine the two parts of the same application to work together.

2. Another objective is to analyze Hector and improve its performance. In particular, several aspects are studied:

    - The parallel processing of the three positional (East, North, and Vertical) components of a station.
    - The choice of the best likelihood method(AmmarGrag, FullCov) since the two can have quite large time differences.
    - The use of Python in the scripts running Hector.
    - The influence of the Operating System. In particular by comparing the CentOS and Ubuntu.

3. A further objective of this application is its integration with several applications that already exist in SEGAL so that the functionalities created here can be exploited in these applications and thus increase their importance in analyzing time series.

1.4 Contributions

This project was developed at SEGAL, Laboratory of the Department of Computer Sciences of Universidade da Beira Interior (UBI). This group focuses on developing IT solutions for Geoscience research

As Hector is a software used daily by the laboratory, its optimization and the possibility of reducing time in processing positional time series are fundamental contributions to the laboratory's daily objectives.

This study also aims to continue the interaction between geoscientists and computer scientists. The positional time series theme exemplifies how the computing community can create tools that help other scientists.

Chapter 2

Positional Time-Series: Concepts and Backgrounds

2.1 Introduction

This chapter aims to introduce the fundamental concepts used in this study, thus providing readers with the necessary knowledge to understand the entire study.

2.2 CORS (Continuously Operating Reference Stations)

The community distinguishes between CORS stations and those installed on a shorter timescale, called Campaign stations,(for instance, to monitor a sudden event such as a volcanic eruption).

GNSS is a technique that allows the determination of the position of points on the Earth's surface. GNSS equipment can be used to estimate the movements due to plate tectonics or other geophysical or anthropogenic causes.[3]

GNSS satellites transmit coded signals at exact times. The receiver of each station decodes and processes the signals emitted by the satellites. The receiver is responsible for estimating the time it takes the signal from the GNSS satellite to reach the antenna of the GNSS station. This calculation makes it possible to arrive at the distance from the satellite to the CORS station [4].

Through the messages sent by the GNSS satellites, it is possible to calculate the position of the station's antenna. The transmission times of signals from at least four GNSS satellites must be measured.

2.3 Solution Files

A station creates a daily file of observations with all its daily observations. These observations can be obtained up to 100Hz but normally are stored every 30 seconds.

By processing the file, it is possible to obtain the station's position on that day. Using the combination of several days, a Solution-type file can be obtained with the position of a station over several days/months/years. These files contain the time series with only

the estimated position points of the station's position (cf. Figure (2.1), through which a program such as Hector will attempt to estimate components such as the station's velocity.

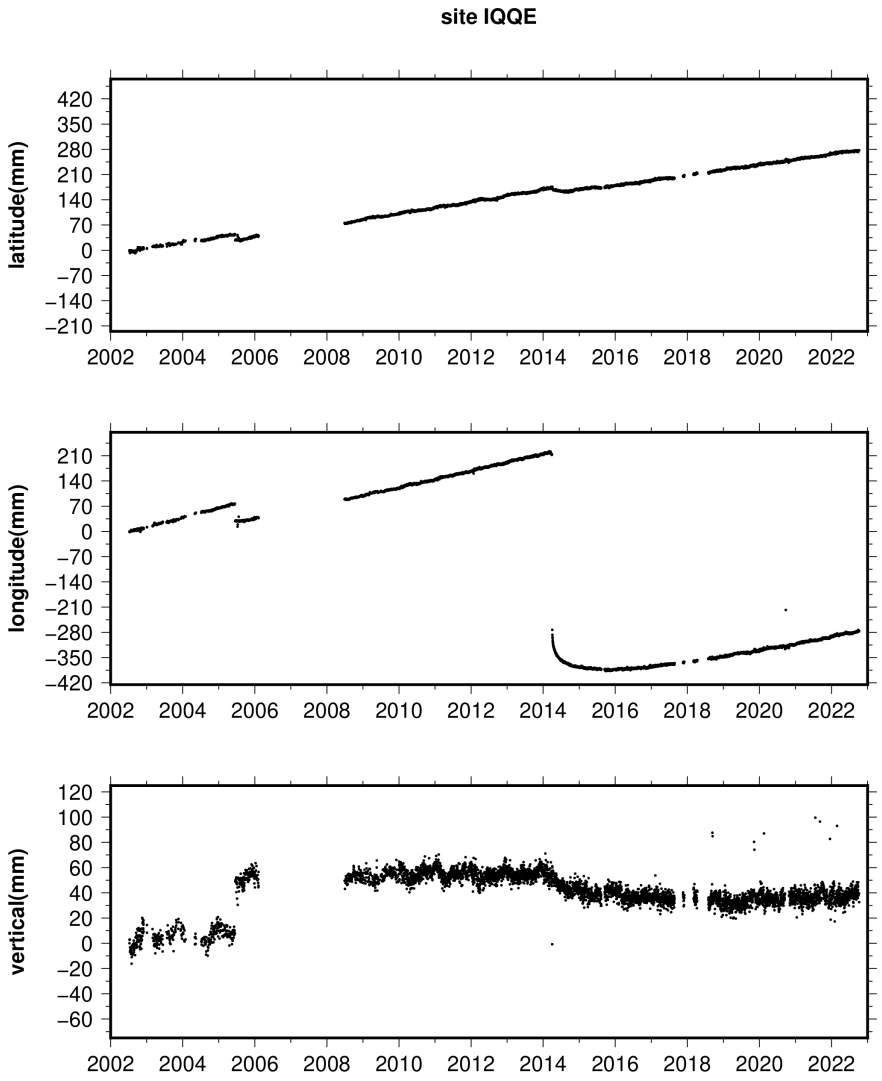

Figure 2.1: Example of an unprocessed positional time series

2.4 Positional Time-Series

A position of a GNSS station in the three coordinate components (Up, Latitude, and Longitude) are recorded as a point of a positional time series.

An example of a positional time series is shown in Figure (2.2). For the different components (latitude or north, longitude or east, and up or vertical), shown in separate panels, the black dots show the estimated positions and the green dots show the trajectory model calculated by Hector. The estimated linear (or secular) trend is also shown in the upper right corner of each component.

Both the fitted trajectory model [5] and the estimated station velocity shown in figure are

parameters calculated by Hector.

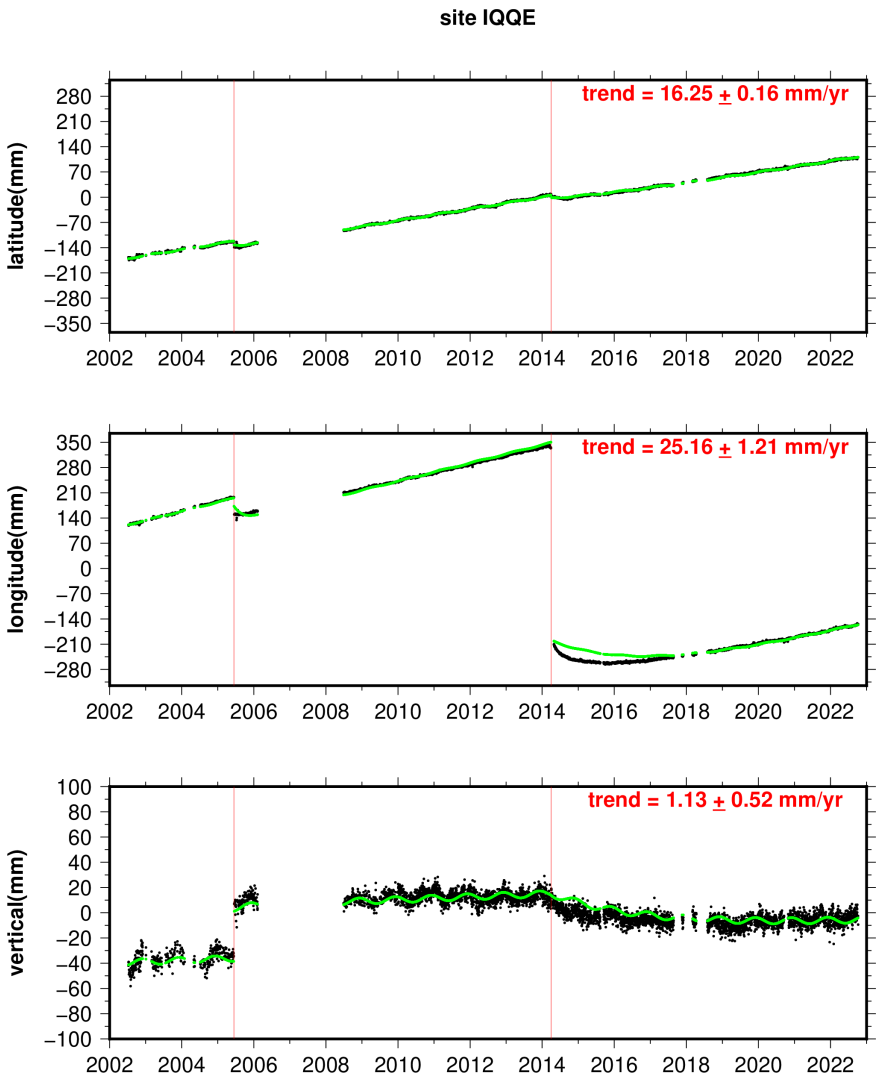

Figure 2.2: Example of a Positional Time-Series of IQQE station

2.5 Time-Series Analysis

A time series of positions can contain different signals:

- Secular motions due to plate tectonics;
- Seasonal signals (due to loadings);
- Offsets due to equipment change or large earthquakes (co-seismic);
- Decay phase (post-seismic) deformation;

In figure (2.3) adapted from [6], we have an example of the various signals that may exist.

Due to these different signals, the analysis that Hector carries out on the time series is essential, allowing the station velocity to be calculated with the greatest possible precision.

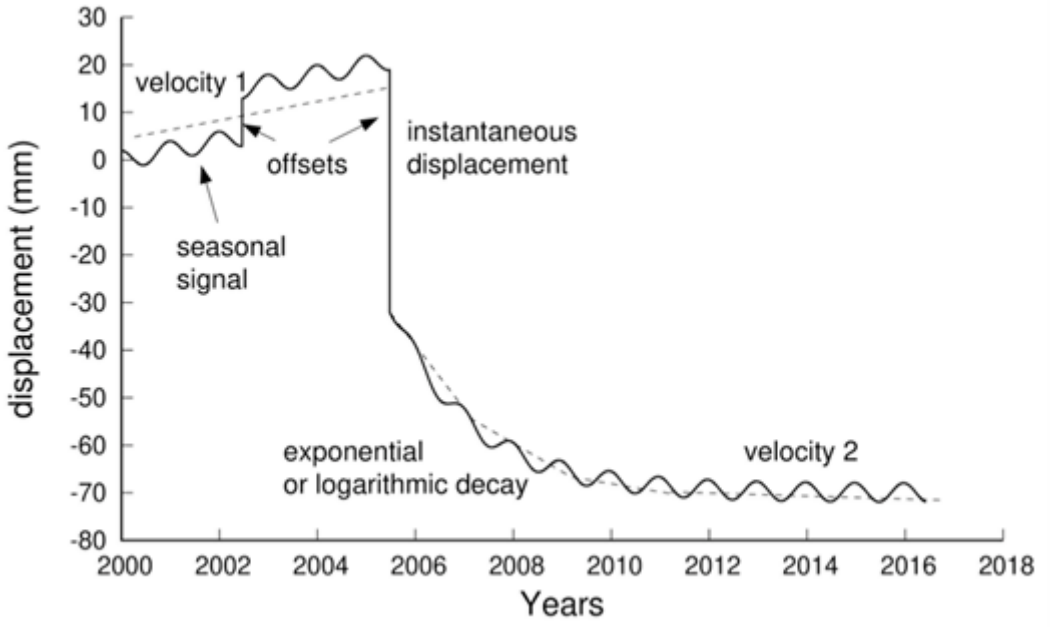

Figure 2.3: Example of various signals that a time series can contain

2.6 Time-Series Analysis Software

In this section, Hector is presented and compared with some software similar to it.

2.6.1 Hector

Hector [2] is an open-source software package that has been popular among the scientific community to quantitatively estimate the different geophysical processes in the time series discussed in the previous section (2.4).

Hector can estimate the linear trend in a time series with temporally correlated noise. If there is noise in these time series, it dramatically affects the accuracy with which the linear trend can be estimated.

In addition, Hector can also help to detect outliers (since automatic detection is not a 100% reliable procedure) and estimate the noise associated with the time series (e.g. caused by the GNSS receiver installation (monument) instability. Hector permits the evaluation of different parameters and models to estimate the variables of interest (linear trend, seasonal signals, noise models).

Hector's main features are: [2]

- Correctly handles missing data [7]. No filling in of the missing data is required, nor is an approximation of the covariance matrix required;
- It is possible to include annual, semi-annual or different period signals in the estimation of the linear trend;
- It has the option to estimate different displacements in different periods;
- It is possible to choose any of the following Noise Models: Power-law noise, Auto Regressive Fractionally Integrated Moving Average (ARFIMA), generalized Gauss-Markov and white noise models;
- It has the functionality to remove outliers and to make power spectral density plots;
- It has the functionality to detect offsets automatically [8];
- It can effectively handle Post-Seismic Deformation (PSD) Moments and Slow Slip Events;

2.6.2 CATS

Create and Analyse Time Series (CATS) is software that allows one to estimate the parameters of a time series [9].

CATS uses Maximum Likelihood Estimation to perform multi-parameter estimation for the time series. The program estimates the parameters in a linear mode divided into two parts. The first part, linear, includes offsets and periodic signals, while the second is used for specific noise models.

This program includes the possibility of some different noise models, also available in Hector. However, it does not include some options available in Hector, such as Post Seismic Decay, Multi-Trend or Slow Slip Events.

2.6.3 LANGBEIN

The LANGBEIN [10] method is used to calculate the velocity from the positional time series. In addition to being applied to GNSS time series, it can also be applied to other types of time series with linear data.

This method consists of making linear adjustments to the time series data to find movement patterns throughout the data. It uses a linear function that adjusts its parameters to best suit the provided data. It can also estimate the uncertainty associated with the velocity calculation.

This also contains features for dealing with dataless time spaces, correctly dealing with linear time series. The problem is dealing with time series with non-linear influences, which can negatively affect the calculations which do not affect calculations in Hector.

2.6.4 MIDAS

MIDAS is a software to perform the estimation of secular velocities [11]. MIDAS can also deal with outliers, gaps and seasonal variations when calculating velocities. One of the biggest problems with time series is seasonal variations. MIDAS focuses heavily on this aspect and ensures they have minimal effects on the secular velocity calculation.

Other existing programs for calculating station velocities are based on two steps: calculating the estimated value using least minimum squares and removing values that could be considered outliers. MIDAS has a different approach to this, based on the Theil-Sen method. This method works by calculating the median trend between all possible pairs of values.

Despite this, MIDAS has some limitations when dealing with situations of seismic deformations or different velocities in the same time series components that Hector can deal with.

2.7 Conclusion

This chapter covers some fundamental concepts and popular software within the scope of this study. This allows the reader to understand the study better and have a view of the fundamental tools.

Chapter 3

Hector

3.1 Introduction

This chapter explains how Hector works and studies its performance. The results of the studies carried out indicate possible changes to optimize Hector.

3.2 Hector Work Flow

Hector, in its normal execution, is executed by auxiliary scripts responsible for executing all Hector programs that allow its normal execution. Analyzing the main script that runs Hector allowed us to understand how it works and understand all the necessary actions that Hector takes to return the results it expects to users.

Firstly, Hector reads all the data users enter, the file with the time series and the configuration data for the time series analysis. This data is used to create Hector configuration files, and the time series file is converted to East North Up (ENU) format. Once Hector has these files, the script to remove the Outliers is executed; the output of this script creates three files, each containing a component (East, North and Up). Files to deal with PSD, Multi Trend or offset situations are created, and the essential part of Hector, the estimated trend, is executed. Individual results for each time series component are obtained and combined to get the time series figure. This explanation is presented in figure (3.1).

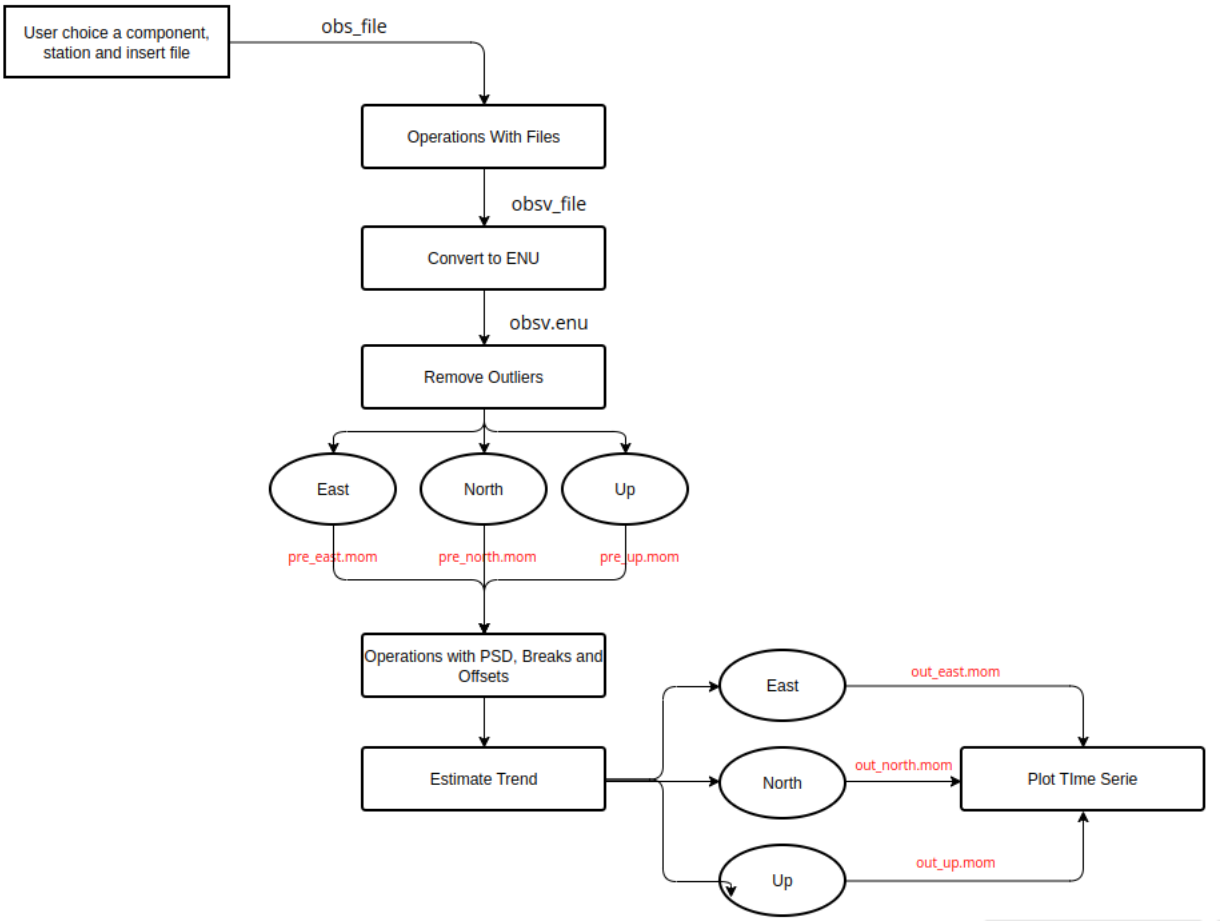

Figure 3.1: Figure explains the normal operation of Hector

3.3 Ubuntu vs CentOS

Hector is a software package that can run on operating systems such as Ubuntu and CentOS. This test aims to evaluate if different operating systems can affect Hector performance.

Ubuntu [12] and CentOS [13] are two Linux-based operating system distributions.

Ubuntu is a Linux distribution based on Debian that is very popular among the computer community. It is recognized for releasing its versions with five-year support that offers stability and security, and it also has extensive documentation for the entire system.

CentOS is a Linux distribution derived from the Red Hat Enterprise Linux (RHEL) source code; it is mainly used in server environments and data centres due to its similarity with RHEL, a very close alternative.

A significant difference between the two is that they use different package management systems, which significantly impacts the packages available for each system, with Ubuntu being updated more often with recent package versions.

Based on the objective of this test, two virtual machines were created with the exact same specifications and just different operating systems. One machine has the Ubuntu operating system, and the other has the CentOS operating system.

Hector was installed similarly on both machines, and one hundred stations were analyzed three times on each machine. The average of the three tests was calculated for each machine. The average of the one hundred stations was then calculated for each operating system. The results are presented in Table (3.1).

Operative System	Times
CentOS	2m28s
Ubuntu	0m52s

Table 3.1: Ubuntu vs CentOS Times Comparison

Ubuntu is clearly better than CentOS. It can be justified in this case because Ubuntu receives package updates, while CentOS prefers to use more stable and secure versions. Hector depends heavily on the C++ [14] libraries used to run it. In Ubuntu, the use of more recent and optimized libraries may have a high impact on Hector processing time.

Therefore, Ubuntu will be the operating system used on the TSA machine and used in all subsequent tests.

3.4 Likelihood Method

This test will focus on choosing between two fundamental methods in Hector processing that strongly influence Hector analysis time.

The likelihood function is a probability measure that allows the determination of a set of parameters that best describe the observed data. This method is fundamental in statistics, allowing mathematicians to estimate specific parameters, make predictions and test hypotheses based on data [15].

Maximum Likelihood Estimation (MLE) computation is a crucial step in fitting stochastic models to GNSS time series data. Methods such as AmmarGrag and Fullcov [7] determine the best estimates for model parameters by evaluating the likelihood of the observed data under different parameter configurations.

In Hector case, the decision to choose between one method or the other is based exclusively on a mathematical parameter of the station's gap percentage. In Hector manual [16], it is indicated that the reference value is 50%. With a gap percentage value higher than 50%, the FullCov method is used, and with a gap percentage value lower than 50%, the AmmarGrag method is selected.

It was decided in this study to investigate further whether this value meets the processing

needs required for maximum Hector optimization, namely, which is the optimal threshold value to select one of the two methods.

The first test carried out within the scope of choosing the method was carried out using a synthetic time series in which the time series was carried out synthetically without data gaps, and then data gaps were introduced to reach specific percentages.

A test was carried out using only the AmmarGrag method and then a test using only Full-Cov; Each test was repeated three times for each method, and then the average of the three tests was taken.

The times of the three tests were averaged to obtain the results shown in Figure (3.2). The linear regression was also calculated for each method. It is seen that the intersection point between the methods is a percentage of gaps equal to 30.75. Before 30.75, the fastest is the AmmarGrag method, and after that value is the FullCov method.

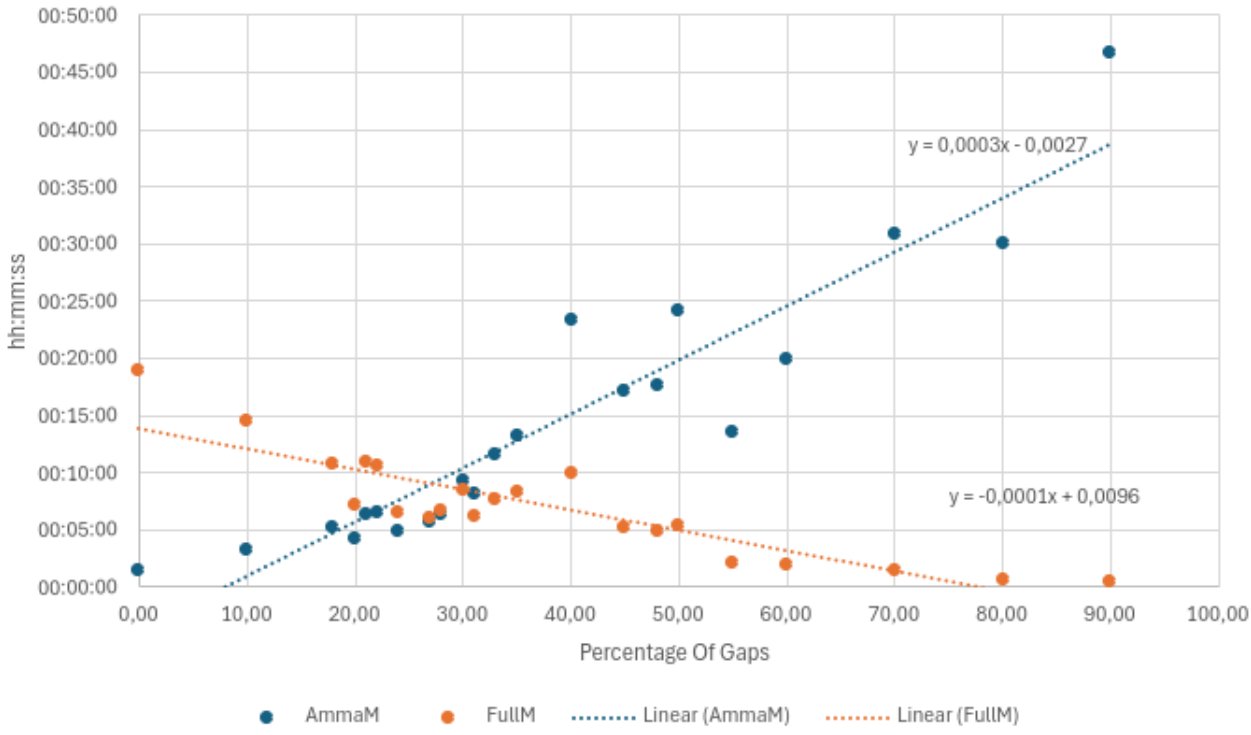

Figure 3.2: Comparison between the AmmarGrag and FullCov for different percentages of data gaps using synthetic data

The second test is carried out with one hundred stations with different percentages of gaps, varying between 0% and 100%. The tests were carried out in the same way as the previous ones. That is, three tests were carried out for each station and for each method. Figure (3.3) shows the correlation between the execution time and the percentage of data gaps for all stations using both methods.

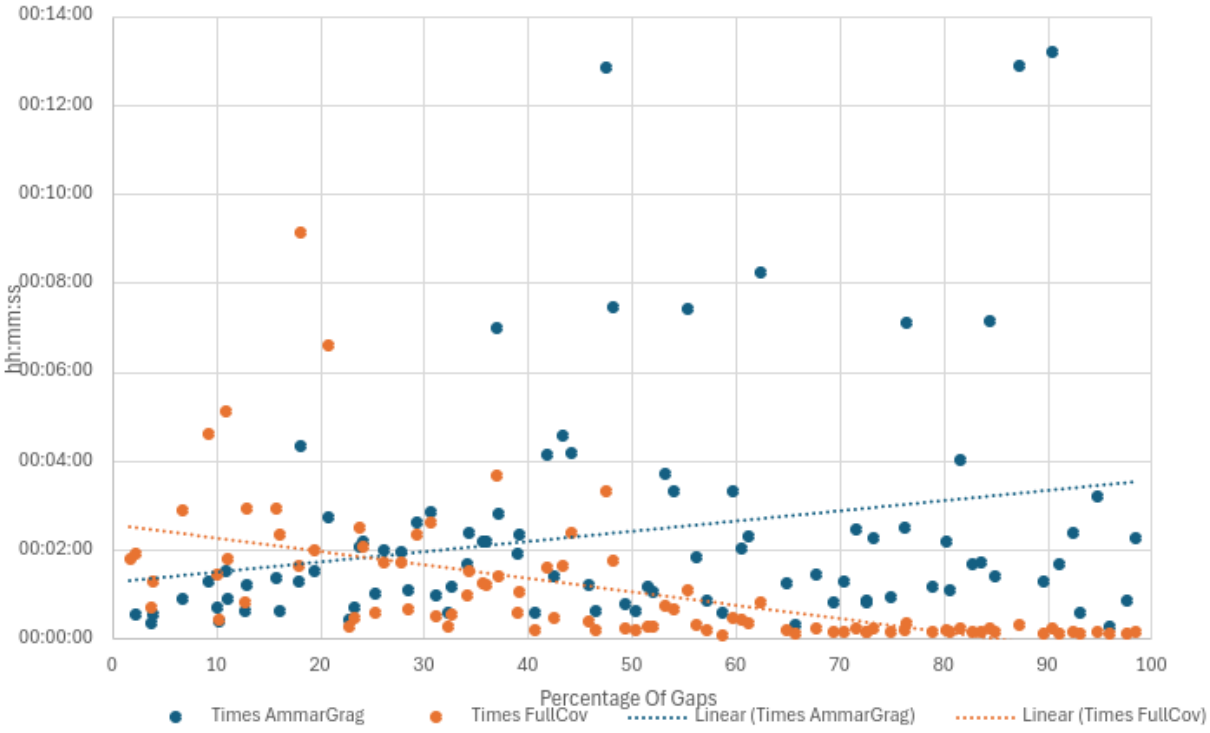

Figure 3.3: Comparison between the AmmarGrag and FullCov for different percentages of data gaps using real data

The regression linear for each method was estimated to calculate the intersection point, or as in the case of this study, the decision point at which a method or another should be selected. In this case, the intersection point has the value of x=31.33; before this value, the AmmarGrag method must be used for the calculations, and after this value, the FullCov method is preferable.

To complete this study and ensure the results, it was decided to add more stations to the tests, carrying out a test with 258 stations. The stations were selected to have data spans between three and twenty years. These stations also have different percentages of gaps between zero and one hundred.

The same methodology was used, repeating the test for each station three times for each likelihood method. Figure (3.4) shows the results obtained.

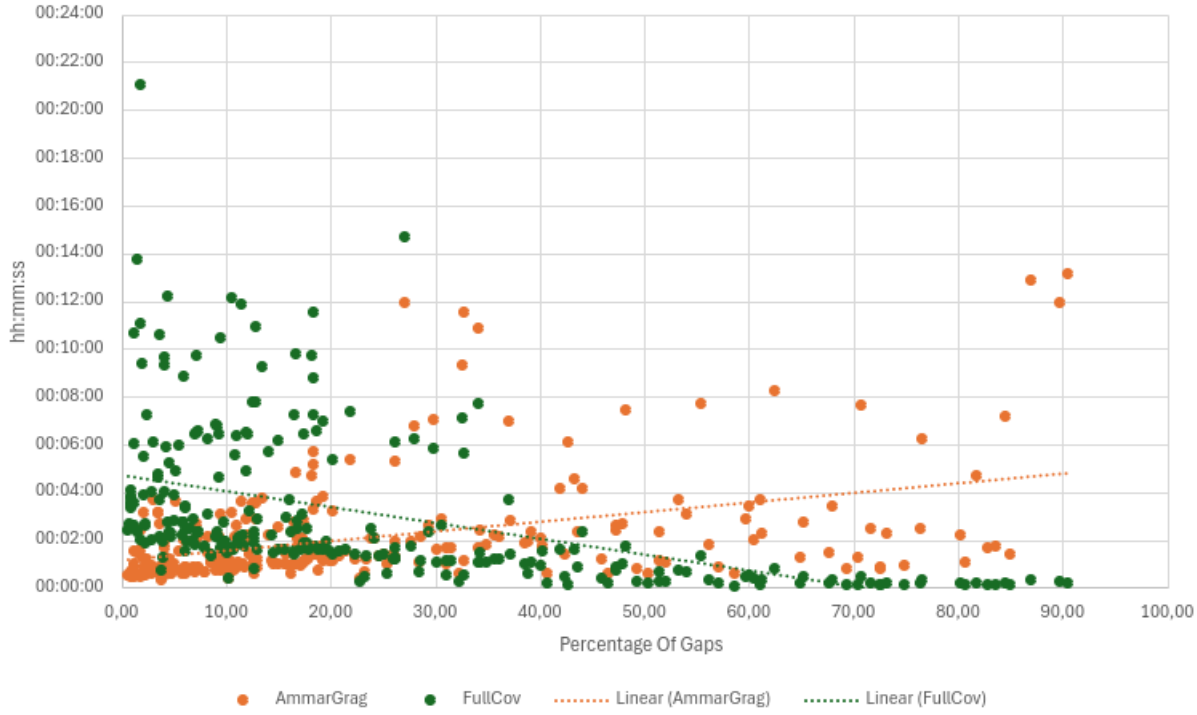

Figure 3.4: Comparison between the AmmarGrag and FullCov for different percentages of data gaps using 258 stations

In this case, the intersection point corresponds to a data gap of x=33.25.

According to all these studies, it is concluded that the optimal value of selection between the two methods is close to 30% of the percentage of gaps in the time series. To obtain a final value, we use the average of the three values, corresponding to a value of 31.7. Therefore, with this part of the study, it is possible to conclude that the value of 50% presented in Hector's manual is incorrect. Therefore, the AmmarGrag method must be selected when the data gaps are smaller than 32%, and the FullCov method must be selected when the time series has data gaps higher than 32%.

3.5 Parallelization Efficiency

As mentioned previously in Section (2.4), the analysis of each station processing consists of calculating the velocity (trend) for three different and independent components (latitude/north; longitude/east; vertical).

One way to speed up the processing time is to analyze the three components in parallel instead of analyzing the three components sequentially. To verify whether this theory is functional, tests were carried out without parallel processing, and the parallel processing of the three components was implemented.

Three tests were carried out with one hundred stations using the two approaches. After obtaining the results, the averages of the three tests were calculated, and the difference between the times without parallel processing and using parallel processing was calculated. Figure (3.5) shows that there was an improvement in the order of 0% to 40% for the majority of the stations.

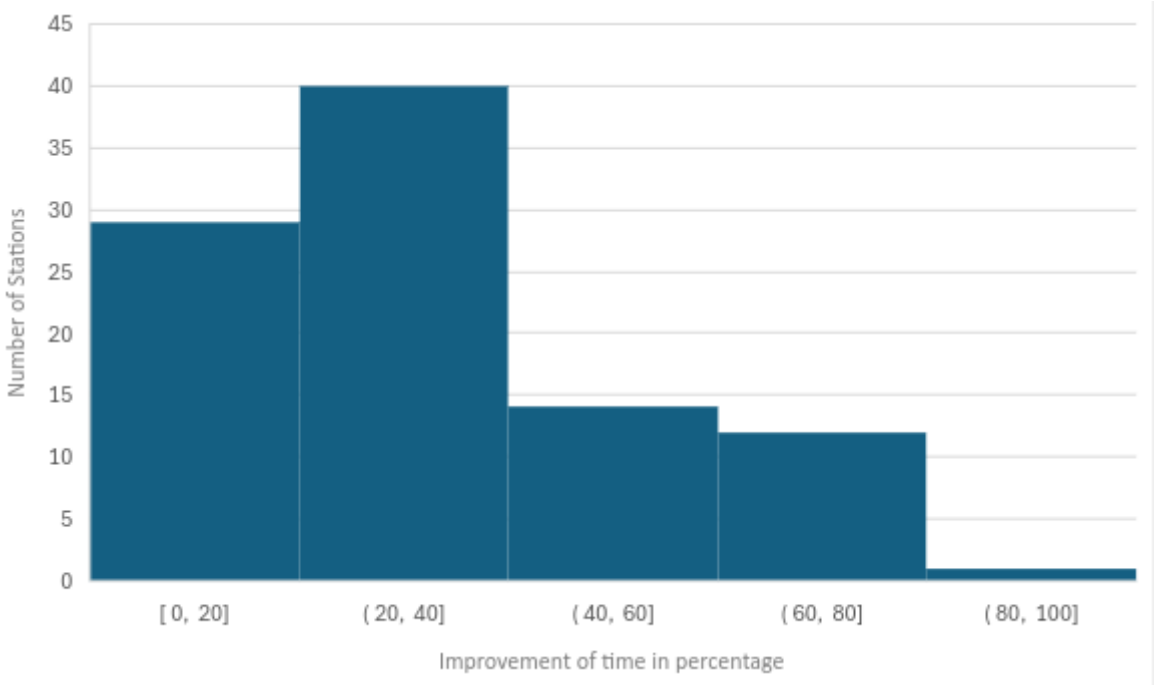

Figure 3.5: Parallelization percentage improvement

In addition to the previous results, the average improvement was also calculated for the 100 stations, with an average improvement of 9 seconds or about 19%. The ideal results predicted an improvement of around 60% of time improvement, since the analysis of the three components is now carried out at the same time, but due to Hector already carrying out parallel processing in some mathematical calculations that he performs, this result is not reached.

However, the results of this test demonstrate that pairing is efficient and the way to optimize Hector.

3.6 Python vs TcShell Scripts

To execute Hector, it is necessary to create several configuration files and prepare the input files. This preparation can be carried out with the help of scripts using TcShell [17] (benefiting of existing scripts already in use at SEGAL) or a different version of Hector using Python scripts. We also compared the efficiency of both scripting languages.

Three tests were carried out using the auxiliary scripts in TcShell and three tests using the scripts in Python. After that, the average of the three tests and the average times for one hundred stations were calculated.

The average times were as in Table (3.2).

Type of Auxiliary Script	Times
TcShell	1m15s
Python	1m16s

Table 3.2: Python vs Shell Times Comparison

As the average times of the one hundred stations were quite similar, the absolute difference between the times of each for one hundred stations was also calculated, as shown in Figure (3.6).

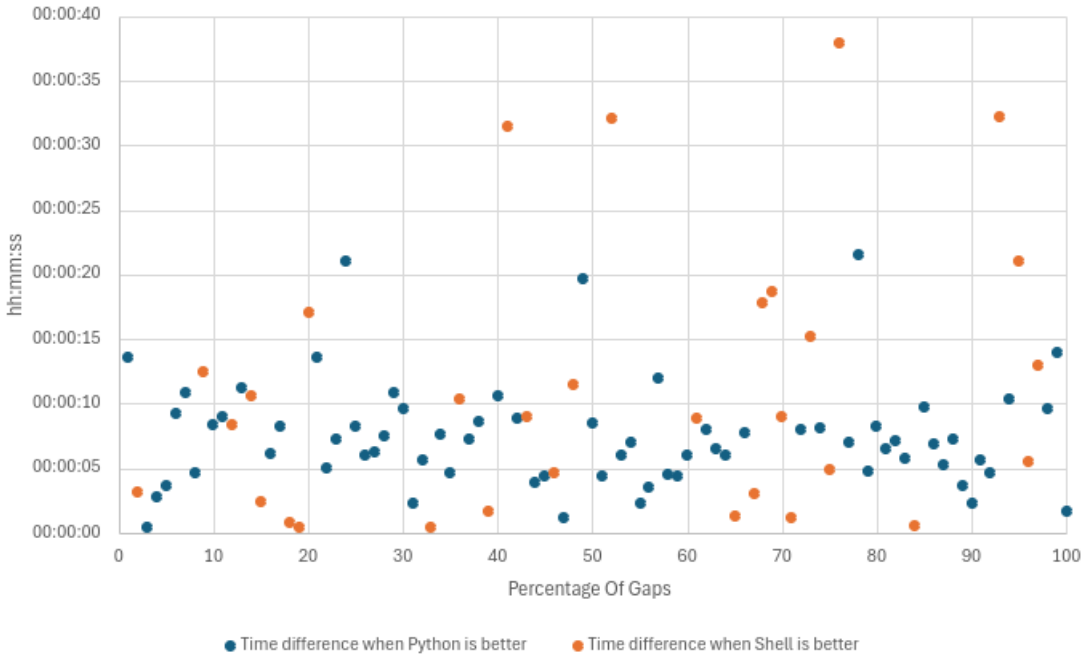

Figure 3.6: Difference Time Between Python and Shell

When analyzing the graph, we found that most stations absolute difference between times is less than 15 seconds. Furthermore, it is possible to check the stations in blue where Python scripts are better and in orange where TcShell scripts are faster; it is possible to verify that there are similar amounts of points in one colour and another.

With this difference in times observed in the graph and the average results calculated previously, it is concluded that the change to using Python scripts is not rewarding enough. TcShell scripts will continue to be used, as they are more adaptable and allow for more functionality when running Hector.

3.7 MultiThreading in the AmmarGrag Method

After analysing Hector code, it was possible to identify that in the AmmarGrag method, the maximum number of Threads is limited to eight, limitation that was not found in the FullCov method code. Because most current machines have a Threads value significantly higher than eight, we also decided in this study to verify if this limitation should be used in current machines with a Threads number greater than eight.

The machine used for this test has a total of 42 Threads, so it was decided to carry out this study in different ways. First, the number of threads used was limited to 4, 8, 12, 16, 20. Ultimately, it was decided also to carry out a study to eliminate this limitation and let Hector decide how to manage the threads of the machine to be used, using the OpenMP [18] library to manage the use of Threads.

The times presented in Table (3.3) were obtained by analyzing the times of 96 stations for each case and then calculating the average of these stations to obtain the values.

Number of Threads	Time
4 Threads	02m46s
8 Threads	03m43s
12 Threads	02m55s
16 Threads	03m48s
20 Threads	02m34s
Threads limited to machine number	02m55s

Table 3.3: Comparison between the number of Threads used by AmmarGrag and the average analysis time

The results obtained do not allow us to conclude if there is a recommended limit for the number of Threads when executing Hector. In fact, the values vary largely and there is no correlation between the number of threads and time of execution. These results permit to conclude that the limitation is meaningless. The maximum number of threads used by Hector will only be limited by the maximum number of threads on the machine.

3.8 Efficiency and Improvements

To verify the effective improvement in performance, ninety-four stations were analyzed, performing each test three times in the same machine by comparing the time of execution without any improvement and implementing the following conditions:

- Parameter for choosing the likelihood method based on the percentage of gaps between AmmarGrag and FullCov, changed to 32%.
- Parallel execution of the three components of a station.

- Ubuntu Operating System.

- Use of the AmmarGrag method without Thread limit.

Figures (3.7) and (3.8) compare the results distributed by the percentage of gaps and by percentage of improvement, respectively.

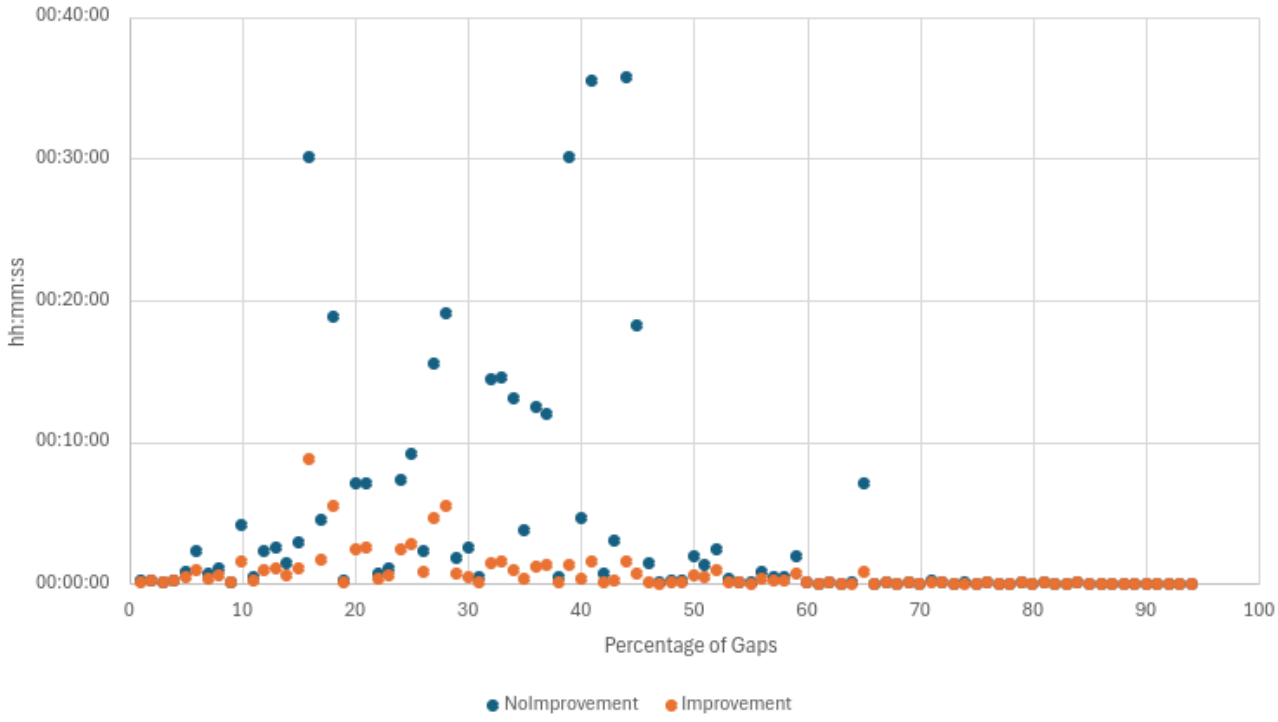

Figure 3.7: Time of execution with and without Hector improvements by percentage of gaps

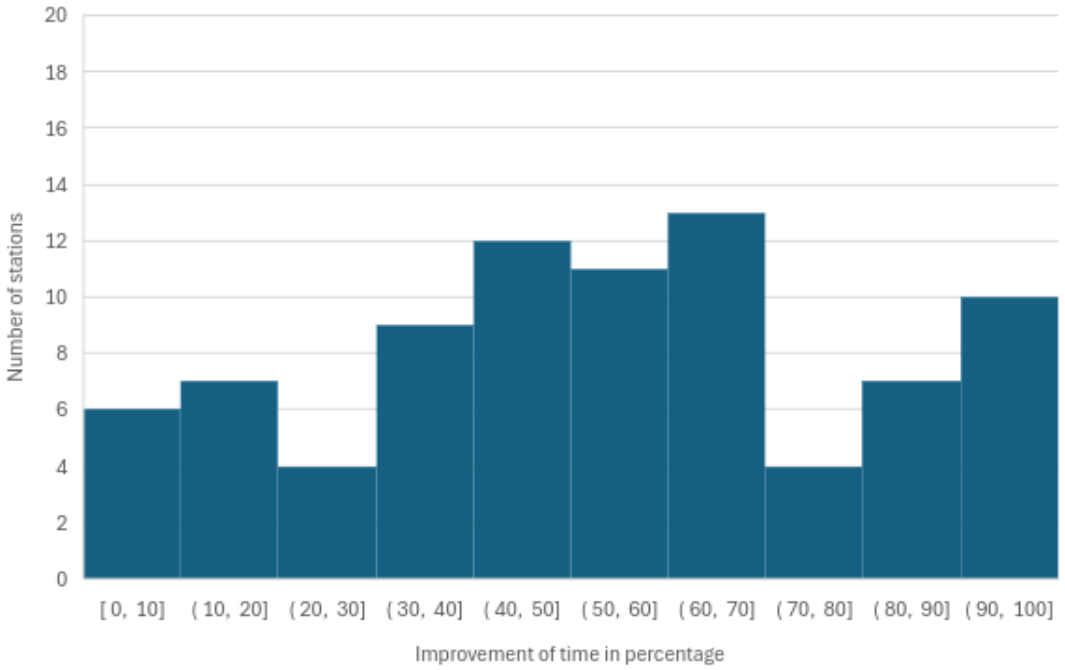

Figure 3.8: Execution gain in percentage

It can be seen that the implemented improvements decrease significantly the time of execution; this is also confirmed by the averages of the times presented in Table (3.4).

Version Hector	Times
No Improvement	03m55s
With improvements	00m43s
Difference	235s

Table 3.4: Gain from Hector improvements

For 54% of stations the improvement is higher than 50% as shown in Figure (3.8). Of the ninety-four stations used in this study, only twenty-eight have an improvement of less than 30

3.9 Conclusion

This chapter describes the analysis of Hector and the tests carried out to improve its performance. Different solutions were proposed to improve Hector functioning in the analysis of the time-series of stations. It was shown that some of the improvements have a significant impact on Hector performance.

Hector was installed on the machine where we go test the application with all the improvements analyzed, covered in the next chapter.

Chapter 4

TSA Application

4.1 Introduction

This chapter describes and analyzes the entire TSA application, its design, and the experience that the user should have when using it.

4.2 Software Engineering

This section presents the software engineering carried out for the application. This section aims to contextualize the user about the application and its objectives.

4.2.1 System Architecture

Figure (4.1) contains the system architecture of the GUI application, through which it is possible to understand how the application works and the pages that users have access to, whether logged in or not.

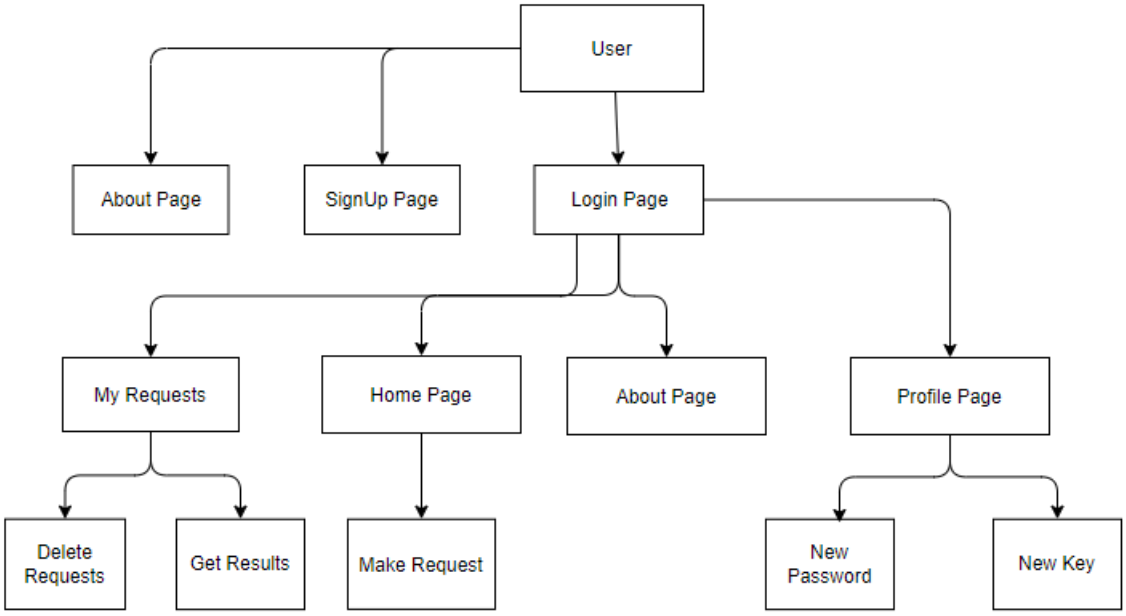

Figure 4.1: This figure describe the application architecture

4.2.2 Application Requirements

We present in this section the application requirements, which are divided into functional and non-functional requirements.

4.2.2.1 Functional Requirements

The application functional requirements implemented are:

- Allow user registration.
- Require users to log in to use the application.
- Provide access to the login and registration pages without requiring a login.
- Allow the user to enter a new time series to be analyzed in Hector by creating a new request.
- Must contain a page that lists all user's requests, allowing them to view the requests, delete requests or obtain requests results.
- Allow the user to retrieve the request results through the web application.
- Allow command line requests using a configuration and user key files.

- Obtains the results and status of requests via the command line.
- Provide users to consult the About page, even if they are not logged in.
- The About page should explain the application and how it works.
- Needs to be interchangeable, making it possible to request via the command line and obtain the result in the web application or vice versa.

4.2.2.2 Non-Functional Requirements

The non-functional requirements are the following:

- The system must load the application promptly.
- The application must work in all browsers.
- The application must securely store the user's credentials in the database.
- The application must be available in English.
- The system must be available using the Hyper Text Transfer Protocol Secure (HTTPS) protocol.
- The system must be available at least 95% of the day, every day of the year, informing the user if the page is temporarily unavailable.

4.2.3 Use Cases

This section presents three possible use cases in GUI, through which it is possible to understand the correct use that a user can have with the application.

Figure (4.2) and Table (4.1) describes the first use case: the interaction of the user with the main page. Login is mandatory to access the application. On the main page, it is possible to place requests to be processed by Hector. The request is saved in the database, and as soon as it is saved, it automatically enters the queue for processing.

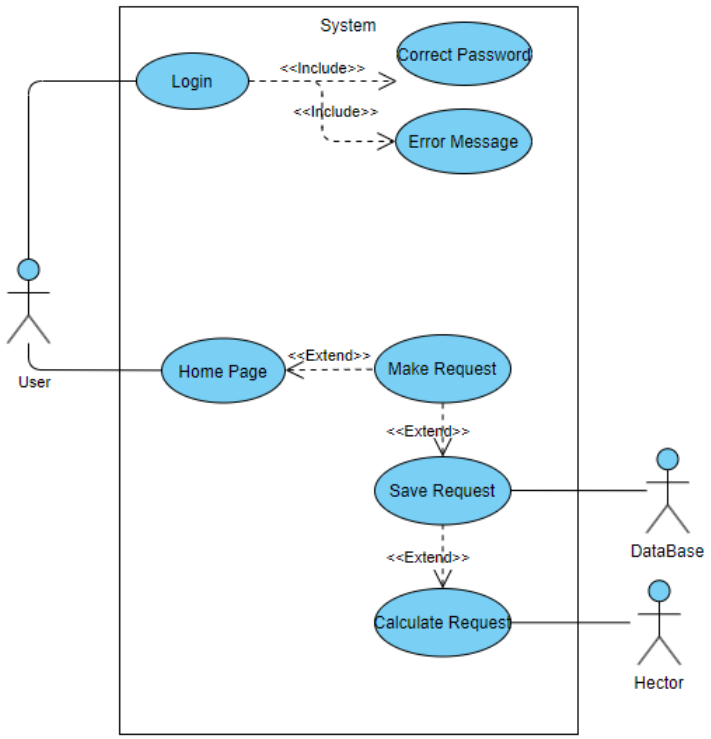

Figure 4.2: Use Case Home Page

Actors	Users, Database, Hector
Description	This diagram represents the use case in which the user accesses the application's main page and makes a new request
Data	Positional Time Series file, Offsets, PSD Period, Breaks, Calculation methods, etc.
Stimulus	This use case is triggered when the user accesses the application main page and makes a request
Response	The response will be indicated by a boolean that demonstrates the correct execution of saving the request in the database and triggers the process so that the request enters the queue to be processed.
Comments	This use case is optional and only happens when the user sends a request

Table 4.1: Description of Use Case Home Page

The use case in Figure (4.3) and Table (4.2) is from the application Requests page. To access it, the user needs to log in to the application. It is possible to view all requests made by the user, delete a request or obtain the results if the request has already been processed.

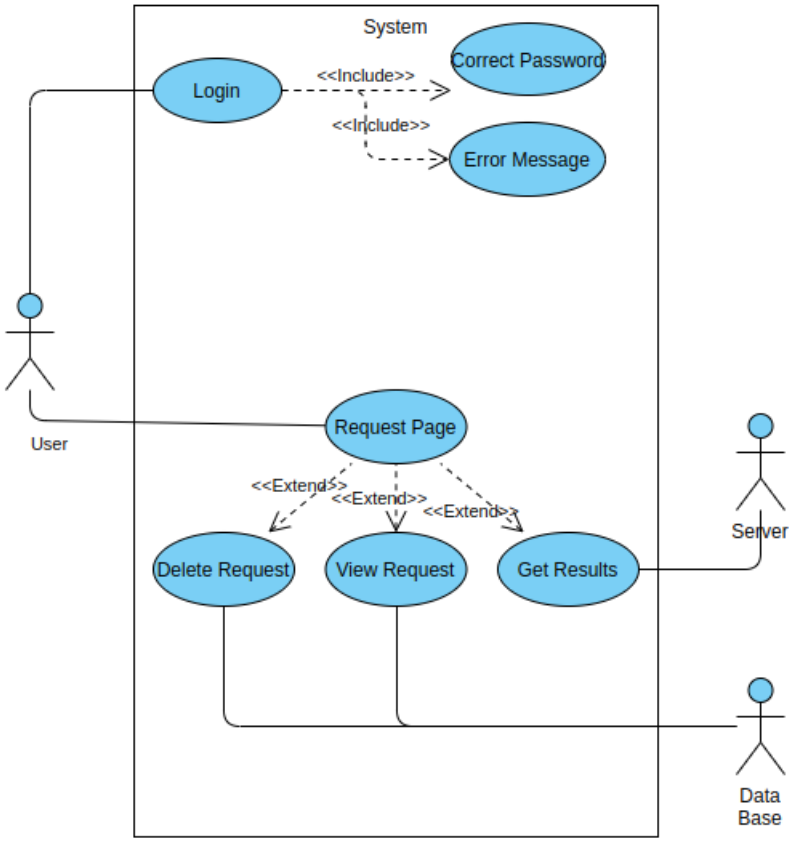

Figure 4.3: User Case Request Page

Actors	Users, Database, Server
Description	This diagram represents the use case in which the user accesses the application request page
Data	ID Request, State of Requests, Results.
Stimulus	This use case is triggered when the user opens the application's Requests page
Response	This use case will result in the correct visualization of all orders placed by the user and the processing status of each order placed.
Comments	This use case is optional and only happens when the user interacts with all requests made by him in the application

Table 4.2: Description of Use Case Request Page

Finally, the use case in Figure (4.4) and Table (4.3) is from the Application Profile page. To access it, the user needs to log in to the application. Viewing all the user data in question, changing the password or generating a new key is possible.

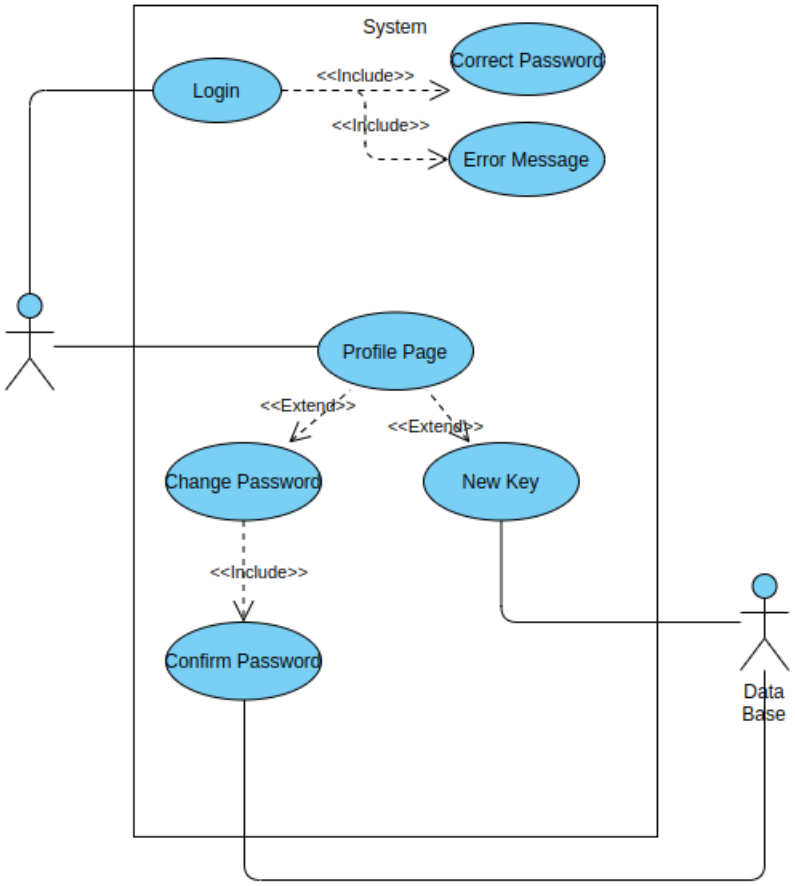

Figure 4.4: User Case Profile Page

Actors	Users, Database, Server
Description	This diagram represents the use case where the user interacts with their profile page
Data	ID User, Password, Key, Email
Stimulus	This use case is triggered when the user accesses their profile page
Response	This use case will result in the correct display of all user profile information, it is also possible to change the user's password and generate a new random user key.
Comments	This use case is optional and only happens when the user interacts with their profile page

Table 4.3: Description of Use Case Profile Page

4.3 Software Design

This section covers the different parts of the application that together form the back-end and front-end of the application. These parts are explained and exemplified so that it is possible to assemble them and understand how they form the single TSA application.

4.3.1 System Work Flow

This chapter presents the application design Figure (4.5). It represents everything that happens in the application when a new request is made, whether the request is made by the GUI or by the CLI. The first step involves reading the various order parameters and saving them in the database. After a request is chosen for processing, it is processed in Hector, and the results are made available to the user.

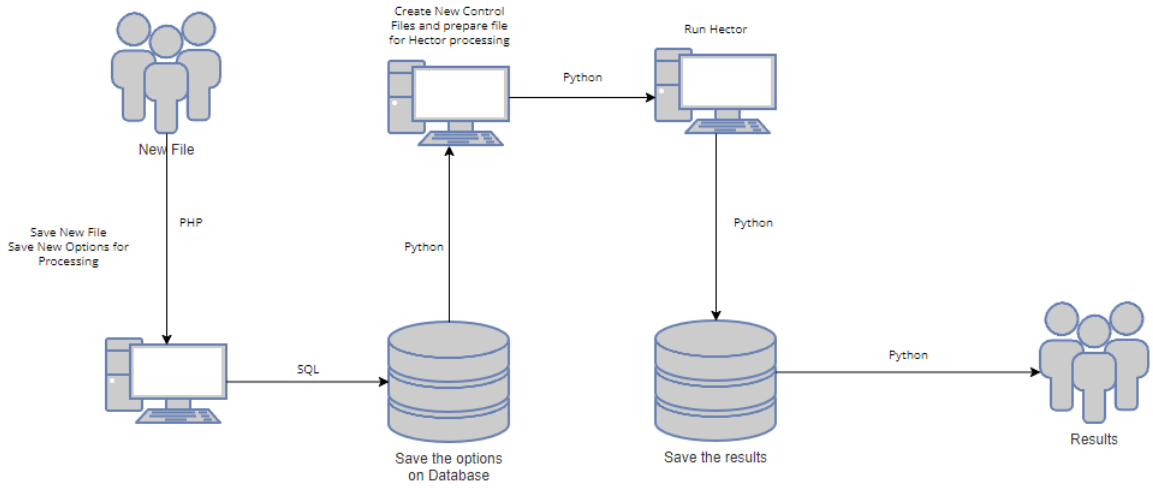

Figure 4.5: The entire system operation, from entering a new request to returning the results to the user

4.3.2 Front End Design

In this section of the study, the different tools used to develop the application in terms of front-end design are discussed.

4.3.2.1 HTML and CSS

HTML [19] and Cascading Style Sheets (CSS) [20] are fundamental for building a current web page. HTML provides all the structure and order of elements on the web page, while CSS provides all the styles of the pages, making them visually appealing. Using the two together allows one to create interactive and visually appealing web pages.

The entire graphic part of the application is carried out using HTML, CSS. In addition to the developed CSS, the Bootstrap [21] library is also used, which contains some predefined CSS elements.

4.3.2.2 JavaScript

JavaScript [22] is a high-level programming language for web development, essential for creating modern web pages. Unlike HTML and CSS, which provide structure and visuals, JavaScript creates dynamism in the user experience. It is used to validate forms, create page animations or dynamically manipulate elements; the JavaScript code used in the application can be found in Appendix (A.1).

4.3.2.3 PHP

PHP [23] is also a language used for dynamic web development. It has easy integration with HTML, which makes it a great choice for creating pages that need to interact with databases, such as online stores, or, as it is in this case, for an application that needs to interact directly with the database.

4.3.2.4 Apache

Apache [24] is an open-source web server used to host websites and web applications. It supports several technologies, such as PHP, Python, and Perl, and it also supports implementing additional modules for more functionalities. It is also highly configurable, which allows it to be optimized depending on the application it needs to run.

The web application and the flask server (4.3.3.1) runs on Apache, being available through any browser. The Apache Configuration File listed in Appendix is (A.2) having both applications configured.

4.3.3 Back-End Design

This part explains the main components that interact with each other to enable the orderly processing of orders in Hector.

4.3.3.1 Flask Server

Flask [25] is a microstructure written in Python and used to create microweb services. Flask has the possibility of creating a server that is available to respond to requests made by clients.

Python [26] is a high-level language known for its simplicity and readability, supporting different programming paradigms. It has a large library and a very active community, which makes project development easier, and it also has many popular libraries such as Django, Flask, NumPy.

In this application, we use Flask and Python to create the TSA server. This server accepts requests made via CLI, allows checking the status of those requests, and allows to return the results of the requests. In addition, the Flask server is also used to return results from the web application.

To use any function of the application, the user needs to have a key. This unique key must be obtained from the web application when registration is complete. It is not possible to place orders without registering on the application. This key is fundamental to the application, serving as the user's authentication token towards the server.

The Flask server script aims to respond to the three main requests that can be made by the users. The first is to receive a request to be processed in Hector. These requests must be made by sending two files: the file with the time series and the configuration file, which includes all the parameters necessary for the correct analysis of the time series in Hector.

When the request is received, a unique ID is assigned to the request, and both files are renamed according to the ID assigned for that request. The time series file is placed in a specific folder on the machine to be processed, the configuration file is analyzed, and the request is added to the database according to the settings provided in the configuration file.

The request may be rejected if the configuration file is not formatted correctly and with all the necessary parameters. The user must create a file in the same folder running the client application with the name *token.id* where he must place his key.

4.3.3.2 Database

The database is a fundamental part of this application, where all requests made are stored. The database was created in SQL with relational tables using the MySQL [27] database engine to manage it. MySQL is an open-source relational database management system and a standard web application.

In addition to all information about requests, information about the users is also recorded. All requests made in the web application or on the command line are sent to the applica-

tion database, and both enter in the same processing queue.

The database has the schema represented in Figure (4.6), with all tables and relationships between them identified. In the Appendix (A.3), it is possible to find the description of all tables.

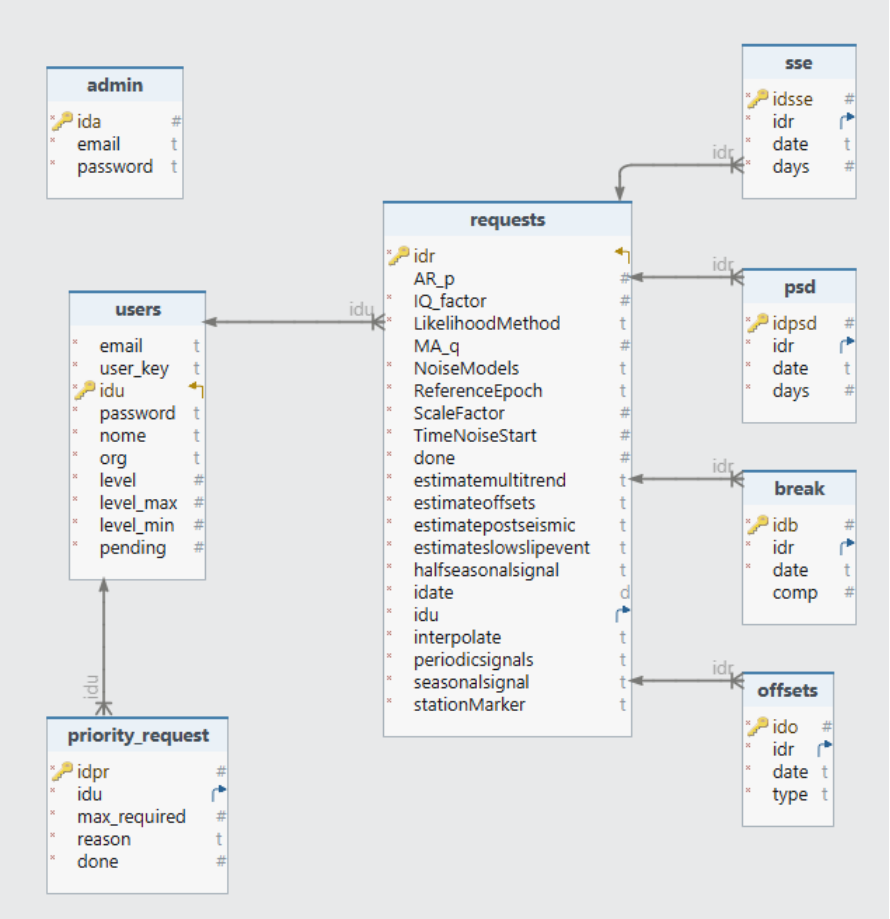

Figure 4.6: Database schema

4.3.3.3 Scheduling system

Due to the complexity of some orders placed with possible long analysis time, it is necessary to implement some selection criteria different from a First In First Out (FIFO) approach. These criteria must also exist because a user can make several requests in a row through the CLI, which can prevent other users to access the TSA in a reasonable period of time. The implemented priority system to minimize these issues are explained below.

Each user has four fundamental parameters. Firstly, the *pending* field represents whether the user has any request pending to be processed. The *level* field represents the user current priority. This field varies between the value of the *level_min* field and the value of the *level_max* field, which represents the maximum priority that the user can achieve, and the *level_min* value that indicates the minimum value that a user priority can reach.

To choose a request to be processed, first all users with pending requests are selected, and then the user with the highest current *level* is chosen for their request to be processed. If two or more users have the same level, then the oldest request is chosen.

Whenever a user has a request processed, their *level* drops by one unit, as long as it is higher than their respective *level_min*. For all users who have pending requests that are not processed, the *level* will be increased by one unit if it is less than his/her *level_max* value.

With this system, a user priority will permanently be changed while they have requests to process, and it will ensure that a user with dozens of requests does not obstruct the queue to the other users.

Users with a higher *level_min* and a lower *level_max* will benefit more, but their priority is not infinite, meaning that all users will have requests processed regardless of their *level_max*.

4.3.3.4 File Structure

Due to the use of a system in which the CLI and the GUI share the same back-end application and the complexity required to run Hector, it is necessary to have a well-defined and organized folder and file system.

This system is fundamental for the correct functioning of the application. The system used will be exemplified in Table (4.4).

Folder	Description
/opt/tsa/hector/	Folder to Execute Hector
/opt/tsa/ctlfiles/idr/	Folder for Hector configuration files
/opt/tsa/solfiles/	Folder to Time-Series Files
/opt/tsa/optionfiles/	Folder for Hector optional files: PSD, offsets, breaks and Slow Slip Events (SSE) files
/opt/tsa/results/idr/	Folder for Hector analysis results
/opt/tsa/api/	Folder for the files that make the CLI application
/opt/flask-app/	Folder for the files that make up the flask server
/var/www/html/TSAbySEGAL/	Folder for the web application

Table 4.4: Table to create the relationship between the folder path and its description

4.4 Software Implementation

In this section, the fundamental parts of the TSA application are exemplified, being divided into two parts: GUI and CLI.

4.4.1 Web application

This section of the document presents the GUI, which has been implemented in the following link: *tsa.segal.ubi.pt*. The GUI is divided into several web pages.

The first three pages are accessible without the user logging in. These are considered the application's home pages.

The user registration page titled *"SignUp"* (cf. Figure (4.7)) is for users to register on the application. It is mandatory to fill in all the fields present there.

Figure (4.7) also shows at the top of the page a menu to navigate between the three home pages.

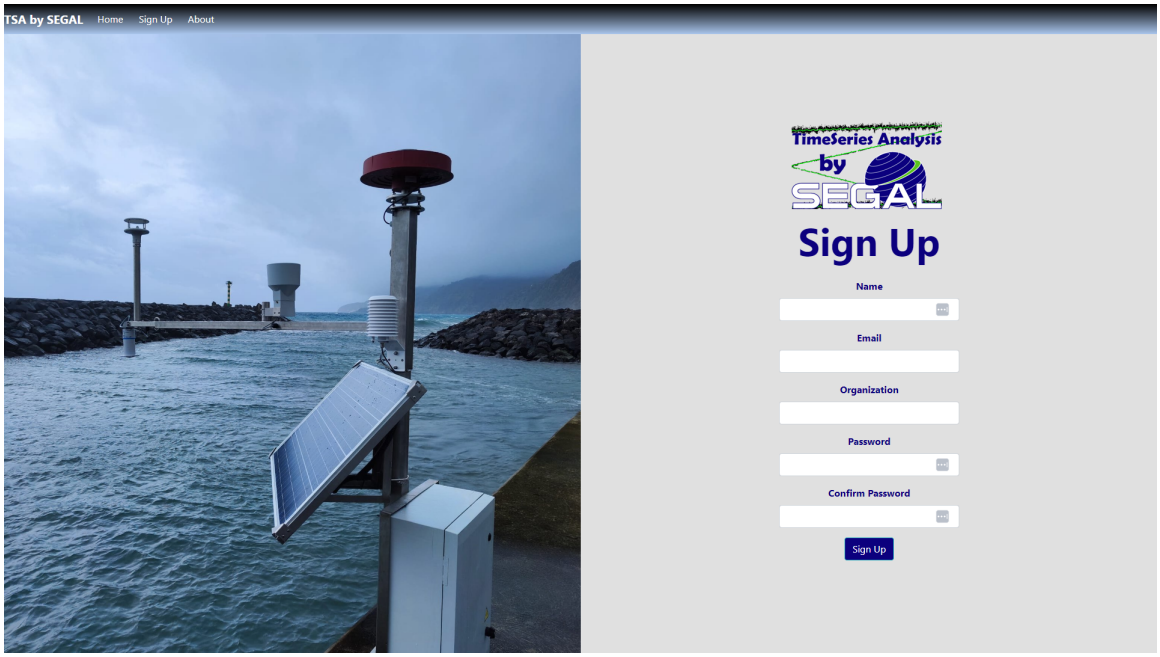

Figure 4.7: Sign Up Page

Figure (4.8) presents *"Login"* page, where the user must use their credentials, email and password to log in to the application.

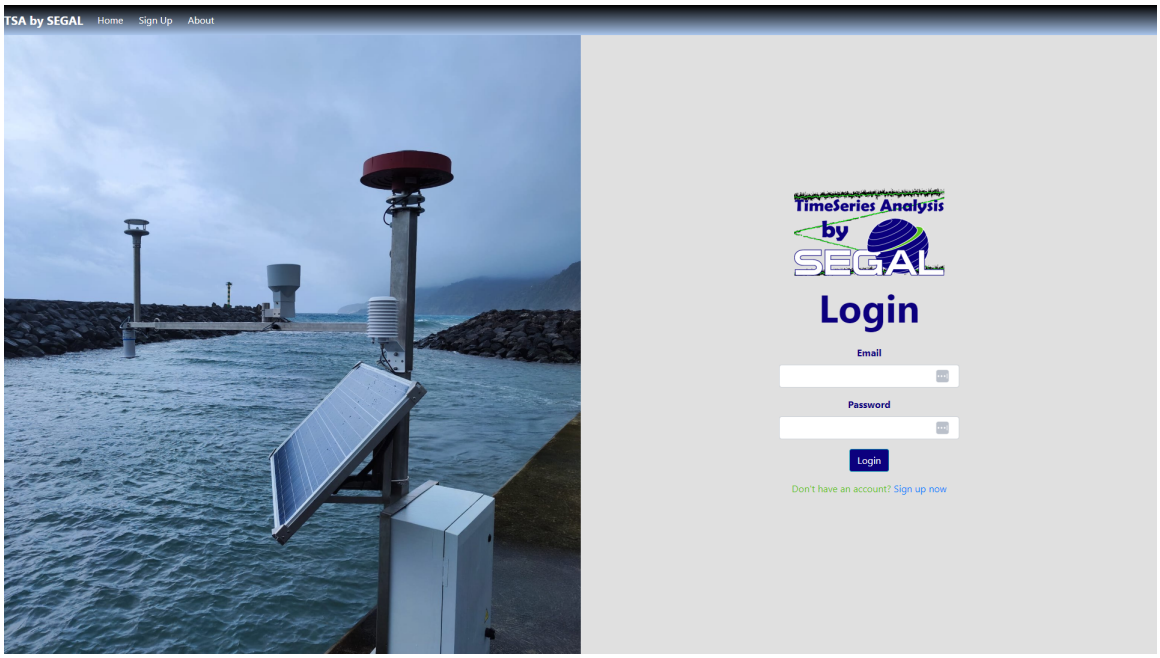

Figure 4.8: Login Page

Finally, Figure (4.9) shows the *"About"* page (cf. Figure (4.9)) where it is possible to find an explanation of how the application works and the steps necessary to use it.

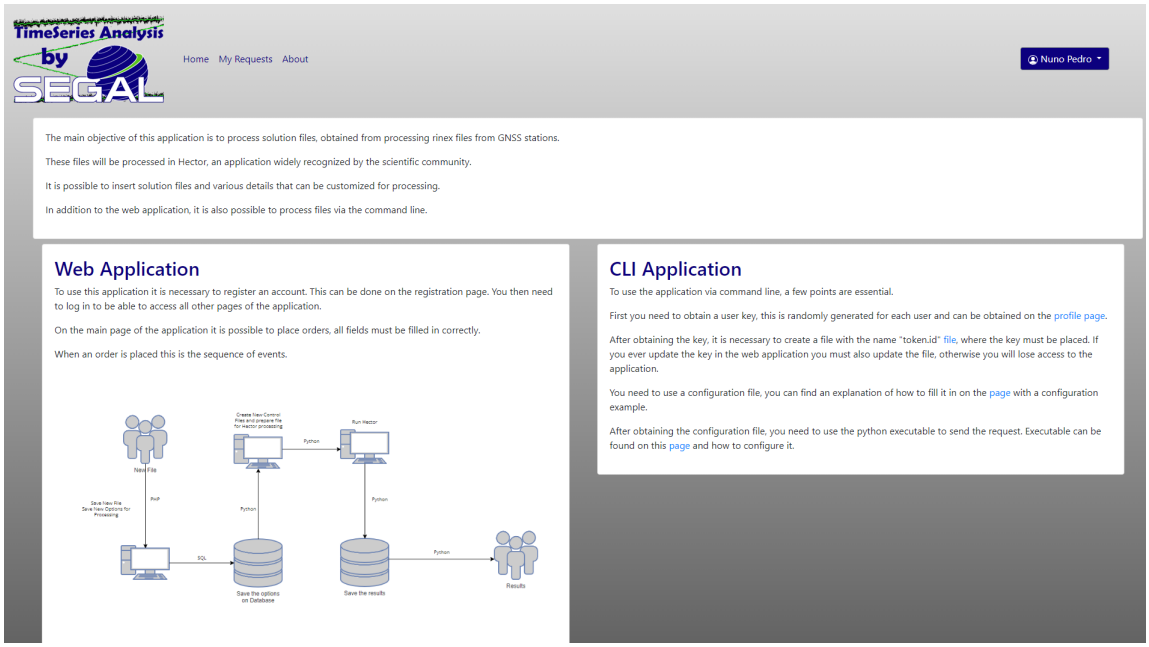

Figure 4.9: About Page

A user must register on the application and log in to access the other TSA pages.

The *"Main"* page of the application (cf. Figure (4.10)) is discussed first. This is the page where the requests are done. It has all the fields necessary to introduce the time series files

and select the different parameters of interest, all with a graphical interface to facilitate user interaction.

It is mandatory to insert the file with the time series to be processed on this page. All fields on this page have a brief explanation to enable the user to remember their functions (a more detailed explanation is available in Hector manual [16]).

The PHP code that allows insert requests into the database using the *"Main"* page can be found in Appendix (A.4).

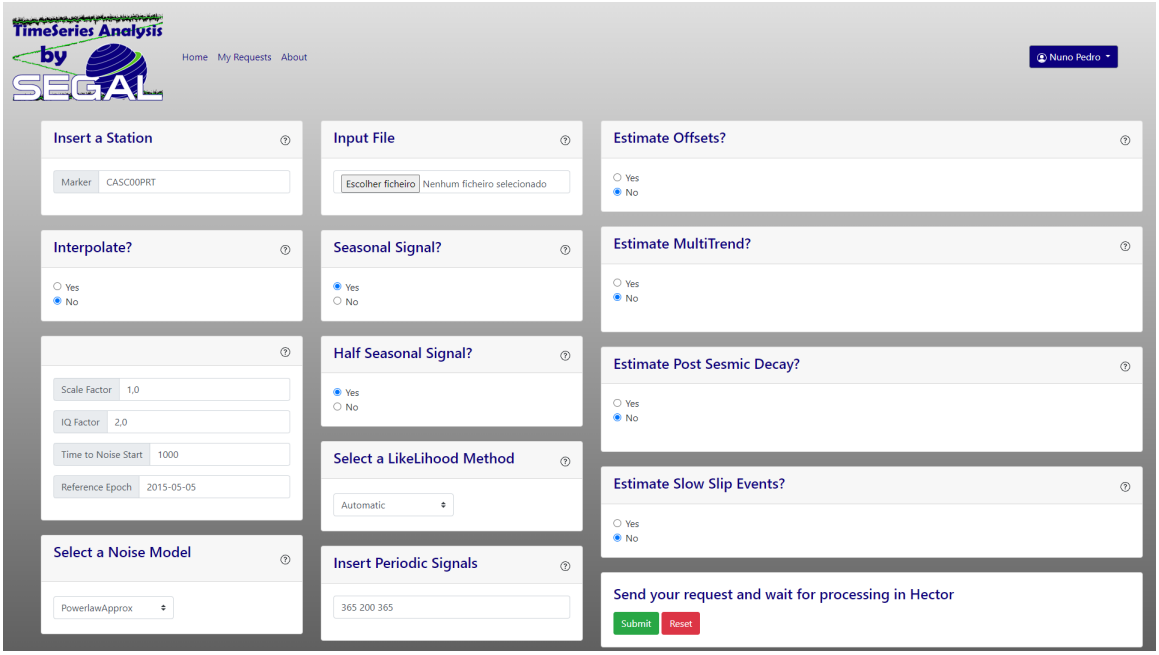

Figure 4.10: Page that the user must use to place a new order

The *"Requests"* page is shown in Figure (4.11). This page contains all requests made by the user. It is possible to review all orders, the associated stations, and the current status, including details about the selected parameters. The user can also delete an order or get the results using two dedicated buttons.

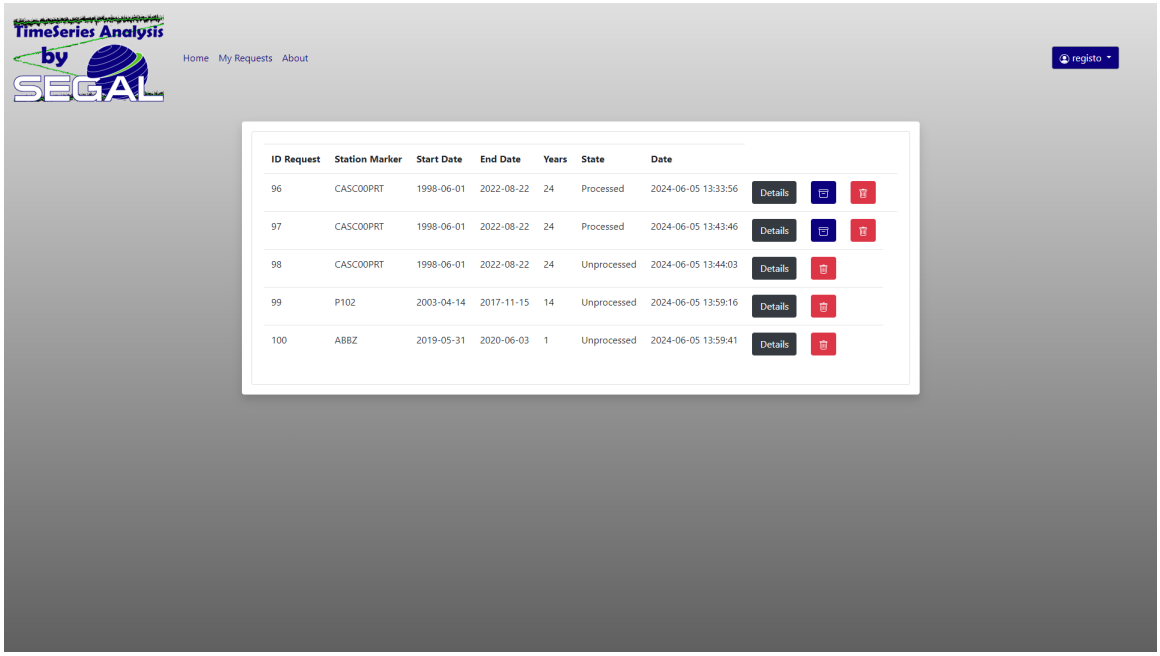

Figure 4.11: This page permit to user view all requests they made

The user *"Profile"* page is shown in Figure (4.12). Here, users can consult their profile and perform fundamental actions.

On this page on the left, the user will find all information about its profile, such as email, name, organization, current priority level, maximum priority level, minimum priority level and the key to use in the application CLI. These values are not editable by the user. At the bottom of the profile information, there is a form where the user can request a new maximum priority level indicating the reason for this request. Each user can only make one request at a time, which must be approved or not by the administrator.

On the right side of the page, there are two different forms. The first allows the user to change their password by entering and confirming a password. The second form allows the user to generate a new random key for use in the CLI application.

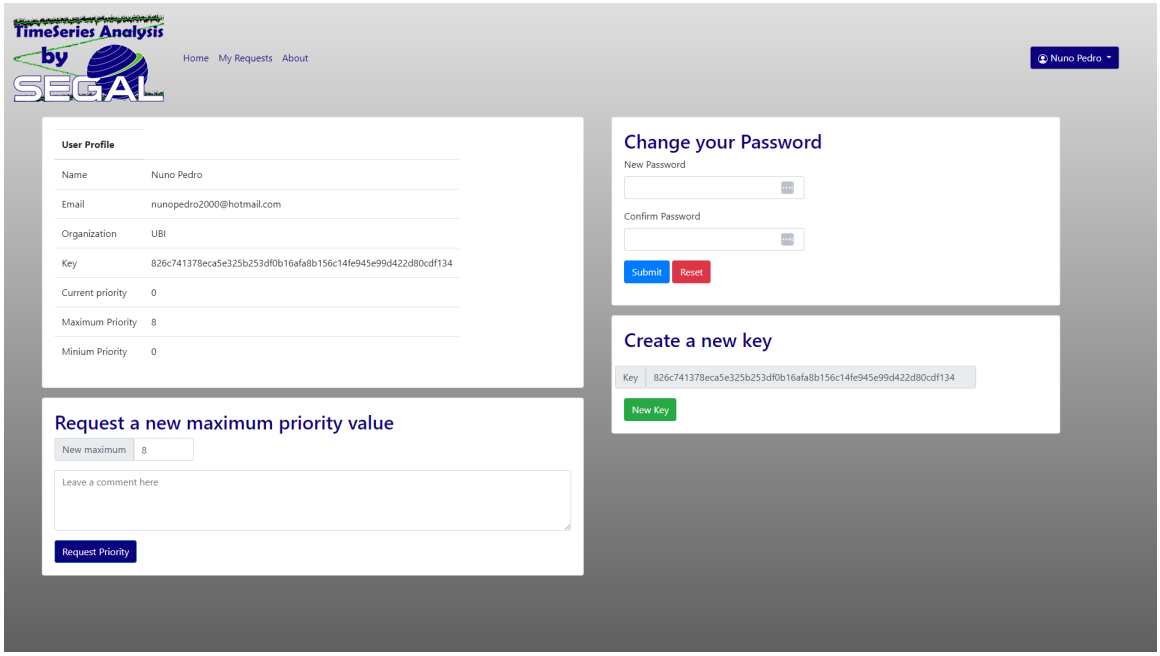

Figure 4.12: This page permits the user to view this profile

4.4.2 CLI

The CLI application is divided into two parts. One part is the server, which is always running on the TSA machine and ready to respond to client requests anytime. The second part is a script that must be executed on the client side to send requests, allow status queries to be performed, and allow request results to be downloaded.

4.4.2.1 Server Functions

The server application was written in Python (see Appendix (A.5)), available through Apache. It is accessed using the address *https://tsa.segal.ubi.pt/api/*. This application consists of three functions that allow the client to communicate with it.

The first function is to receive requests and manage the received files by first analyzing and storing them correctly, assigning them a request number, calling the script responsible for reading and verifying the configuration file, and saving the request in the database. In the end, this function returns the ID order to the client.

The second function, which can be called intermediate, checks the request status. It uses the request ID number to query the database and obtain the request status, returning it to the client. It responds with five possible codes:

- 0 - the order is waiting to be processed;
- 1 – the order is currently being processed;

- 2 - the order has already been completed, and the customer can obtain the results;
- 3 - the order results were obtained, but the files were not yet deleted;
- 4 - the order files have already been deleted;

The last function of the server is to download files. It receives a request ID and returns the file with the results of that order to the user.

The functions described previously are available through the following three endpoints:

- Endpoint: /get_status Parameters: request_id,key
- Endpoint: /receive_request Parameters: key
- Endpoint: /download_files Parameters: request_id,key

All these functions verify the key that the client sends, thus guaranteeing its authenticity.

4.4.2.2 Client Functions

The client application is executed every time that the user wants to send a request, know the status of a request or download results. To use the client application, the user must create a file in the same folder with the name *token.id* where he must place his key obtained in the GUI application.

The user can run the client (which is an executable developed in Python (see Appendix (A.6))) to send a new request or get results regardless of what was done previously.

For each different job, the first part is sending a request where the arguments must contain the path to the file with the time series and the path to the Hector configuration file.

The second part is to obtain the status of the results. This part reads all the request IDs stored in the file *all_id.txt* and obtains their status.

The request sending part sends the two files to the server using the *token.id* file to read the user's key and the *all_id.txt* file to save the ID of the order the user just sent.

The server the status of the requests in the text file *all_id.txt*. The message "Processing not started" is shown if Hector has not begun to analyze a request.

If the order is completed, the results are downloaded, and the order number is deleted from the text file *all_id.txt*. Additionally, after the download of the results, all files related to that particular order are deleted from the TSA machine. So, the user does not have the possibility to download twice the same request. This was implemented to ensure that no user utilizes too much space in the server (the orders are also deleted after 7 days if not downloaded).

4.5 Conclusion

In this chapter, we present the entire design and implementation of the TSA application. The initial design phase and subsequent implementation were explained. Significant effort was carried out to make the GUI intuitive and user-friendly to the user and the access through CLI robust by developing both efficient server and client applications.

Chapter 5

Implementation and Testing

5.1 Introduction

In this chapter, several tests carried out on the application are discussed to test the functionalities mentioned in the previous chapters and also to describe the interactions of the user when exploring the application.

5.2 Implementation

After all the tests carried out on Hector and the development of the TSA application, the entire software package (TSA+Hector) was installed on a dedicated machine, which is accessible at the following link: *tsa.segal.ubi.pt*. This allowed the TSA application and Hector to work together and perform the necessary tests to verify the correct operation of all the functionalities mentioned in Section (4.2.2).

The application database (cf. Section (4.3.3.2)) is also located on the same machine and is responsible for storing all the essential information for the correct functioning of the application.

5.3 Results File

When the user obtains the results of a request, he receives a *.tar* file, which contains all the files shown in Figure (5.1).

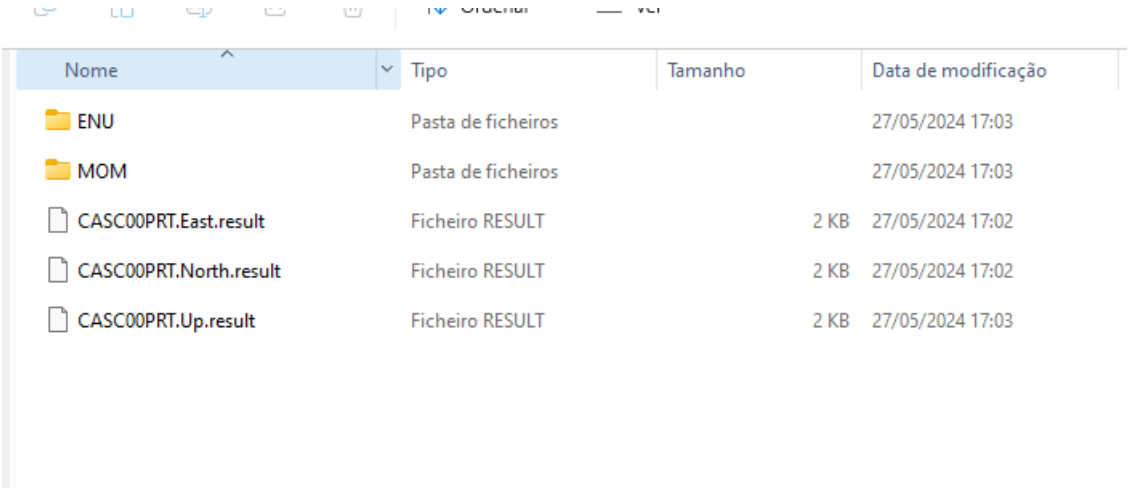

Figure 5.1: Contents of the tar file with results

There are three files with the extension *.result*, one file for each component. These files contain all the information about the station velocity; in Appendix (A.7), it's possible to find an example of one of these files.

The ENU folder contains the time series file converted to ENU format. This format contains four columns: the first column contains the time in Modified Julian Date (MJD), and the second to the fourth column contains the value for the three components(East, North and Up) [16]. In the Appendix (A.8), it is possible to find the first 30 lines of a file in this format.

MOM folder contains two files per component in MOM format (see Figure (5.2)). Each component has a file with the extension *"pre.mom"* (see Appendix (A.9)) that contains two columns: the first is the date in MJD format, and the second is the observed value (the difference between the current observation and the first observation). The second is a file with the extension *"out.mom"* (see Appendix (A.10)) that contains the date in MJD format in the first column, in the second the observed value and in the third the estimated value by Hector.

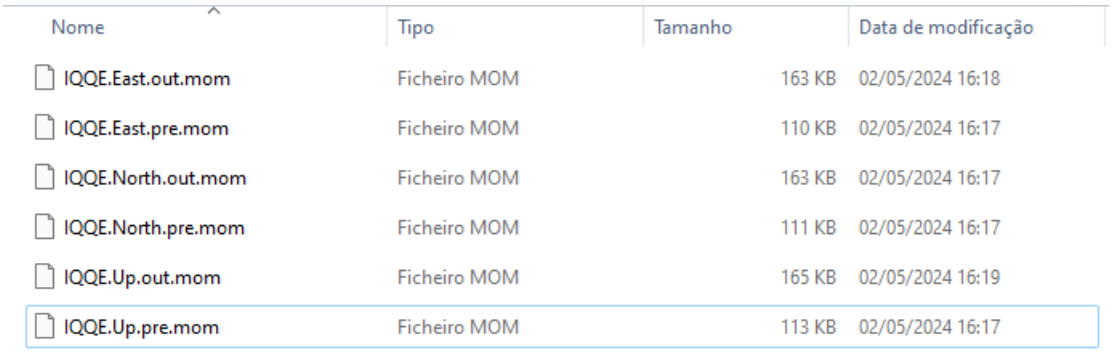

Figure 5.2: List of all files in the MOM folder

5.4 Testing

The first test was to register and login into the web application to verify that all these steps were carried out successfully and that the information is correctly saved in the database. This was carried out using the Selenium [28], which is a common application for this kind of analysis.

Figure (5.3) lists the results showing the user registration and login test were successful.

http://194.210.154.208/TSAbySEGAL/welcome.php

	Command	Target	Value
1	✓ *open*	http://194.210.154.208/TSAbySEGAL/welcome.php	
2	✓ *set window size*	1920x1032	
3	✓ *click*	linkText=Sign up now	
4	✓ *click*	name=username	
5	✓ *type*	name=username	registo
6	✓ *click*	name=email	
7	✓ *type*	name=email	registo@hotmail.com
8	✓ *click*	name=org	
9	✓ *type*	name=org	UBI
10	✓ *click*	name=password	
11	✓ *type*	name=password	1234
12	✓ *click*	name=confirm_password	
13	✓ *type*	name=confirm_password	1234
14	✓ *click*	name=SignUp	
15	✓ *click*	name=email	
16	✓ *type*	name=email	registo@hotmail.com
17	✓ *click*	name=password	
18	✓ *type*	name=password	1234
19	✓ *click*	name=Login	
20	✓ *click*	id=dropdownMenuButton	
21	✓ *click*	linkText=My Profile	

Figure 5.3: Shows the results obtained using the Selenium test to Signup and Sign In

The second test consisted of making a request in the web application and checking whether it was correctly processed by Hector, obtaining the results at the end.

The first step of this test was to make the request on the GUI. Figure (5.4) shows the image of the main page filled in according to the request to be made for a station in Chile (IQQE00CLI). This station was selected because its time series are affected by large earthquakes.

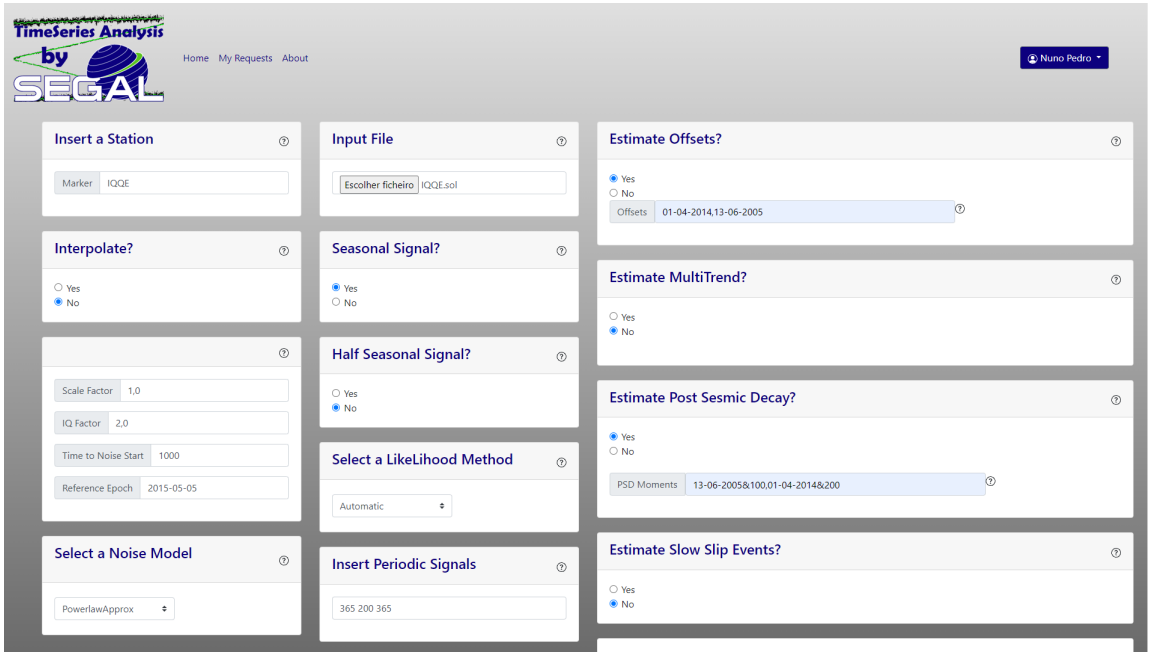

Figure 5.4: Filled out page of a new order for the IQQE station

After placing a request in the web application, we checked whether the request was saved correctly in the database. Figure (5.5) shows the record of the request ID 49 in the database, which corresponds to this particular request.

Table requests

idr	stationMarker	done	idu	interpolate	seasonalsignal	halfseasonalsignal
49	IQQE	0	2	No	Yes	No

periodicsignals	estimateoffsets	estimatepostseismic	estimateslowslipevent
365 200 365	Yes	Yes	No

ScaleFactor	IQ_factor	ReferenceEpoch	NoiseModels	estimatemultitrend
1	2	2015-05-05	PowerlawApprox	No

Table Offsets

ido	idr	date	type
55	49	01-04-2014	All
56	49	13-06-2005	All

Table psd

idpsd	idr	date	days
41	49	13-06-2005	100
42	49	01-04-2014	200

Figure 5.5: Database entries for the request for the IQQE station

It was also checked if the inserted file was saved correctly and in the respective folder; which is shown in Figure (5.6) where are shown the first ten lines of the file containing the time series of daily solutions.

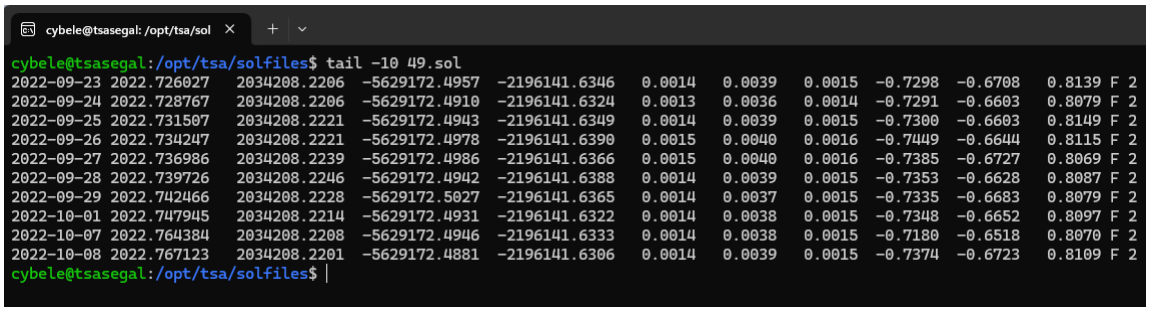

Figure 5.6: Checking of stored Sol file for the IQQE station

After checking that the request was saved correctly, it is necessary to wait for Hector to process it and check if the results files exist and if we can obtain them through the respective page of the application. Figure (5.7) shows the files created by Hector where is

confirmed that the files with the solutions for the three components were successfully created.

Figure 5.7: Request ID 49 results successfully created

Another test was to make a request using the key provided in the web application and making the request via the CLI using the Python client application.

To carry out this test, it was first necessary to create a file with the exact name *token.id* (cf. Figure (5.8)), where it is required to place the user key. This file is mandatory to be able to run the client, and it is necessary to update it whenever the user updates their key in GUI.

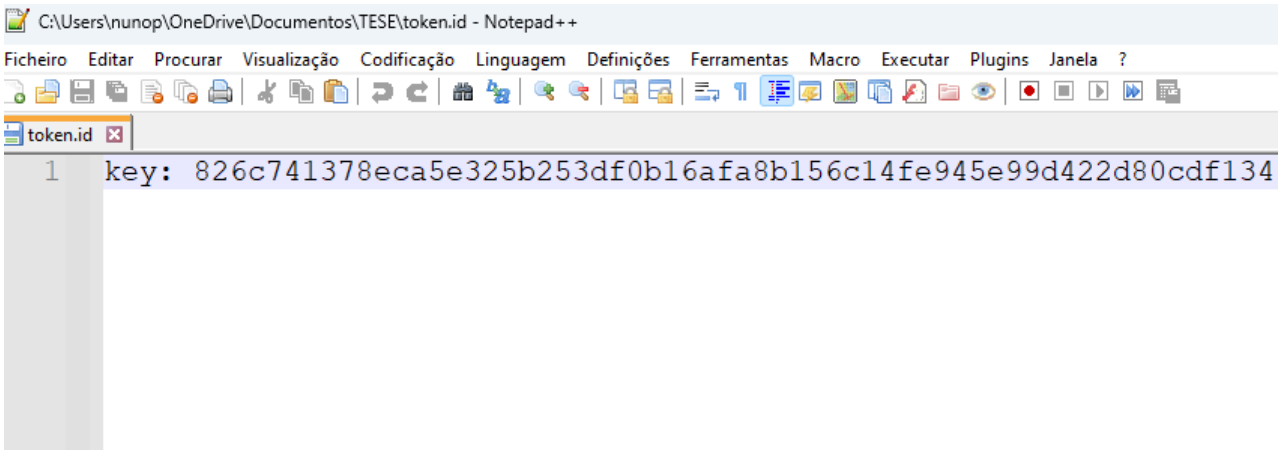

Figure 5.8: Example of the File token.id contains the user key

Secondly, it is necessary to create the configuration file. In this test, we use the oldest station in Portugal CASC00PRT. This file is displayed in Figure (5.9).

```
station-marker CASC00PRT
Offsets 19-04-2008,18-09-2019
Breaks 14-02-2017&1,25-11-2016&2
PSD 20-10-2016&200,25-11-2016&142
SSE 20-10-2018&20,25-11-2012&30
interpolate             yes
seasonalsignal          no
halfseasonalsignal      yes
periodicsignals         2020
estimateoffsets         yes
estimatepostseismic     yes
estimateslowslipevent   yes
estimatemultitrend      yes
ScaleFactor             1.0
IQ_factor               6.0
ReferenceEpoch          2022-05-05
NoiseModels             GGM
LikelihoodMethod        Auto
TimeNoiseStart          1000

```

Figure 5.9: Example of a Hector configuration file

The time series solution file and this file were submitted, using the client executable, through the terminal.

On the client side, we performed the commands listed in Figure (5.10) to send the files and then get the results.

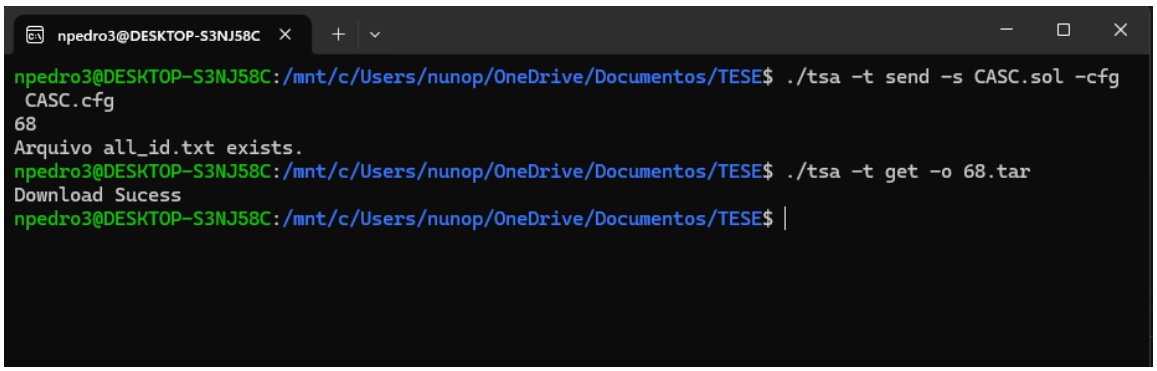

Figure 5.10: Figure shows how to make a request and get the results by command line

Figure (5.11) shows the success of the server in receiving requests, receiving the status query request, the file download request and finally, the request to clean all files.

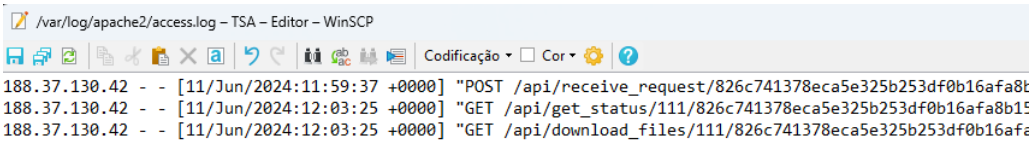

Figure 5.11: Figure shows the server side of receive a request and return results for the user

After carrying out the tests with positive results, error cases were also tested to evaluate the CLI application response to these cases and how the user can perceive the errors made. The following figures show some examples of the tests carried out for possible errors.

The first possible error case is when the user forgets to put some mandatory parameter in the configuration file or puts the wrong configuration. Figure (5.12) shows an example of the message that the user receives when making a mistake in one of the parameters or when it is missing to put the parameter.

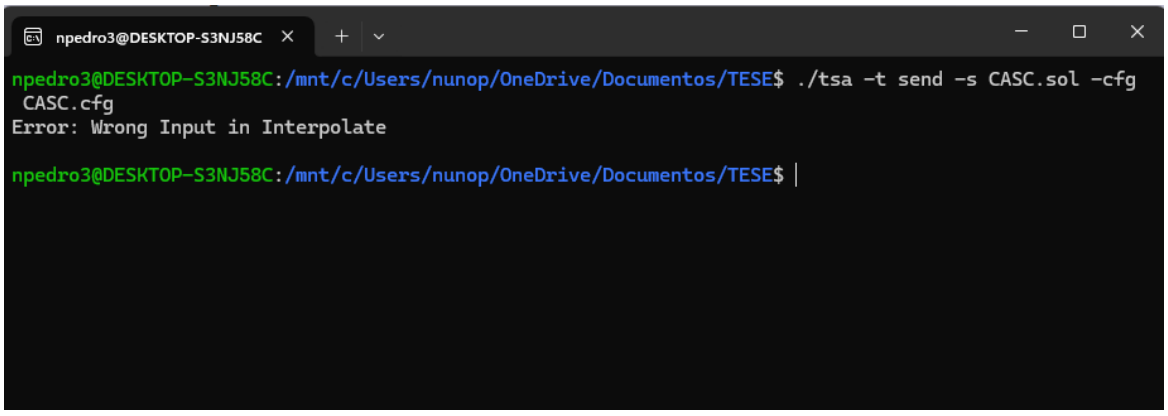

Figure 5.12: This error occurred when the user put a wrong parameter; in this case its an error in the Interpolate parameter

Another possible error the user can make in the configuration file is to put *estimatepost-seismic* at yes and not put post-seismic limit epochs. In this case, the client application also immediately warns the user of this error as shown in Figure (5.13).

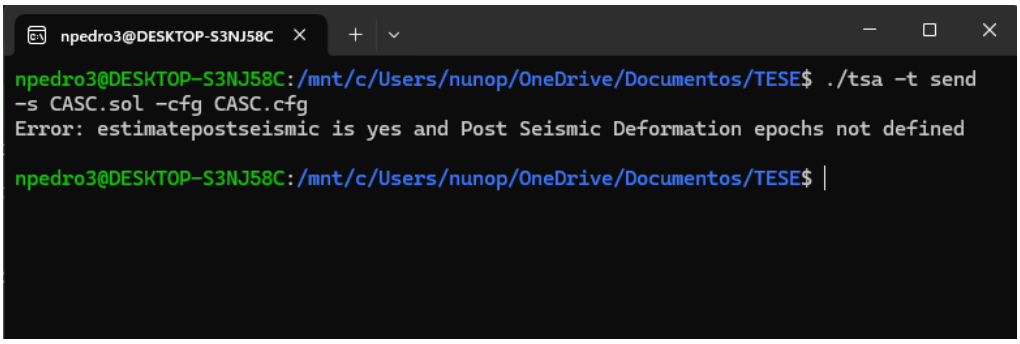

Figure 5.13: Error in missing of epochs for estimation of post-seismic deformation

Another possible error is when the file with the time series has less than ten observations; this is the minimum number of observations that Hector needs to analyze the time series. When this happens, the client application immediately warns the user that the file with the time series does not have the necessary observations, as shown in Figure (5.14).

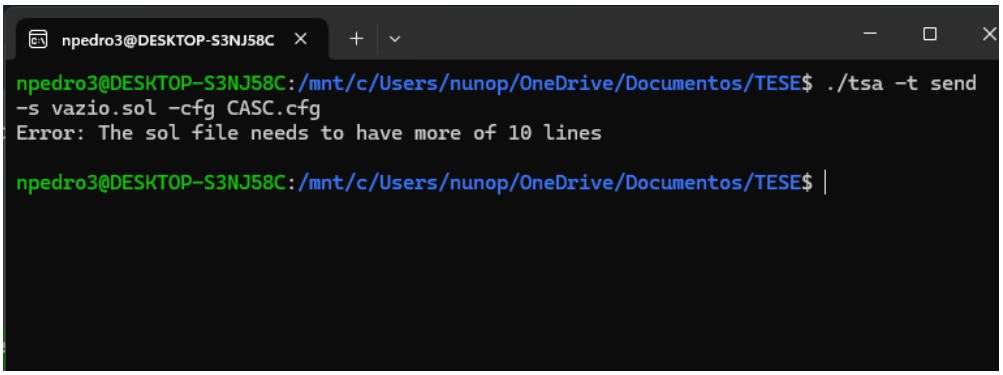

Figure 5.14: File with the time series does not have enough daily solutions to be processed in Hector

5.5 Conclusion

With all the tests carried out previously, it was possible to understand what the application experience should be like and verify if the required functionalities (cf. Section (4.2.2)) were fulfilled. It was also possible to identify possible errors and define accurate messages to the user when there is an error entering information.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This work was focused on the Hector application, where one of the main goals was to optimize its usage. To complete this objective, it was essential to gain an in-depth knowledge of Hector, investigate and analyse its implementation code and understand all the configuration files necessary for its operation.

It has been possible to identify some points that could be optimized and thereby effectively improving the performance of Hector, this can be critical when a large number of time series need to be processed simultaneously. This concluded one of the main objectives of this study.

The development of the TSA web application to run Hector was another fundamental goal. The final application has an GUI component and a CLI component. The CLI component allows further integration since it can be used to analyze routinely time series.

In conclusion all the objectives mentioned at the beginning of this study were successfully achieved.

6.2 Future Work

As Hector is a fundamental tool in the processing of time series, its continuous improvement is always an important goal; this point is also justified due to the increase in the data span of the time series to be analyzed, which makes it always necessary to optimize Hector.

Future work could be on improvements to the TSA application, such as performing metrics in terms of requests made to understand the use of this application or creating specific scripts that allow improving the application security against attacks.

As Hector is a very complex application, it is possible to follow different approaches to those carried out in this study. For example, it is possible to create new complementary algorithms for Hector and compare results with this study.

Bibliography

[1] R. Fernandes, C. Bruyninx, P. Crocker, J.-L. Menut, A. Socquet, M. Vergnolle, A. Avallone, M. Bos, S. Bruni, R. Cardoso *et al.*, "A new european service to share gnss data and products," *Annals of Geophysics*, vol. 65, no. 3, pp. DM317–DM317, 2022. 1

[2] L. TeroMovigo Earth Innovation, "TeroMovigo-Hector," 2022, [Online] https://teromovigo.com/product/hector/. Last access on February 12th, 2024. 1, 8, 9

[3] R. E. N. John M. Dow and C. Rizos, "The international gnss service in a changing landscape of global navigation satellite systems," *Journal of Geodesy*, vol. 83, no. 3, pp. 191–98, 2009. 5

[4] Tallysman, "GNSS Positioning Techniques," 2023, [Online] https://www.tallysman.com/gnss-positioning-techniques/. Last access on February 9th, 2024. 5

[5] M. Bevis and A. Brown, "Trajectory models and reference frames for crustal motion geodesy," *Journal of Geodesy*, vol. 88, pp. 283–311, 2014. 6

[6] S. W. . R. F. Machiel S. Bos, Jean-Philippe Montillet, "Introduction to geodetic time series analysis," *Geodetic Time Series Analysis in Earth Sciences*, pp. 29–52, 2019. 7

[7] M. Bos, R. Fernandes, S. Williams, and L. Bastos, "Fast error analysis of continuous gnss observations with missing data," *Journal of Geodesy*, vol. 87, no. 4, pp. 351–360, 2013. 9, 13

[8] J. Gazeaux, S. Williams, M. King, M. Bos, R. Dach, M. Deo, A. W. Moore, L. Ostini, E. Petrie, M. Roggero *et al.*, "Detecting offsets in gps time series: First results from the detection of offsets in gps experiment," *Journal of Geophysical Research: Solid Earth*, vol. 118, no. 5, pp. 2397–2407, 2013. 9

[9] S. D. P. Williams, "Cats: Gps coordinate time series analysis software," *GPS Solutions*, vol. 12, p. 147–153, 2008. 9

[10] J. Langbein, "Methods for rapidly estimating velocity precision from gnss time series in the presence of temporal correlation: A new method and comparison of existing methods," *J Geophys Res Solid Earth.l*, vol. 125, no. 7, p. e2019JB019132, 2020. 9

[11] W. C. H. Geoffrey Blewitt, Corné Kreemer and J. Gazeaux, "Midas robust trend estimator for accurate gps station velocities without step detection," *J Geophys Res Solid Earth.l*, vol. 121, no. 3, p. 2054–2068, 2016. 10

[12] C. Ltd, "Canonical Ubuntu," 2024, [Online] https://ubuntu.com/. Last access on May 2nd, 2024. 12

[13] T. C. Project, “CentOS,” 2024, [Online] https://centos.org/. Last access on May 3th, 2024. 12

[14] S. C. Foundation, “Standard C++,” 2024, [Online] https://isocpp.org/get-started. Last access on May 3th, 2024. 13

[15] J. Brooks-Bartlett, “Towards Data Science,” 2018, [Online] https://towardsdatascience.com/probability-concepts-explained-maximum-likelihood-estimation-c7b4342fdb Last access on February 25th, 2024. 13

[16] L. TeroMovigo Earth Innovation, “Hector user manual,” 2021, [Online] https://teromovigo.com/wp-content/uploads/hector/hector_manual_2.0.pdf. Last access on May 30th, 2024. 13, 36, 42

[17] I. Foundation, “tcsh - Invoke a C shell,” 2023, [Online] https://www.ibm.com/docs/en/zos/3.1.0?topic=descriptions-tcsh-invoke-c-shell. Last access on May 2nd, 2024. 17

[18] O. A. R. Board, “OpenMP Application Programming Interface,” 2024, [Online] https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5.0.pdf. Last access on April 7th, 2024. 19

[19] W. W. W. Consortium, “HTML5,” 2021, [Online] https://www.w3.org/TR/html5/. Last access on May 25th, 2024. 30

[20] W. Consortium, “CSS: Cascading Style Sheets,” 2021, [Online] https://www.w3.org/Style/CSS/. Last access on May 23rd, 2024. 30

[21] B. team, “Bootstrap,” 2022, [Online] https://getbootstrap.com/. Last access on February 25th, 2024. 30

[22] M. D. Network, “JavaScript,” 2021, [Online] https://developer.mozilla.org/en-US/docs/Web/JavaScript. Last access on May 10th, 2024. 30

[23] P. Group, “PHP,” 2024, [Online] https://www.php.net/. Last access on May 10th, 2024. 30

[24] T. A. S. Foundation, “Apache HTTP Server,” 2024, [Online] https://httpd.apache.org/. Last access on May 10th, 2024. 30

[25] Flask, “Flask,” 2010, [Online] https://flask.palletsprojects.com/en/3.0.x/. Last access on February 10th, 2024. 31

[26] P. S. Foundation, “Python Documentation,” 2023, [Online] https://docs.python.org/3/. Last access on May 18th, 2024. 31

[27] Oracle, “MySQL,” 2024, [Online] https://www.mysql.com/. Last access on February 29th, 2024. 31

[28] Selenium, “ESelenium Dev,” 2024, [Online] https://www.selenium.dev/. Last access on April 7th, 2024. 43

Appendix A

Appendix

A.1 JavaScript Use on Application

```
1  //Open Menu Profile
2  function abrir() {
3    var testElement = document.getElementById('opcoes');
4    if (testElement.classList.contains('d-block')) {
5      $('#opcoes').removeClass("d-block");
6    } else {
7      $('#opcoes').addClass("d-block");
8    }
9  }
10 // Get input for new priority request
11 function getInputValues(user) {
12   var idElemento = "NewLevelR." + user;
13   var levelValue = document.getElementById(idElemento).value;
14   $.ajax({
15     url: "pedidos.php",
16     type: "POST",
17     data: {
18       "idu": user,
19       "newlevel": levelValue
20     }
21   }).done(function (data) {
22     document.getElementById("pedidos").innerHTML = data;
23   });
24
25 }
26 // Show PSD Input
27 function showHidePSDDiv(value) {
28   var psdDiv = document.getElementById("psd-div");
29
30   if (value === "Yes") {
31     psdDiv.style.display = "block";
32   } else {
33     psdDiv.style.display = "none";
34   }
35 }
36 // Show Break Input
```

```
37 function showHideMultiDiv(value) {
38   var multiDiv = document.getElementById("break-div");
39
40   if (value === "Yes") {
41     multiDiv.style.display = "block";
42   } else {
43     multiDiv.style.display = "none";
44   }
45 }
46 // Show Offsets Input
47 function showHideOffDiv(value) {
48   var offiDiv = document.getElementById("off-div");
49
50   if (value === "Yes") {
51     offiDiv.style.display = "block";
52   } else {
53     offiDiv.style.display = "none";
54   }
55 }
56 // Show Slow Slip Events Input
57 function SlowSlipEvents(value) {
58   var sseDiv = document.getElementById("sse-div");
59
60   if (value === "Yes") {
61     sseDiv.style.display = "block";
62   } else {
63     sseDiv.style.display = "none";
64   }
65 }
66 // Function to send data to get a new key
67 function changeKey() {
68   $.ajax({
69     url: "perfil.php",
70     type: "POST",
71     data: { "newKey": 1 }
72   }).done(function (data) {
73     document.getElementById("chave").innerHTML = data;      //div onde mostra os
         resultados, neste caso ResultadoOpcao
74   });
75 }
76 // Function to send data to delete request
77 function eliminarRre(id) {
78   console.log("Entrou")
79   if (confirm("Are you sure delete " + id + " request")) {
80     $.ajax({
81       url: "index.php",
82       type: "POST",
```

```
83        data: { "requestIDE": id }
84      }).done(function (data) {
85        document.getElementById("tableR").innerHTML = data;      //div onde mostra
              os resultados, neste caso ResultadoOpcao
86        window.location.reload();
87      });
88    }
89  }
90  // Function to show details of request
91  function Details(id) {
92    $.ajax({
93      url: "one.php",
94      type: "POST",
95      data: { "detail": id }
96    }).done(function (data) {
97      document.getElementById("tableR").innerHTML = data;
98    });
99  }
100 // Function to download files results
101 function downloadFile(numero_pedido,key) {
102   fetch(`https://tsa.segal.ubi.pt/api/download_files/${numero_pedido}/${key}`)
103     .then(response => {
104       if (!response.ok) {
105         throw new Error('Error download File');
106       }
107       return response.blob();
108     })
109     .then(blob => {
110       const url = URL.createObjectURL(blob);
111       const link = document.createElement('a');
112       link.href = url;
113       link.download = `arquivo_${numero_pedido}.tar`;
114       link.click();
115       URL.revokeObjectURL(url);
116     })
117     .catch(error => {
118       console.error('Erro:', error);
119     });
120 }
```

A.2 Apache Configuration File

```
1 <IfModule mod_ssl.c>
2 <VirtualHost *:443>
3     ServerAdmin webmaster@localhost
```

```
4       DocumentRoot /var/www/html
5
6       # Config of Flash to use API
7       WSGIDaemonProcess flask-apis user=www-data group=www-data threads=5 python-
            home=/opt/flask-app/flask-venv
8       WSGIScriptAlias /api /opt/flask-app/flask-app.wsgi
9
10      # Directory to flask application
11      <Directory /opt/flask-app>
12          WSGIProcessGroup flask-apis
13          WSGIApplicationGroup %{GLOBAL}
14          Require all granted
15      </Directory>
16
17      # Directory for PHP application
18      <Directory /var/www/html/TSAbySEGAL>
19          Options Indexes FollowSymLinks
20          AllowOverride All
21          Require all granted
22      </Directory>
23
24      # Apache Logs
25      ErrorLog ${APACHE_LOG_DIR}/error.log
26      CustomLog ${APACHE_LOG_DIR}/access.log combined
27
28      # Rules for redirect requests for API
29      RewriteEngine On
30      RewriteCond %{REQUEST_URI} !^/api
31      RewriteRule ^/?$ /TSAbySEGAL/welcome.php [L,R=301]
32  #Configs for https
33  ServerName tsa.segal.ubi.pt
34  SSLCertificateFile /etc/letsencrypt/live/tsa.segal.ubi.pt/fullchain.pem
35  SSLCertificateKeyFile /etc/letsencrypt/live/tsa.segal.ubi.pt/privkey.pem
36  Include /etc/letsencrypt/options-ssl-apache.conf
37  </VirtualHost>
38  </IfModule>
```

A.3 DataBase Description

Users, This table contains all information about users registered in the application.

- idu, the primary key of the table;
- email, user email;

- nome, user name;
- password, hashed encrypted password;
- user_key, the user key to be used in the configuration file for CLI requests;
- org, user organization;
- level, user's current priority number
- level_max, maximum user priority number
- pending, an identifier for whether the user has pending requests

Requests, This table contains all requests made on the application.

- idr, the primary key of table;
- station-marker, the station marker
- idu, id of the user who made the request;
- interpolate, If there is interpolate;
- offsets, If there is a calculation considering the offsets;
- quadratic, If the sign is quadratic;
- seasonalsignal, Whether there is seasonal signal processing;
- halfseasonalsignal, Whether there is half seasonal signal processing;
- NoiseModels, Noise model for processing in Hector;
- periodicsignals, If there are periodic signs
- estimateoffsets, If there are offsets in the data and if data were entered to estimate
- estimatepostseismic,If there is PSD in the data and if data were introduced to estimate
- estimateslowslipevent, If there is SSE and trend in the data and if data were introduced to estimate
- estimatemultitrend, If there is a multi trend in the data and if data were introduced to estimate
- LikelihoodMethod, The method to be used
- ReferenceEpoch, the reference period for the calculation
- AR_p, Ar value for Hector Processing;

- MA_q, Ma value for Hector Processing;
- ScaleFactor, Scale for scaling the time series;
- TimeNoiseStart, Time to start noise in time series;
- IQ_factor, Inter Quartile Factor for Hector Processing;
- done, if the order is completed;
- idate, order creation date

PSD, This table contains all PSD moments entered by the user. All moments are associated with a request and are used in Hector processing.

- idpsd, the primary key of table;
- date, the initial date of the PSD moment;
- idr, the corresponding request ID;
- days, the number of days that decay takes place;
- marker, the marker of the respective station;

Offsets, These tables contain all offset values entered by users that are used in Hector's processing.

- ido, the primary key of table;
- idr, the corresponding request ID;
- date, the date of the offset;
- type, the type of the offset;

Break, This table contains information about break moments used to calculate a component-time series with different speeds.

- idb, the primary key of table;
- idr, the corresponding request ID;
- date, the date of the respective break;
- comp, the component of the respective break;

SSE, This table contains information about the SSE, these servers for calculating time series with slow slip events.

- idsse, the primary key of table;
- idr, the corresponding request ID;
- date, the start date of SSE;
- days, the number of days that SSE;

priority_request, This table contains the maximum priority change requests made by users

- idpr, the primary key of table;
- idu, id of the user who made the request;
- max_required, the new top priority required
- reason, justification for the request
- done, if the request has already been evaluated

admin, This table contains the administrator list of the application

- ida, the primary key of table;
- email, admin email;
- password, hashed encrypted password;

A.4 Upload Page

```
1  <?php
2  require_once "header/header.php";
3  session_start();
4
5  function getFirstAndLastDate($filename) {
6    $file = fopen($filename, 'r');
7    if (!$file) {
8        die('Error to open the file.');
9    }
10
11   $firstDate = null;
12   $lastDate = null;
13   $lineNumber = 0;
14
15   while (($row = fgetcsv($file, 0, ' ')) !== FALSE) {
16       if (count($row) == 0) {
```

```
17              continue;
18          }
19
20          if ($lineNumber == 0) {
21              $firstDate = $row[0];
22          }
23          $lastDate = $row[0];
24          $lineNumber++;
25      }
26
27      fclose($file);
28
29      if ($firstDate === null || $lastDate === null) {
30          die('The sol file not have valid data');
31      }
32
33      return array($firstDate, $lastDate);
34  }
35  function get_max_idr($file_path) {
36      if (!file_exists($file_path)) {
37          return null;
38      }
39
40      $file_contents = file_get_contents($file_path);
41      $lines = explode(PHP_EOL, $file_contents);
42
43      $max_idr = null;
44      foreach ($lines as $line) {
45          $idr = intval(trim($line));
46          if ($idr > $max_idr) {
47              $max_idr = $idr;
48          }
49      }
50
51      return $max_idr;
52  }
53
54  $email =  htmlspecialchars($_SESSION["email"]);
55  $SQL3 = "SELECT idu FROM users WHERE email = '";
56  $SQL3 .= $email;
57  $SQL3 .= "';";
58  $result3 = mysqli_query($link, $SQL3);
59  $nu3 = mysqli_fetch_row($result3);
60
61
62  $sql5 = "SELECT MAX(idr) FROM requests";
63  $result5 = mysqli_query($link, $sql5);
```

```
64 $col5 = mysqli_fetch_row($result5);
65 $col5[0] = $col5[0] + 1;
66 $idr= $col5[0];
67
68 $file_path = '/opt/tsa/api/forProcess.txt';
69 $max_idr = get_max_idr($file_path);
70
71 if ($max_idr === null) {
72     $max_idr = 0;
73 }
74
75 $idr = isset($idr) ? $idr : 0;
76 if ($max_idr >= $idr) {
77     $idr = $max_idr + 1;
78 }
79
80
81 $station = $_POST['marker'];
82
83 if ($station == -1) {
84
85   echo '<script>
86 alert("Please choice a station");
87 window.location.href="index.php";
88   </script>';
89 } else {
90   $target_dir = "/opt/tsa/solfiles/";
91   $file = $_FILES['my_file']['name'];
92   $path = pathinfo($file);
93   $filename = $path['filename'];
94   $ext = $path['extension'];
95   $temp_name = $_FILES['my_file']['tmp_name'];
96   move_uploaded_file($temp_name,$target_dir.$col5[0].".sol");
97   $target_dir2 = "/opt/tsa/solfiles/".$col5[0].".sol";
98   shell_exec("chmod 777 $target_dir2");
99
100 list($firstDate, $lastDate) = getFirstAndLastDate($target_dir2);
101
102
103   $interpolate = $_POST['interpolate'];
104   $estimateoff = $_POST['estimateoff'];
105   $ss = $_POST['ss'];
106   $hss = $_POST['hss'];
107   $NoiseModel = $_POST['NoiseModel'];
108   $sf = $_POST['sf'];
109   $tns = $_POST['tns'];
110   $iqfactor = $_POST['iqfactor'];
```

```
111  $llhood = $_POST['llhood'];
112  $estimatemulti =  $_POST['estimatemulti'];
113  $estimatepsd = $_POST['estimatepsd'];
114  $estimateSSE = $_POST['sse'];
115  $periodic= $_POST['ps'];
116  $referencepoch =$_POST['re'];
117
118  if ($estimatemulti == "Yes") {
119    $breaks = $_POST['breaks'];
120    $arrayDeBreaks = explode(",", $breaks);
121    foreach ($arrayDeBreaks as $valor) {
122      $valores = explode("&", $valor);
123      $idr = mysqli_real_escape_string($link, $idr);
124      $valor = mysqli_real_escape_string($link, $valor);
125      if (!in_array($valores[1], [1, 2, 3])) {
126        if ($valores[1] == 'N') {
127            $valores[1] = 1;
128        }
129        if ($valores[1] == 'E') {
130            $valores[1] = 2;
131        }
132        if ($valores[1] == 'U') {
133            $valores[1] = 3;
134        }
135    }
136      $sql8 = "INSERT INTO break (idr, date,comp) VALUES ('$idr', '$valores
         [0]','$valores[1]')";
137      $result8 = mysqli_query($link, $sql8);
138      }
139  }
140  if ($estimatepsd == "Yes") {
141    $psdm = $_POST['psdm'];
142    $arrayDePSD = explode(",", $psdm);
143    foreach ($arrayDePSD as $valor) {
144      $valores = explode("&", $valor);
145      $idr = mysqli_real_escape_string($link, $idr);
146      $sql9 = "INSERT INTO psd (idr, date,days) VALUES ('$idr', '$valores[0]','
         $valores[1]')";
147      $result9 = mysqli_query($link, $sql9);
148      }
149  }
150  if ($estimateoff == "Yes") {
151    $offsets = $_POST['offsets'];
152    $arrayDeOffs = explode(",", $offsets);
153    foreach ($arrayDeOffs as $valor) {
154      $idr = mysqli_real_escape_string($link, $idr);
155      $valor = mysqli_real_escape_string($link, $valor);
```

```
156         $sql7 = "INSERT INTO offsets (idr, date, type) VALUES ('$idr', '$valor','
                All')";
157         $result7 = mysqli_query($link, $sql7);
158         }
159     }
160     if ($estimateSSE == "Yes") {
161       $ssem = $_POST['ssem'];
162       $arrayDesse = explode(",", $ssem);
163       foreach ($arrayDesse as $valor) {
164         $valores = explode("&", $valor);
165         $idr = mysqli_real_escape_string($link, $idr);
166         $sql14 = "INSERT INTO sse (idr, date,days) VALUES ('$idr', '$valores
                [0]','$valores[1]')";
167         $result14 = mysqli_query($link, $sql14);
168         }
169     }
170     if ($NoiseModel == 'ARFIMA' || $NoiseModel == 'ARMA') {
171       $arc = $_POST['arc'];
172       $mac = $_POST['mac'];
173     } else {
174       $arc = 0;
175       $mac = 0;
176     }
177     $idu = $nu3[0];
178     $done = 0;
179     $sql1 = "INSERT INTO requests (idr,stationMarker,done,idu,interpolate,
            seasonalsignal,halfseasonalsignal,periodicsignals,estimateoffsets,
            estimatepostseismic,estimateslowslipevent,estimatemultitrend,ScaleFactor,
            IQ_factor,ReferenceEpoch,NoiseModels,LikelihoodMethod,TimeNoiseStart,AR_p
            ,MA_q,start_date,end_date) VALUES
            (?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?,?)"; //SQL parea inserir o
            pedido na base de dados
180     if ($stmt2 = mysqli_prepare($conn, $sql1)) {
181       mysqli_stmt_bind_param($stmt2, "isiissssssssddsssiddss",$idr, $station,
              $done, $idu, $interpolate, $ss, $hss, $periodic, $estimateoff,
              $estimatepsd, $estimateSSE, $estimatemulti, $sf, $iqfactor,
              $referencepoch,$NoiseModel,$llhood,$tns,$arc,$mac,$firstDate, $lastDate
              );
182     }
183     if (!mysqli_stmt_execute($stmt2)) {
184       echo 'erro: ' . mysqli_error($conn);
185     } else {
186       echo '<script>
187     alert("Request Send");
188     window.location.href="index.php";
189       </script>';
190     }
```

```
}
```

A.5 Server Application Code

```
from flask import Flask, request, jsonify, send_file
import os
import mysql.connector
import subprocess
from flask_cors import CORS
import shutil

app = Flask(__name__)
app.config['ENV'] = 'production'
CORS(app)
def update_request_status(idr):
    try:
        mydb = mysql.connector.connect(
            host="localhost",
            user="nunop",
            password="pjeYmG26COSfK-7H",
            database="hector-api"
        )

        cursor = mydb.cursor()
        sql = "UPDATE requests SET done = 3 WHERE idr = %s"
        val = (idr,)
        cursor.execute(sql, val)
        mydb.commit()

    except mysql.connector.Error as err:
        print("Error: ", err)

    finally:
        if mydb.is_connected():
            cursor.close()
            mydb.close()

def check_key_validity(key):
    mydb = mysql.connector.connect(
        host="localhost",
        user="nunop",
        password="pjeYmG26COSfK-7H",
        database="hector-api"
    )
    mycursor = mydb.cursor()
```

```
42      mycursor.execute(f"SELECT idu FROM users WHERE user_key = '{key}'")
43      result = mycursor.fetchone()
44      mycursor.close()
45      mydb.close()
46      return result is not None
47  def get_max_idr(file_path):
48      max_idr = None
49      with open(file_path, 'r') as file:
50          for line in file:
51              idr_str, _ = line.strip().split()
52              idr = int(idr_str)
53              if max_idr is None or idr > max_idr:
54                  max_idr = idr
55      return max_idr
56  def get_idu(key):
57      mydb = mysql.connector.connect(
58          host="localhost",
59          user="nunop",
60          password="pjeYmG26COSfK-7H",
61          database="hector-api"
62      )
63      mycursor = mydb.cursor()
64      mycursor.execute(f"SELECT idu FROM users WHERE user_key = '{key}'")
65      result = mycursor.fetchone()
66      return result[0]
67
68  def get_idu_request(idr):
69      mydb = mysql.connector.connect(
70          host="localhost",
71          user="nunop",
72          password="pjeYmG26COSfK-7H",
73          database="hector-api"
74      )
75      mycursor = mydb.cursor()
76      mycursor.execute(f"SELECT idu FROM requests WHERE idr = '{idr}'")
77      result = mycursor.fetchone()
78      return result[0]
79  @app.route('/receive_request/<string:key>', methods=['POST'])
80  def receive_request(key):
81      pedidos  = []
82      mydb = mysql.connector.connect(
83      host="localhost",
84      user="nunop",
85      password="pjeYmG26COSfK-7H",
86      database="hector-api"
87  )
88      if check_key_validity(key):
```

```
89          id_user = get_idu(key)
90          mycursor = mydb.cursor()
91          mycursor.execute("SELECT MAX(idr) FROM requests")
92          result = mycursor.fetchone()
93          numero_pedido = result[0] + 1 if result[0] is not None else 1
94          file_path = '/opt/tsa/api/forProcess.txt'
95          max_idr = get_max_idr(file_path)
96          if max_idr is None :
97              max_idr = 0
98          if max_idr >= numero_pedido :
99              numero_pedido = max_idr +1
100         diretorio_arquivos = '/opt/tsa/api/files/'
101
102         arquivo1 = request.files.get('arquivoSOL')
103         arquivo2 = request.files.get('arquivoCFG')
104
105         diretorio_conf = os.path.join(diretorio_arquivos, str(numero_pedido))
106         os.makedirs(diretorio_conf, exist_ok=True)
107         caminho_arquivo_conf = os.path.join(diretorio_conf, 'arquivo.cfg')
108
109         arquivo2.save(caminho_arquivo_conf)
110         caminho_arquivo_sol = os.path.join(diretorio_conf, 'arquivo.sol')
111         arquivo1.save(caminho_arquivo_sol)
112         os.chmod(caminho_arquivo_conf, 00777)
113         os.chmod(caminho_arquivo_sol, 00777)
114         os.chmod(diretorio_conf, 00777)
115         print(numero_pedido)
116         output_file = f"/opt/tsa/api/out/{numero_pedido}.out"
117         if os.path.exists(output_file):
118             os.remove(output_file)
119         command = "python3 /opt/tsa/api/verificador.py " + str(numero_pedido) +
                " " + str(id_user) +" >> /opt/tsa/api/out/"+str(numero_pedido)+".
                out"
120         out_file = "/opt/tsa/api/out/"+str(numero_pedido)+".out"
121         os.system(command)
122         os.chmod(out_file, 00777)
123         with open(output_file, 'r') as f:
124             if 'Error:' in f.read():
125                 with open(output_file, 'r') as f:
126                     return jsonify({'pedidos': f.read()})
127             else:
128                 return jsonify({'pedidos': numero_pedido})
129     else:
130         return jsonify({'Error:': 'Invalid Key'}), 400
131 @app.route('/get_status/<int:numero_pedido>/<string:key>', methods=['GET'])
132 def get_status(numero_pedido, key):
133     if check_key_validity(key):
```

```
134         mydb = mysql.connector.connect(
135             host="localhost",
136             user="nunop",
137             password="pjeYmG26COSfK-7H",
138             database="hector-api"
139         )
140         mycursor = mydb.cursor()
141         mycursor.execute(f"SELECT done FROM requests WHERE idr = {numero_pedido
                }")
142         result = mycursor.fetchone()
143         mycursor.close()
144         mydb.close()
145         return jsonify({'Info': result[0]})
146     else:
147         return jsonify({'Error:': 'Invalid Key'}), 400
148
149 @app.route('/download_files/<int:numero_pedido>/<string:key>', methods=['GET'])
150 def download_files(numero_pedido,key):
151     output_file = f"/opt/tsa/api/out/{numero_pedido}.out"
152     if check_key_validity(key):
153         extensao_arquivo = "tar"
154         nome_arquivo = str(numero_pedido) + "/" + str(numero_pedido) + "." +
                extensao_arquivo
155         diretorio_arquivos = '/opt/tsa/results/'
156         caminho_arquivo = os.path.join(diretorio_arquivos, nome_arquivo)
157         if os.path.exists(caminho_arquivo):
158             update_request_status(numero_pedido)
159             return send_file(caminho_arquivo, as_attachment=True)
160         else:
161             with open(output_file, 'r') as f:
162                 return jsonify({'Error': f.read()}), 400
163     else:
164         return jsonify({'Error': 'Invalid Key'}), 400
165
166
167 @app.route('/')
168 def api_index():
169     return 'API Index'
170
171 if __name__ == '__main__':
172     app.run()
```

A.6 Client Application Code

```
1 import requests
```

```
2  import json
3  import sys
4  import argparse
5  import time
6  import os
7  url='https://tsa.segal.ubi.pt/api/'
8  def read_key(filename, key):
9      with open(filename, 'r') as file:
10         for line in file:
11             columns = line.split()
12
13             if columns and columns[0] == key:
14                 if len(columns) >= 2:
15                     return columns[1]
16                 else:
17                     return None
18     return None
19
20 def create_file_if_not_exists(filename):
21     try:
22         with open(filename, 'x') as file:
23             print(f'Arquivo {filename} created.')
24     except FileExistsError:
25         print(f'Arquivo {filename} exists.')
26
27 def check_key_exists(filename, key):
28     with open(filename, 'r') as file:
29         for line in file:
30             if line.startswith(key + ' '):
31                 return True
32     return False
33
34 def remove_lines_from_file(filename, lines_to_remove):
35     try:
36         with open(filename, 'r') as file:
37             lines = file.readlines()
38         filtered_lines = [line for line in lines if line.strip() not in
               lines_to_remove]
39         with open(filename, 'w') as file:
40             file.writelines(filtered_lines)
41     except Exception as e:
42         print(f"Error on Remove Lines: {e}")
43
44 def send_request(arquivo1, arquivo2, key_value):
45     global url
46     final_url = url +"receive_request/"+ str(key_value)
47     document1 = open(arquivo1, 'rb')
```

```
48      document2 = open(arquivo2, 'rb')
49      all_files = {'arquivoSOL': (arquivo1, document1), 'arquivoCFG':(document2.
            name, document2)}
50      response = requests.post(final_url, files=all_files)
51      if response.status_code == 200:
52          data = response.json()
53          number_request = data.get('pedidos')
54          return number_request
55
56  def get_status(numero_pedido,key):
57      global url
58      final_url = url +"get_status/"+str(numero_pedido)+"/"+str(key)
59      response = requests.get(final_url)
60      if response.status_code == 200:
61          data = response.json()
62          numero_consulta = data.get('Info')
63          return numero_consulta
64      else:
65          print(f"Error: {response.reason}")
66  if __name__ == "__main__":
67      parser = argparse.ArgumentParser(description='Description of the script:')
68      parser.add_argument('-t', '--type', type=str, help='Type of Client, can be
            send or get')
69      parser.add_argument('-s', '--sol', type=str, help='Sol File')
70      parser.add_argument('-cfg', '--config', type=str, help='Cfg File')
71      parser.add_argument('-o', '--output', type=str, help='Name Output File')
72      args = parser.parse_args()
73
74      if args.type not in ['send', 'get']:
75          print("Type argument must be 'send' or 'get'. Use -h or --help for help
                .")
76          sys.exit(1)
77
78      if args.type == 'send' and (not args.sol or not args.config):
79          print("All arguments are mandatory to send a file. Use -h ou --help for
                help.")
80          sys.exit(1)
81
82      if args.type == 'get' and not args.output:
83          print("Output argument is mandatory to get a file. Use -h ou --help for
                help.")
84          sys.exit(1)
85
86      if args.type == 'send':
87          if not os.path.isfile(args.sol):
88              print(f"Sol file '{args.sol}' not found.")
89              sys.exit(1)
```

```
90          if not os.path.isfile(args.config):
91              print(f"Config file '{args.config}' not found.")
92              sys.exit(1)
93      if not os.path.isfile("token.id"):
94          print(f"File witk key: token.id not found.")
95          sys.exit(1)
96
97      filename= 'all_id.txt'
98      scriptType = args.type
99      if scriptType == 'send' :
100         sol = args.sol
101         config = args.config
102         key_value = read_key("token.id", 'key:')
103         number_request = send_request(sol, config,key_value)
104         print(number_request)
105         if(type(number_request) != int):
106             sys.exit(1)
107         create_file_if_not_exists(filename)
108         with open(filename, 'a') as file:
109             file.write(f'{number_request}\n')
110         sys.exit(1)
111
112     if scriptType == 'get' :
113         output = args.output
114         linhas_remover = []
115         key_value = read_key("token.id", 'key:')
116         with open(filename, 'r') as file:
117             for linha in file:
118                 linha = linha.strip()
119                 number_request = linha
120                 estado = 0
121                 estado = get_status(number_request,key_value)
122                 if estado == 0 :
123                     print("Processing not started")
124                 if estado == 2 :
125                     print("Processing")
126                 if estado == 3 or estado == 4 :
127                     print("Request Finish, Files Deleted")
128                 if estado == 1 :
129                     linhas_remover.append(str(number_request))
130                     urlD = f"https://tsa.segal.ubi.pt/api/download_files/{
                            number_request}/{key_value}"
131                     response = requests.get(urlD)
132                     if response.status_code == 200:
133                       with open(f"{number_request}.tar", "wb") as file_local:
134                           file_local.write(response.content)
135                           print("Download Sucess")
```

```
                    else:
                        data = response.json()
                        erro = data.get('Error')
                        print(f"Error Download File:: {erro}")
        remove_lines_from_file(filename, linhas_remover)
```

A.7 IQQE.North.result

```
0) PowerlawApprox
Data format: MJD, Observations, Model
MJD=56748, nan, nan, nan
MJD=53534, nan, nan, nan
Filename                    : ./IQQE.North.pre.mom
Number of observations+gaps: 7396
Percentage of gaps          : 38.75

----------------
  FullCov
----------------
No Polynomial degree set, using offset + linear trend
icount=22, z=1.239982e-07
icount=42, z=4.824117e-21
   44     0.38800
  Return code IFAULT = 0

  Estimate of minimizing value X*:
  F(X*) = 7418

  Number of iterations = 44
  Number of restarts =   0

Likelihood value
--------------------
min log(L)=-7418.277
k          =8 + 1 + 1 = 10
AIC        =14856.553
BIC        =14920.738
BIC_tp     =14902.359
BIC_c      =14955.753
ln_det_I   =31.015

Noise models
------------
PowerlawApprox:
fraction   = 1.00000
```

```
sigma      = 3.86089 /yr^0.19400
d          = 0.3880 +/- 0.0090
kappa      = -0.7760 +/- 0.0180

STD of the driving noise: 1.2290
bias : 190.567 +/- 1.059 mm (at 2015/1/1, 0:0:0.000)
trend: 15.370 +/- 0.089 mm/year
cos yearly : 0.713 +/- 0.111 mm
sin yearly : 1.559 +/- 0.111 mm
Amp yearly : 1.718 +/- 0.111 mm
Pha yearly : 65.474 +/- 3.717 degrees
offset at 56748.0000 :   -5.29 +/-  1.72 mm
offset at 53534.0000 :   -7.68 +/-  1.53 mm
exp relaxation at 53534.0000 (T=  100.00) :   -3.82 +/-  1.86 mm
exp relaxation at 56748.0000 (T=  200.00) :  -13.05 +/-  2.10 mm
--> IQQE.North.out.mom
Total computing time: 42.00000 sec
```

A.8 ENU File Format

```
# sampling period 1.0
   52465.0         0.000         0.000         0.000
   52466.0        -1.808        -3.148         2.733
   52467.0        -0.850        -1.437         3.037
   52470.0        -1.257        -7.869        -6.872
   52471.0        -0.880        -2.901        -3.235
   52474.0        -0.652        -1.361        -5.416
   52478.0         1.355        -5.492        -4.477
   52481.0         3.743        -0.053        -8.513
   52482.0         0.896        -5.455        -7.842
   52484.0         2.199        -5.135        -5.531
   52485.0         3.589        -4.893        -6.898
   52487.0         4.215        -0.958       -16.158
   52489.0         1.824        -1.724       -11.016
   52490.0         0.311        -1.653        -1.877
   52496.0         2.954        -2.490        -9.050
   52497.0         1.604        -5.249        -8.438
   52499.0         2.716        -4.688        -3.456
   52502.0         1.352        -4.571        -1.985
   52503.0         2.809        -1.662        -0.458
   52504.0         3.666        -5.869        -1.747
   52508.0         4.726        -3.586         2.125
   52509.0         4.503        -3.821        -7.169
   52510.0         1.553        -2.076        -4.756
   52512.0         5.530        -6.208        -5.551
```

```
26    52513.0        3.494        -4.236        -3.097
27    52514.0        3.527        -1.629        -0.080
28    52515.0        1.623        -1.160         0.033
29    52517.0        2.140        -3.635        -0.893
30    52519.0        6.020        -3.387        -0.511
```

A.9 First 30 lines of file East.Pre.Mom

```
1  # sampling period 1.0
2  # offset 53534.000000
3  # exp 53534.000000 100
4  # offset 56748.000000
5  # exp 56748.000000 200
6  # sampling period 1
7  52465.000000   0.000000
8  52466.000000   -3.148000
9  52467.000000   -1.437000
10 52471.000000   -2.901000
11 52474.000000   -1.361000
12 52478.000000   -5.492000
13 52481.000000   -0.053000
14 52482.000000   -5.455000
15 52484.000000   -5.135000
16 52485.000000   -4.893000
17 52487.000000   -0.958000
18 52489.000000   -1.724000
19 52490.000000   -1.653000
20 52496.000000   -2.490000
21 52497.000000   -5.249000
22 52499.000000   -4.688000
23 52502.000000   -4.571000
24 52503.000000   -1.662000
25 52504.000000   -5.869000
26 52508.000000   -3.586000
27 52509.000000   -3.821000
28 52510.000000   -2.076000
29 52512.000000   -6.208000
30 52513.000000   -4.236000
```

A.10 First 30 lines of file East.Out.Mom

```
1  # sampling period 1
```

```
# offset 56748
# offset 53534
# exp 53534  100
# exp 56748  200
52465.000000   0.000000   7.519787
52466.000000  -1.808000   7.555330
52467.000000  -0.850000   7.590832
52470.000000  -1.257000   7.697149
52471.000000  -0.880000   7.732548
52474.000000  -0.652000   7.838715
52478.000000   1.355000   7.980452
52481.000000   3.743000   8.087120
52482.000000   0.896000   8.122781
52484.000000   2.199000   8.194299
52485.000000   3.589000   8.230169
52487.000000   4.215000   8.302163
52489.000000   1.824000   8.374539
52490.000000   0.311000   8.410887
52496.000000   2.954000   8.631598
52497.000000   1.604000   8.668878
52499.000000   2.716000   8.743919
52502.000000   1.352000   8.857768
52503.000000   2.809000   8.896084
52504.000000   3.666000   8.934594
52508.000000   4.726000   9.090676
52509.000000   4.503000   9.130235
52510.000000   1.553000   9.170020
52512.000000   5.530000   9.250291
52513.000000   3.494000   9.290787
```