UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

New real-time GNSS algorithms for the detection and measurement of potential geoeffective stellar flares

Bachelor Degree in Informatics Engineering

Author
David Moreno Borràs
Specialization in Computing

Supervisor
Manuel Hernández-Pajares
Department of Mathematics

Ponent
Karina Gibert
Department of Statistics and Operations Research

July 05, 2019

Abstract

Stellar flares are sudden electromagnetic emissions on a star's surface large amounts of energy. These flares are detected by telescopes such as Swift or Fermi by performing radiation observations from low Earth orbit. However, the radiation also has an effect on Earth's ionosphere electron content. Another approach for detecting these events is possible: the aforementioned electron content variation can be processed using data from Global Navigation Satellite Systems (GNSS) such as GPS to study the flares.

The *Blind GNSS Search of Extraterrestrial EUV Sources* (BGSEES) algorithm for detecting solar flares without knowing the location of the source, that is, the position of the Sun relative to Earth, is presented, as well as a study on the feasibility of the detection of such events for the challenging scenario of far-away stars, aiming to find an alternative detection method to that of the telescopes and using free, open access data.

Keywords: Solar flares, Stellar flares, GNSS, GPS, IGS, GRB

Resumen

Las fulguraciones estelares son incrementos repentinos de radiaciones electromagnéticas en la superficie de una estrella que contienen gran cantidad de energía. Son detectadas por telescopios tales como Swift o Fermi realizando observaciones desde satélites artificiales en órbita baja terrestre. Esta radiación también afecta al contenido total de electrones de la ionosfera terrestre. Esto permite una alternativa para detectar las fulguraciones: detectar el incremento repentino y según un cierto patrón espacial característico del mencionado contenido de electrones, que puede ser procesado usando las medidas transionosféricas en doble frecuencia de los Sistemas de Navegación Global por Satélite (Global Navigation Satellite Systems, GNSS), como por ejemplo GPS.

Se propone el algoritmo *Blind GNSS Search of Extraterrestrial EUV Sources* (BGSEES) para detectar fulguraciones solares sin conocimiento previo de la posición de la fuente, es decir, la posición del Sol relativa a la de la Tierra en la caracterización del algoritmo ante fulguraciones solares conocidas, así como un estudio sobre la viabilidad de la detección de fulguraciones que tienen lugar en estrellas lejanas. El objetivo final es el de confirmar la validez de un nuevo metodo de detección alternativo al de los telescopios que usa datos de libre acceso y es omnidireccional.

Keywords: Solar flares, Stellar flares, GNSS, GPS, IGS, GRB

Resum

Les fulguracions estel·lars són increments de radiacions electromagnètica en la superfície d'una estrella que contenen una gran quantitat d'energia. Són detectades per telescopis com Swift o Fermi realitzant observacions des de satèl·lits artificials en òrbita baixa terrestre. La radiació també afecta el contingut total d'electrons de la ionosfera terrestre. Això permet una alternativa per detectar les mencionades fulguracions: detectar l'increment i seguint un patró espacial característic del contingut d'electrons, que pot ser processat utilitzant les mesures transionosfèriques en doble freqüència dels Sistemes de Navegació Global per Satèl·lit (Global Navigation Satellite Systems, GNSS), com per exemple GPS.

Es proposa l'algoritme *Blind GNSS Search of Extraterrestrial EUV Sources* (BGSEES) per detectar fulguracions solars sense cap coneixement previ de la posició de la font, és a dir, la posició del Sol relativa a la de la Terra en la caracterització de l'algoritme davant de fulguracions solars conegudes, així com un estudi sobre la viabilitat de la detecció de fulguracions que tenen lloc a estrelles llunyanes. L'objectiu final és el de confimar la validesa d'un nou mètode de detecció alternatiu al dels telescopis que és omnidireccional i fa servir dades de lliure accés.

Keywords: Solar flares, Stellar flares, GNSS, GPS, IGS, GRB

Contents

Listings

List of Figures

Chapter 1

Project Management

1.1 Introduction

Stellar flares are sudden electromagnetic emissions on a star's surface that release large amounts of magnetic energy. For the case of solar flares (originating from the Sun), these flares emit radiation that has an effect on Earth's ionosphere electron content and therefore the many satellites orbiting it. [9]

Several NASA missions that aim to detect these and other flares from far-away stars exist, like the Swift and Fermi missions, these satellites, however, perform this by using their instruments to study the gamma-ray, x-ray and ultraviolet radiation bands. [6]

The aforementioned ionosphere electron content variation, however, makes it possible to detect these flares with a more indirect approach: using data from the satellites belonging to global positioning systems.

As the sudden increase of electron content in the ionosphere has an effect on the signals these satellites receive and send, this data can be used to detect flares by using the appropriate algorithms. Parameters such as the angle between the Sun and the zenith of the Earth, or the Total Electron Content (TEC) in the air have to be taken into consideration for them to work. This is already feasible with flares that have the Sun as a source, so our goal is to, first of all, expand on that by detecting these flares without knowing the location of the Sun, and then apply that to study if it is possible to detect flares from far-away stars by developing the appropriate algorithms.

Therefore, the main objective of our project is to present an algorithm: *Blind GNSS Search of Extraterrestrial EUV Sources* (BGSEES), able to detect Solar flares without taking into consideration the position of the Sun, so that we can later adapt it to the challenging scenario of flares from far-away stars. Finally, it could be extended to real-time detection.

The aim of this chapter is to give a detailed description of the project, its scope

and context, as well as the planning for the different objectives of the project and its impact: environmental, social and economic.

1.2 Scope of the project

Now that we have defined the problem that we want to solve, we proceed to define the scope of our project: how are we going to tackle it, and what could be some obstacles that might arise during its development.

1.2.1 Objectives

For a possible progression we could present the objectives of the project as follows:

- To understand how the already existing algorithms for solar flare detection work, and see how we can apply them to the challenging scenario of far-away stars.
- Be able to work with GNSS data in order to use it as the input for our algorithms and compare that to flares registered by satellites like Swift or Fermi.
- Using this data, developing new algorithms that can perform the detection using solar flares first but without knowing the origin of the source, that is, not only detecting the solar flare, but the position of the Sun relative to the Earth.
- Applying this to the challenging scenario of far-away stars without knowing the position of the potential ionizing source.
- Prepare these algorithms to be applied for real-time data.

1.2.2 Scope

We need to study the impact of stellar flares on the Earth's ionosphere by adapting the already existing algorithms that work with the Sun, but without knowing the source of the flare to see if we can apply this method to the scenario of far-away stars.

And finally, if possible, using the result to adapt the solution to run in real-time.

1.2.3 Methodology and rigor

The project has been planned to assure that it is developed in a bottom-up style, from less to most challenging objectives because every step of the development relies on the previous one to work. Therefore, as we have specified before, the project is going to start by working with a less-challenging scenario: the Sun.

The objective is to develop an algorithm that is able to detect solar flares without knowing their location, so we can later extend it to far-away stars. Before starting with this algorithm, a first approach will be done knowing the location of the source (the location of the Sun relative to the Earth in the moment of a recorded flare) so we can assure that the computations are correct.

Parallel to this, the feasibility of detecting far-away flares will be studied, using the currently existing algorithms [9] but applied to recorded stellar flares, rather than Solar. If it is possible, we will adapt the previously developed algorithm so that it works not only with the Sun, but far-away stars (of which we don't know the location).

Finally, if any of the two objectives is successful, the algorithms will be adapted to run in real time: instead of just testing them with previous data and checked against recorded events, they will be run in real time using the latest available GNSS data.

Development tools

Git and GitHub are going to be the tools used for version control and code maintenance.

The platform we will be developing on will be Linux, and regarding programming languages, C-Shell will be used for scripts, AWK for the pre-processing of data and Fortran for the main algorithms.

Fortran was used for computing the relation between the Sun's angle with the Earth's zenith and the VTEC given by the data of a satellite, but a new part of our algorithm that didn't have to be considered with the previous ones is how to traverse the set of GNSS satellites of the whole globe and decide which ones should compute this relation, that is, efficiently consider which satellites could possibly lead us to results, instead of checking that for all of them with a brute force algorithm.

We will consider if there's any alternative that may bring us more benefits than using Fortran for this part.

Others tools might be used in the process, for example we could use something like Python to scrap the website of the Swift satellite for the data we want or download it and process it with C-shell or Bash.

Regarding the GNSS data we are going to be using, this is made available by the International GNSS Service (IGS). This voluntary federation offers open access GNSS data that can be used to obtain ionospheric information. [10]

Progress monitoring

In order to track the progress and comment on the results, a weekly meeting with the project director is organized, where we set several "Action Items" to be done during the week prior to our next meeting. Additionally, communication via email is also a possibility for any problems that might arise during the week.

Validation of the results

Data from the Swift or Fermi missions is going to be used for the validation of our results. Using past GNSS data, we will develop algorithms that use it to detect flares and the location of the source.

Swift and Fermi data will have information about the flares, so we will compare our results to see if we really detected a flare. This will also be done for the scenario of far-away stars.

Regarding the last phase, the algorithms running in real time, the only way to validate the results is to wait for any matching data from the Swift and Fermi databases.

1.2.4 Obstacles and risks of the project

Although we have stated the objectives that we aim to follow in our project, its development may be hampered by some common problems that appear when developing software and algorithms, and others that may arise due to the nature of our problem.

Understanding of the problem

The problem has a considerable physics background that I, as a Computer Science student, lack the knowledge to completely understand it. Although a basic knowledge in the field will suffice for developing the algorithms, not having a background in physics might lead to confusion at some point.

Unfeasibility of the solution

The problem that we want to solve is clear: detecting stellar flares from far-away stars. This has been studied for some cases [17], concluding that we face the possibility that the proposed solution is not totally feasible due to the nature of the problem: flares from far-away stars will not have an impact on the ionosphere as noticeable as the one from the sun, so it may be difficult, or even impossible, for them to be detected in some cases.

Interferences with the Sun

With the Sun, only the daylight hemisphere is studied for the detection of flares using GPS data (it is the only one flares' effects can reach)

In our case we don't have a fixed source, but rather aim to find it. Flares could be having an effect on any part of the ionosphere so it may not be possible to study their effect on the daylight hemisphere because of the Sun's (presumably) higher effect on it.

Because of this factor we might have to focus only on the night hemisphere and we would be missing on possible flares.

Understanding previous algorithms

It is often difficult to understand code that has not been written by oneself, let alone understanding complex algorithms without any previous knowledge. This could be another possible obstacle, as the study of the previously developed algorithms will play an important role in the development of ours.

Bugs

As we will be writing code it is clearly possible that we face problems with bugs that may appear in the process.

Computational power

Taking into consideration we may be dealing with large volumes of data, its processing may be another challenge for the project, we will have to find efficient ways to do so and think about which strategies will work best in our algorithms.

1.3 Contextualization

In this section we aim to give a brief description of the area of interest of the project and present which actors are going to be involved in its development.

1.3.1 Areas of interest

The problem has a clear background in the field of **physics** and **astronomy**. I wanted to work on a project related to astronomy to see how computer science could be applied to this field.

Although there's an important theory part behind, the weight of the project lies in developing the **algorithms**.

On the other hand, the **study of large sets of data** is another area of interest of the project, and how we can use all the GNSS data efficiently in order to generate new information.

If successful, it could also be expanded with other interesting fields like AI or Machine Learning to aid in the detection or classification of these flares, for example.

1.3.2 Stakeholders

Developers

The project is being developed by myself, David Moreno Borràs. Computer Science student at the Barcelona School Of Informatics (FIB).

I will be writing the documentation and working on the project but as I lack the knowledge of the more theoretical part of the project, the director, Manuel Hernández-Pajares, will aid me with these aspects, as this is his area of expertise.

Directors

The director of the project is Manuel Hernández-Pajares, professor from the department of Applied Mathematics at the Technical University of Catalonia (UPC).

He has conducted several studies related to the field of this project (ionospheric sounding and GNSS navigation) and is the creator of the already existing algorithms used for detecting solar flares, so he can assist me when understanding how they work and how they can be used to develop the new ones.

Benefited Actors

The mainly benefited actors would be astronomers because this technique would allow to use GPS as an astronomical instrument for the measurement of the Sun's EUV radiaton.

This would be a ground system with zero cost to detect stellar flares by using free, publicly available GNSS data.

1.3.3 State of the art

In this section we will discuss the situation of the project regarding previous studies on the field, and what does it aim to extend upon.

Far-away stars

Detection of flares by far-away stars and Gamma-Ray Bursts, powerful explosions caused by supernovas (the result of a dying star) are studied by the Swift or Fermi missions [6], for example. As will be seen later in the sustainability section, our project, if successful, would be presenting an alternative to these telescopes with less complex technology.

A first-study of this topic was done in the Master Thesis *"First study on the feasibility of Stellar Flares detection with GPS"* [17] written by David Martinez Cid and also directed by Manuel Hernández-Pajares.

The project concluded by stating that more stellar flares should be studied in order to determine whether the solar flare detection algorithms are able to detect stellar flares, which is what we intend to do with new algorithms that don't rely on the location of the source.

It is a challenging scenario due to the location of the source, as it will not have the same impact on Earth as flares from the Sun, but considering more powerful stars exist, their effects might be able to reach Earth.

Solar Flares

As mentioned before, the project director, Manuel Hernández-Pajares has conducted several studies on this topic and presented different solutions as can be seen in *"GNSS measurement of EUV photons flux rate during strong and mid solar flares"* [9] where a detailed explanation of the case is presented and *"GPS as a solar observational instrument: Real-time estimation of EUV photons flux rate during strong, medium, and weak solar flares"* [22], in collaboration with the Indian Institute of Technology.

In both of these papers the use of GPS measurements is presented as an accurate Solar observational tool using the GNSS solar flare activity indicator (GSFLAI) algorithm.

The project would be expanding on this topic by presenting a solution that does not consider the source of the flare, that is, detecting it without knowing the position of the Sun relative to the Earth. This would be a tool able to determine the position of the Sun and the event of a solar flare without a dedicated satellite, using only free, open-source data.

1.4 Planning and scheduling

The aim of this section is to present the tasks and stages of the project and how are they going to be planned and scheduled so the project meets its objectives within the given deadlines.

Because we rely on some stages of the project to work before we begin to tackle others, this planning might be updated as the project progresses, perhaps because a part took longer to develop than expected or (luckily) less.

1.4.1 Task description

In this section a description of each task is presented in a similar order to that which will be seen later in the planning section.

1 Introduction to the problem

Study past research papers related to the problem to gain some background on the project, this includes the following topics:

- The use of GNSS data as a solar flare meter. Some research papers about this topic were discussed in the State of the Art section, most of them written by the director, Manuel Hernández-Pajares. [9]
- Solar and stellar bursts and their effect on Earth's ionosphere. The aforementioned papers give a brief introduction to the topic, although others can be found that cover the topic with more depth. [18]

2 GEP

The GEP course is done early in the project to help with the documentation of the thesis, understanding the scope and context of the project, and its planning. Its different stages are:

- Context and Scope of the project
- Project planning
- Budget and sustainability

This also involves a final deliverable that includes the previously listed parts but taking into consideration the feedback of the professors of the course.

Finally, an oral presentation will be done describing the work done during this course, which will also be a starting point of the final presentation of the thesis.

3 Feasibility of the detection of flares from far-away stars

Before developing the algorithms to be able to perform this without knowing the source of the flare, a study should be done with the already existing algorithms. This is one of the most challenging problems of the project as it is not clear yet if this is possible.

To do this, we will use open-source data from missions like Swift or Fermi (satellites that are able to detect flares or burst) and see if there is any correlation between that data and the results given by the algorithms that detect solar flares.

4 Detection of solar flares with no information about the location of the Sun

This task will be done in parallel to the previous one. The current algorithms are able to, knowing the position of the Sun, study if any flares have had an effect on the ionosphere, detectable by the satellites belonging to GNSS.

We aim to do the same, but assuming we don't have any information about the position of the Sun relative to the Earth. This is an important task for the project, because it will be necessary when expanding it to far-away stars, of which we ignore the location.

5 Detection of stellar flares in real-time

If the previous systems work, instead of studying flares using past GNSS data and checking that the results match detections by satellites like Swift or Fermi, we will use the latest available data to detect them in real-time, without knowing the location of the ionizing source. For this to work the detection of far-away stars using this system and the detection of solar flares without knowing the location of the Sun have to work properly.

6 Writing the report

This task will be done in parallel to the rest. As we perform the other tasks a memory of the project will be written giving a detailed explanation of all the phases, the methodology and development of the solutions, problems or obstacles that might have appeared, and the final results of each of them.

7 Final presentation

The final task, once the report of the project is finished, is to prepare an oral presentation for the defense of the thesis. This will try to cover all the progress of the work done during the last months as concisely as possible, presenting the results, obstacles that may have appeared and solutions presented for the problems.

1.4.2 Time table

Table 1 shows the estimation of the amount of hours that will have to be dedicated for the completion of each task. The expected amount of dedicated hours to the project is 18 ECTS x 30h/ECTS = 540 hours, of which 3x30 = 90 hours are dedicated to the GEP course.

Task	Dedication Time (hours)
Introduction	20
GEP	90
Study of flares from far-away stars	120
Detection of solar flares	120
Detection in real-time	100
Writing report	90
Final presentation	4
Total	544

Table 1.1: Dedication time to each of the tasks

Taking into consideration the project will span 14 weeks, a weekly dedication of 34 hours is possible for the project planning to work as scheduled.

1.4.3 Scheduling: Gantt chart

Having started mid-February, the development of the project will span between 4 or 5 months as the oral presentations are scheduled for the first week of July.

The report should be handed in one week prior to the lectures, so for the planning, our objective is to finish the project a week before: Friday, 21th of June, so that there is enough time to revise it and make any convenient changes.

To visually represent the schedule of the planning, a Gantt chart is shown below in Figure 1. This chart has been generated using the online tool teamgantt.com:

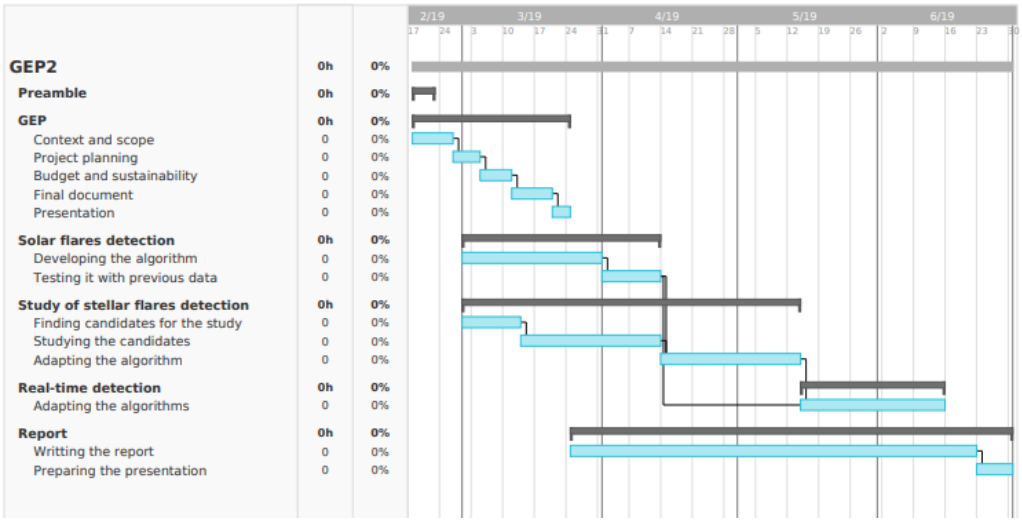


Figure 1.1: Gantt chart with the planning of the project

1.4.4 Action plan

Our idea is to work in the presented order as planned in the the previous sections, some tasks depend on previous ones to be finished to continue, and others are going to be done in parallel, like writing the report.

The project was been scheduled as previously presented, but if the dedication for some of the tasks is more than expected, the schedule should be modified as the project progresses to adapt to these situations. Another factor that might have an effect on our scheduling are some problems that might appear during the development of the project, this may cause delays and that will force us to reschedule the project planning, some of which are presented below.

As seen in the methodology section, a weekly meeting with the supervisor is held, so if any delays have appeared during the week, the schedule can be changed accordingly.

Understanding of the problem

The problem has a considerable physics background that I, as a Computer Science student, lack the knowledge to completely understand it. Although a basic knowledge in the field will suffice for developing the algorithms, not having a background in physics might lead to confusion at some point, and will make it more difficult to understand what does the algorithm have to exactly do.

Understanding previous algorithms

It is often difficult to understand code that has not been written by oneself, let alone understanding complex algorithms without any previous knowledge. This could be another possible obstacle, as the study of the previously developed algorithms will play an important role in the development of ours.

Bugs

As we will be writing code it is clearly possible that we face problems with bugs that may appear in the process.

1.4.5 Resources

Several tools that will be studied in detail in the following section, along with the cost they imply, will be needed for the development of the project. These resources can be classified in three major groups:

Software resources

Many software tools will be needed for the development of the project, although all of them will be free and open-source. All the used tools are listed in the next section, although some of the main ones are Git, which will be used for version control, everything will be running on Linux and LaTeX will be used for the reports.

Hardware resources

In this case only a computer will be needed, we will be relying on data that has been obtained using far more complex technologies (all the satellites and hardware involved in GNSS) but a computer and its peripherals will be the only hardware used during the project.

Human resources

One person will be developing the project and will have three roles: the project manager (time management and writing the report), software developer (developing the necessary algorithms) and tester (testing said algorithms).

1.5 Cost estimation

In the following sections, an estimation of the cost is presented. These are going to be divided in four major sections: hardware, software, human resources and indirect costs. Because some of the tasks will use the same resources, they have

been grouped, but those that use different resources will be studied in a different section.

1.5.1 Software resources

Common resources

Product	Units	Price	Useful life (years)	Amortization
Ubuntu 18.04	1	0 €	-	0 €
Google Chrome	1	0 €	-	0 €
Evince	1	0 €	-	0 €
Total		0 €		0 €

Table 1.2: Software costs

Developing the algorithms

Product	Units	Price	Useful life (years)	Amortization
Git	1	0 €	-	0 €
GitHub	1	0 €	-	0 €
Sublime Text 3	1	0 €	-	0 €
Python	1	0 €	-	0 €
GNSS Data	1	0 €	-	0 €
GFortran	1	0 €	-	0 €
Total		0 €		0 €

Table 1.3: Software costs

GEP and writing the report

1.5.2 Hardware resources

The following table contains the costs of the hardware that is going to be used for the project. These resources are common to all phases.

Product	Units	Price	Useful life (years)	Amortization
LibreOffice	1	0 €	-	0 €
LaTeX	1	0 €	-	0 €
TeamGantt	1	0 €	-	0 €
Total		0 €		0 €

Table 1.4: Software costs

Product	Units	Price	Useful life (years)	Amortization
Asus X555L	1	750 €	6	60 €
PC devices	1	200 €	6	20 €
Total		950 €		80 €

Table 1.5: Hardware costs

1.5.3 Human resources

The project is going to be developed by one person, which will have to be the project manager, software developer and tester.

We estimated in the planning section a total dedication time for the project of 550 hours, so here we present an estimation of the distribution of those hours between the roles and the cost of each.

Role	€/hour	Hours	Cost
Project manager	45	100	4500
Software developer	40	300	12000
Tester	30	150	4500
Total		550	21000

Table 1.6: Human resources costs

1.5.4 Indirect costs

Indirect costs of elements that will be needed in order to use the previous hardware are shown in table 6:

We can estimate the energy expenditure during the project assuming the computer consumes an average of 200 watts per hour. If we plan to use it during the 550 hours of the project, we cam estimate a total of 110 kW spent.

Product	Use	Price	Estimated cost
ADSL	4 months	40 €/month	160 €
Electricity	110 kWh	0.1067 €/kWh	11.7 €
Total			172 €

Table 1.7: Indirect costs

1.5.5 Budget per task

In the following table we can see an estimation of the total cost of the project distributed among the tasks presented in the planning section, according to the dedication time of each of these tasks and the total cost of the project:

Task	Estimated cost
Introduction to the problem	1106 €
GEP	4424 €
Feasibility of the detection of flares from far-away stars	4424 €
Detection of solar flares with no information about the location of the Sun	4424 €
Detection of stellar flares in real-time	3318 €
Writing the report and final presentation	4424 €
Total	22122 €

Table 1.8: Budget per task

1.5.6 Total budget

In the following table, the total cost of the project can be seen, estimated using data seen in the previous tables.

As we can see, there is no software cost because only open-source or free tools have been used.

Resource	Estimated cost
Software	0 €
Hardware	950 €
Human resources	21000 €
Indirect costs	172 €
Total	22122 €

Table 1.9: Total cost of the project

1.5.7 Budget control

As seen in the planning section, some of the tasks may take longer than estimated because of unexpected difficulties, which would in turn increase the total cost of the project. So we have to consider the fact that these delays could lead to an increase in the total cost of the project.

Weekly meetings are held to check that everything is going as scheduled, so if any problem appears we can try to reschedule the planning of the project and avoid as much extra costs as possible.

Although unlikely, hardware faults might occur that would require more resources, but the main factor that might influence the budget during the project is time, which would increase the amount of work hours done by either the project manager, the software developer or the tester.

1.6 Sustainability

The form presented by EDINSOST has helped me reflect on how, although in some courses during the degree the relation between Computer Science (or engineering in general) and sustainability has been studied, as engineers, we don't usually consider these factors on our own, such as the environmental or social impact. The economical impact is something usually considered, specially by project managers, but seldom is the effect of the technology on the environment taken into consideration.

I have realized how in many of the projects I collaborate, I do not usually stop to think about the impact they are having, for example, on the environment, and how

I have no experience in these fields, specially in the economic management part. Therefore, I hope I can gain some experience by studying these aspects more in depth and the impact of the project in them.

In this section we will focus on evaluating the impact of our project by studying its sustainability in three different aspects: environmental, economical and social.

The analysis is going to be based on the application of the following sustainability matrix which is scored in a [0,10] range and then will study each of the three main aspects:

	PPP	**Exploitation**	**Risks**
Environmental	(2) Design consumption	(2) Ecological footprint	(2) Environmental risks
Economic	(4) Resources needed	(2) Cost	(7) Human resources
Social	(9) High personal impact	(5) Medium social impact	(2) Low social risks

Table 1.10: Sustainability matrix

1.6.1 Environmental sustainability

During the project we are going to use the minimum amount of resources possible, which have been presented in the Cost estimation section. Because what we are going to use is mainly software, the resource from the project which will have an environmental impact is going to be the energy spent by the devices running during the project (the computer).

Furthermore, if the project is successful, it would present an alternative to currently working satellites that detect Gamma-Ray Bursts (GRB), like the Gamma-ray Large Area Space Telescope (GLAST) or weather satellites like the Geostationary Operational Environmental Satellite (GOES).

As seen in its specification manual (https://www.nasa.gov/pdf/221503main_GLAST-041508.pdf) GLAST needs about 1500 watts average over an orbit, which is significantly more than the consumption of the computer that we have estimated before: 110 kWh. The telescope, however, is equipped with solar panels that can supply up to 3122 watts in sunlight.

While our alternative would not obtain results with the precision and information that these missions aim to achieve, some results would be similar, so we can also

consider the difference in environmental impact between both.

The larger environmental impact of the GLAST mission, however, lies in the design, build and launch of the telescope. While information about the cost of the previous factors is available and will be studied in the next section, there is no information provided regarding its environmental impact, although we can say that it likely has a considerably larger one than that of our project, in which only a computer is used.

In conclusion, the project's resources are mainly software and the factor with the biggest environmental impact will be the energy expenditure of the computer, which is significantly lower than that of the currently existing alternatives.

1.6.2 Economic sustainability

In previous sections we have studied the cost of our project (hardware, software and human resources). From an economical point of view, our project presents an alternative to telescopes like GLAST or GOES. Albeit less precise and equipped, some of its aspects and purposes are shared.

The cost to design, build and launch GLAST, for example, had a total international contribution of 690 US dollars. Considering our project relies only on free or open-source data and software, it would be offering an alternative with a lower economical impact.

It would be difficult to do this project with a lower cost, considering the only resource that has an economical impact besides the human work is the hardware (a computer). It would be difficult to lower the costs of this area considering a computer is needed for most computer science projects.

1.6.3 Social sustainability

Personally, the project is very relevant to me. I wanted to work on a project to see how computer science could be applied to a field like astronomy or physics. I think the project and algorithms we are developing are a good example of the place CS has in this fields and the role it plays.

It has also helped me gaining experience in terms of information retrieval and research. Both writing reports like this one, planning projects and researching information from reputable sources that can be used in our project.

If the project is successful, it could turn into a useful tool for astronomers that could be used as an astronomical instrument to measure the Sun's EUV using only open-source GPS data, rather than a dedicated telescope, which would be a useful, less expensive alternative.

Chapter 2

Background

As the project requires a certain background in physics and astronomy, some of the relevant topics that are going to be studied are introduced: Global Navigation Satellite Systems, the Ionosphere, Stellar Flares and Gamma-Ray Bursts.

2.1 Global Navigation Satellite Systems

GNSS and GPS

These two terms may lead to confusion as Global Navigation Satellite Systems (GNSS) is the generic term for all satellite navigation systems. The Global Positioning System (GPS), in particular, is the United States' GNSS system, the world's most used GNSS. Other systems, for example, are the European Galileo or Russian GLONASS [8].

Global Navigation Satellite Systems use satellites to determine the clock error and the position of a given receiver or device in terms of, for instance, latitude, longitude and height.

Positioning

In short, GNSS works as follows: out of all the GNSS satellites orbiting Earth, at least four of them are constantly visible from a specific point and transmitting information at two or more frequencies. When a device receives a signal from one of them, the distance to the satellite can be calculated by means of the speed of light and the time required to reach it. As many variables might affect the speed of light such as the medium through which it is propagating, this estimation of the distance is called **pseudo-range**.

Thus, the location of the receiver can be estimated using a technique which can be understood through its simplified version: **trilateration**. Having three spheres

around each of the satellites with the range as their radius, the intersection of these spheres yields the location of the receiver [12].

The International GNSS Service

In 1998 the International GNSS Service (IGS) was created as a collaboration of several members of the scientific community: Center for Orbit Determination in Europe (CODE), (European Space Agency) ESA, Jet Propulsion Laboratory (JPL) and Polytechnic University of Catalonia (UPC). This voluntary federation has made available open access GNSS data since its creation [2] [3].

Its data is provided by more than 300 GPS receivers around the globe and is processed by the previous institutions which compute the global distribution of the Vertical Total Electron Content (TEC) [10].

GNSS is a key component to this project because, as mentioned before, many variables can affect the speed of light and therefore the time it takes for the transmitter's signal to be intercepted by the receiver. One of these variables is the electron content of the layer of the atmosphere where GNSS satellites operate: the ionosphere.

2.2 Ionosphere

The ionosphere is a layer of the Earth's atmosphere that lies 75-1000km above the surface of the planet [23].

High energy from Extreme UltraViolet (EUV) and X-ray radiation can cause its atoms to be ionized and create a layer of electrons [20]. Due to these free electrons and ionized molecules, it is capable of affecting radio wave propagation, thus having an effect on Global Navigation Satellite Systems (GNSS) technology, this phenomena allows these satellites to be used as a global scanner for the ionosphere [11].

The main physical quantity used for describing the electron content of the ionosphere is the **Vertical Total Electron Content (TEC)**, the TEC is the total number of free electrons between two points $(r1, r2)$ along a cylinder of base $1m^2$. Slant TEC (STEC), in particular, can be defined as the TEC in which $r1$ and $r2$ are a satellite and a receiver's positions [22].

The unique properties of the ionosphere enable us to use the data provided by the GNSS technology to study stellar flares, a powerful phenomena that occurs in many stars across the universe.

Ionospheric Pierce Points

Ionospheric Pierce Points (IPP) are going to be very relevant throughout the development of the project. These will be the locations that we are going to use for our measurements, not those of satellites or receivers. A Ionospheric Pierce Point is the

point where the line between the satellite and a receiver intersect at the ionosphere's effective height, where the Vertical Total Electron Content can be estimated. [4]

2.3 Stellar flares

Flares from stars, in particular those that have the Sun as a source (more noticeable due to its proximity) are sudden flashes of brightness in the surface of stars which release large amounts of energy across the whole electromagnetic spectrum[1]. Flares that have the Sun as a source can increase the electron content of the ionosphere and have an effect on waves passing through it, affecting satellite communications and causing a delay. This phenomena is the key element of the project, as it enables us to study these events.

Satellites can also be harmed by this effects: the flares heat up the outer atmosphere, which in turn increases the drag on these satellites reducing their lifetime in orbit [9].

The previous phenomena applies to flares originating from the Sun, whether a flare that has a star from outside the Solar System as a source has an effect on the Earth's atmosphere or not is one of the topics that is going to be studied in this project, as the distance may reduce their effect on the Earth.

2.4 Gamma-Ray Bursts

Throughout the project, in particular when studying the feasibility of stellar flares detection, another type of event will be mentioned and studied as well: Gamma-Ray Bursts (GRBs):

GRBs are highly energetic explosions that occur in distant galaxies, releasing large amounts of radiation, in particular Gamma rays, hence the name of the event. These bursts, despite being millions of light years away from Earth are so powerful they might still have an impact on the ionosphere, like the aforementioned stellar flares. The main difference with flares originating from stars is that GRBs are thought to be originated from the death of massive stars, that is, supernovas.

Despite not being a stellar flare, the phenomena we aim to detect, this event is going to be studied in the following section to test the currently working algorithms, mainly for two reasons: it has been studied and cataloged by telescopes such as the Fermi Observatory and there's is available information that we can use for our study, and the large amounts of energy they emit it could mean GRBs are a more feasible target to detect.

[1] X-rays and Extreme Ultraviolet (EUV) radiation

Chapter 3

Solar flare detection

Before developing the main solar flare detection algorithm a first study is presented targeting a powerful solar flare for which we knew the time of the event and therefore, the location of the Sun.

Although the aim of the main algorithm is detecting the solar flare without taking into consideration the position of the Sun, this study was done to understand how the core of the algorithm works: studying the correlation between the cosine of the solar-zenith angle and the VTEC increase.

This chapter also provides an introduction to the formatting and use of the Global Navigation Satellite Systems data (GPS in this case) and how the main parameters necessary for the algorithms are computed.

3.1 Data

3.1.1 GPS Data

As we have seen in previous sections, the International GNSS Service (IGS) has made available open access GNSS data since its creation. The Crustal Dynamics Data Information System (CDDIS) is a central data archive for the NASA's Crustal Dynamics Project (CDP), dedicated to archiving space geodesy data for research. This archive has been storing and providing access to the GNSS data generated by the IGS since 1992.

Figure 3.1a shows how data is stored in de CDDIS server (`ftp://cddis.nasa.gov/gps/data/hourly/`).

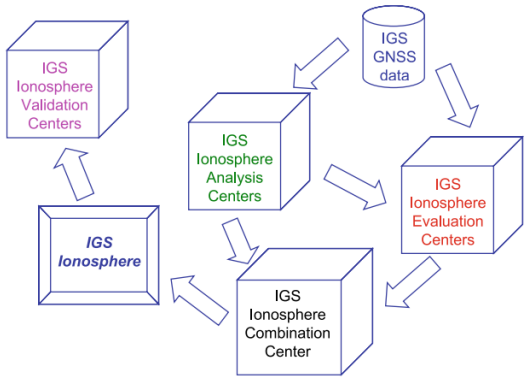
The files in this server contain raw GPS data that is then pre-processed to obtain VTEC maps in the form of **ti** files. An example diagram of this complex, several step procedure is shown in figure 3.1b, extracted from the paper *"The IGS VTEC maps: a reliable source of ionospheric information since 1998"* [10] by Manuel Hernández-Pajares, which offers a detailed explanation of this process.

Index of /gps/data/hourly/

[parent directory]

Name	Size	Date Modified
1999/		4/1/19, 6:30:00 AM
2005/		11/15/17, 1:00:00 AM
2006/		11/15/17, 1:00:00 AM
2007/		11/16/17, 1:00:00 AM
2008/		11/15/18, 12:23:00 PM
2009/		1/30/18, 1:00:00 AM
2010/		1/30/18, 1:00:00 AM
2011/		1/29/18, 1:00:00 AM
2012/		1/29/18, 1:00:00 AM
2013/		1/26/18, 1:00:00 AM
2014/		1/26/18, 1:00:00 AM
2015/		1/25/18, 1:00:00 AM
2016/		1/24/18, 1:00:00 AM
2017/		1/23/18, 1:00:00 AM
2018/		12/13/18, 1:31:00 AM
2019/		4/10/19, 2:31:00 AM
reports	0 B	8/9/17, 2:00:00 AM

(a) Files



(b) Data flow

Figure 3.1: CDDIS server (a) and data flow to obtain IGS data (b)

However, for this and the following section, only the pre-processed ti files of past dates were needed, as detecting the flares in real time is a task that will be discussed later in the project, in which this pre-processing will have to be taken into consideration. The project supervisor, Manuel Hernández-Pajares, provided some of data sets to use as input for the algorithms, along with information about the formatting of this files.

3.1.2 Formatting

The ti files contain several rows of pre-processed GPS data. Each row has a Receiver Id. and a Transmitter Id., therefore, for each row we have the Ionospheric Pierce Point between the Receiver and Transmitter. Each IPP has several parameters that are relevant for our computations:

- The **GPS time**
- The **Receiver Id.**
- The **Transmitter Id.**
- The **double derivate of Li**
- The **xmappingion**
- The **right ascension** and **declination** of the IPP

```
 1 Field number | Example value | Description
 2 [...]
 3 3      0.008333333333     GPS time/hours (tsecdayobs/3600.d0)
 4 4      cand               Receiver Id.
 5 5      3                  Transmitter Id.
 6 [...]
 7 21     -0.5131586E-02     d2li
 8 [...]
 9 43     0.1565765332E+01     xmapping_ion
10 44     334.449              xraion
11 45     33.092               xlation
12 [...]
```

Listing 3.1: Format of the ti file

The files also contain the right ascension and declination of the **Sun**, which will be used in the last chapters to study the error of the algorithm's estimations.

3.1.3 The Halloween Solar Storm: X17.2 flare

The data set we used was that of the so-called Halloween Storm, a powerful solar storm that took place from October to early November in the year 2003. In particular, we will try to replicate the results shown in figure 3.2, shown in the paper *"GNSS measurement of EUV photons flux rate during strong and mid solar flares"* for a poweful flare that took place in October 28th, 2003 [9].

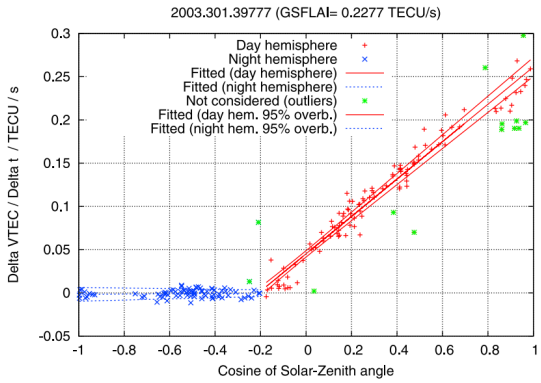


Figure 3.2: VTEC as a function of the cosine of the solar-zenith angle

As we can see the plot of the flare called X17.2 took place exactly at 2003.301.39777 (year.day.seconds of GPS time). In hours, 39777 seconds of a day is $39777s * 1h/3600s = 11.049...h$, around 11AM.

The ti files provided contained data from 10.5h to 11.5h (with a sampling rate of 30 seconds), so that we could see the VTEC distribution throughout the day.

With this data we can compute the two parameters that will yield the plot shown in figure 3.2: the **VTEC value** and the **cosine of the solar-zenith angle**.

3.2 Vertical Total Electron Content (VTEC)

Fisrt we wanted to obtain the VTEC distribution throughout the day, to visually see if any spikes appeared confirming that the moment we were going to study based on the paper was correct.

For each epoch in our data set (from 10.5 to 11.5 with a sampling rate of 30 seconds) we needed to compute an estimation of the VTEC value.

3.2.1 Computing the VTEC

As we have mentioned before, one of the main paramenters relevant to the computation is the **double derivate of LI**, the d2li field in the ti file. The Li is the "ionospheric combination of carrier phases" [9], a direct measurement of TEC. Because this is the double derivative, with the following operation we obtain **curvature of the VTEC**, this will be observed in figure 3.3.

This value can be estimated using the following operation:

$$\frac{d^2V}{dt^2} = \frac{d^2Li}{M} \tag{3.1}$$

Where $M = \frac{1}{\cos Z}$ is the "ionospheric mapping function", the inverse of the cosine of the satellite-zenith angle that we have for each IPP. [9]. This is the **xmappingion** field in the ti file.

Although the data that will be used throughout the project is going to be the VTEC curvature, it will be referenced simply as VTEC from now on for readability. This value indicates whether a point is a maximum or a minimum, so it can work as an indicator of the VTEC.

Implementation

Below is the code used to compute the VTEC value in Fortran that we will use to replicate the plot from figure 3.2.

```
1 double precision function estimateVTEC (mapIon, d2Li)
2   implicit none
3   double precision, intent(in) :: mapIon, d2Li
4   double precision :: vtec
5
6   vtec = d2Li/mapIon
7   return
8 end function estimateVTEC
```

Listing 3.2: Simple Fortran function to compute the VTEC value

3.2.2 Distribution throughout the day

Because the only operation that had to be performed was the previous division, a simple AWK script was used to filter the two necessary fields from the data file and print the resulting value as well as its time.

```
1 {
2   /a/
3   d2li = $21;
4   mappingFunc = $43;
5   vtec = d2li/mappingFunc;
6   print $3 " " vtec
7 }
```

Listing 3.3: AWK script to estimate the VTEC

```
1 #!/bin/bash
2 tiDataFile="../data/ti.2003.301.10h30m-11h30m.gz"
3
4 zcat "$tiDataFile" | gawk -f previewVTECDistribution.awk >
      vtecValues
5 gnuplot -e "set terminal png; set output 'vtecDistribution.png';
      set title 'VTEC Distribution'; set xlabel 'Time of the day (
      hours)'; set ylabel 'VTEC'; set grid; plot \"vtecValues\" using
      1:2 with point"
```

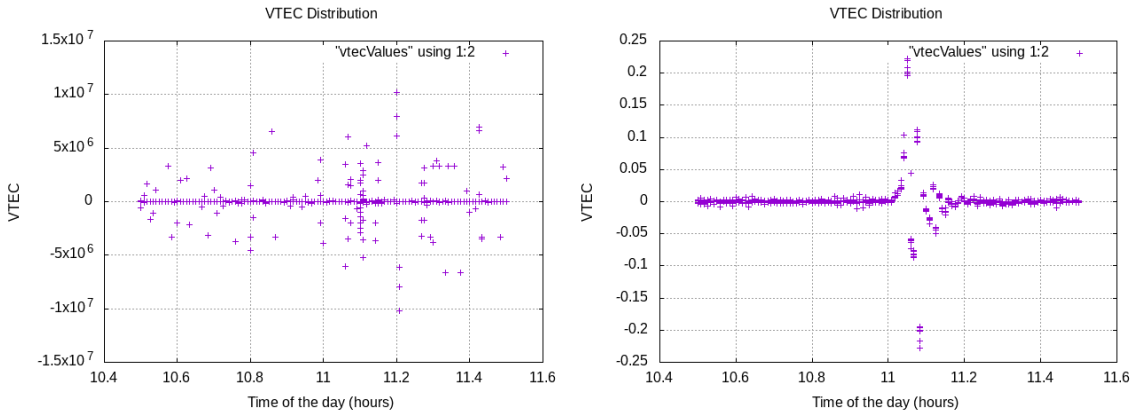
Listing 3.4: Bash script to execute the procedures

The bash script executes the AWK process with the data as the input and outputs n rows with two columns: the **time of the day** and the **calculated VTEC**, and finally plots the results using Gnuplot.

The result can be seen in figure 3.3a, where we can see how the VTEC value evolves throught the day. Visually, a spike can be seen between 11 and 11.2 hours.

To see the event more clearly, though, we can focus on one specific receiver (which will still yield multiple IPPs, as the receiver works with different satellites). For the particular case of the Villafranca, Spain station (identified as Vill in the ti files), we obtain the plot from figure 3.3(b). At this time of the day around 11:00h the Sun would have a greater effect on the IPPs of this station due to its location, so the spike can be seen more clearly.

As mentioned before, the flare took place at 11.05, so we could proceed using the studied data range and this epoch in particular.



(a) All IPPs (b) Villafranca station

Figure 3.3: VTEC distribution throughout the day for all IPPS (a) and for IPPs that have Vill as the receiver (b)

3.3 Solar-zenith angle

The solar-zenith angle (denoted χ from now onward) plays a major role when studying this event: it is the angle formed by the Sun and the Earth's zenith and indicates the effect the flare is having on a particular IPP. It is expected that this variable presents a correlation with the increase in VTEC, which is what we aim to observe in this chapter.

Figure 6.1, at the end of the chapter, provides a visual representation of this variable that along with the results depicts how it can affect the VTEC value.

Right Ascension and Declination are two concepts similar to longitude and latitude, respectively, used to describe the location of objects in the sky, in particular in a sphere of infinite radius with the Earth as its center called the **celestial sphere**.

Taking this into account, Right Ascension is the equivalent of longitude, expressed in degrees (or more commonly in hours, minutes and seconds) and Declination, the equivalent of latitude, is expressed in degrees between the two poles: +90° and -90°, this reference system is used to describe the position of objects in the sky. [24]

The angle between two celestial objects is obtained by means of equations 3.2, 3.3 and 3.4, by performing the dot product of the two unit vectors that define the position of the objects using their right ascension (α) and declination (δ).

$$unitVectorObjectA = \begin{bmatrix} \cos\delta_g * \cos\alpha_g \\ \cos\delta_g * \sin\alpha_g \\ \sin\delta_g \end{bmatrix} \tag{3.2}$$

$$unitVectorObjectB = \begin{bmatrix} \cos\delta_s * \cos\alpha_s \\ \cos\delta_s * \sin\alpha_s \\ \sin\delta_s \end{bmatrix} \tag{3.3}$$

$$\cos\beta = unitVectorObjectA \cdot unitVectorObjectB \tag{3.4}$$

For this case, the cosine of the solar-zenith angle χ is computed using the IPP's Right Ascension and Latitude (equivalent to declination). The previous dot product can be simplified to:

$$\cos\chi = \sin\delta_{IPP} * \sin\delta_{Sun} + \cos\delta_{IPP} * \cos\delta_{Sun} * \cos(\alpha_{IPP} - \alpha_{Sun}) \tag{3.5}$$

The following Fortran code is the function that implements equation 3.5 and returns $\cos\chi$:

```
1 double precision function computeAngle (raIPP, decIPP, raSun,
    decSun)
2   implicit none
3   double precision, intent(in) :: raIPP, decIPP, raSun, decSun
4   double precision :: solarZenithAngle
5
6   solarZenithAngle = sin(decIPP)*sin(decSun) + cos(decIPP)*cos(
    decSun)*cos(raIPP - raSun)
7   return
8 end function computeAngle
```

Listing 3.5: Computation of the solar-zenith a angle's cosine

3.4 Results

Taking 212.338° and −13.060° as the Sun's right ascension and declination, respectively, and the measurements of all IPPs at 11.05 hours, figure 3.4 shows the plot of the output of our program.

As we can observe, the resulting plot, similar to the one from figure 3.2, shows a strong corelation between the cosine of the solar-zenith angle and the VTEC content, which increases from $\cos\chi = 0$ (90°) to $\cos\chi = 1$ (0°) (the effect of the Sun on the IPP increases) and it does not seem to be affected from $\cos\chi = -1$ (180°) to $\cos\chi = 1$ (0°) (when the IPP is in the night hemisphere).

In conclusion, we can see that there appears to be a correlation between the two variables. This correlation will be studied in more detail in the following section,

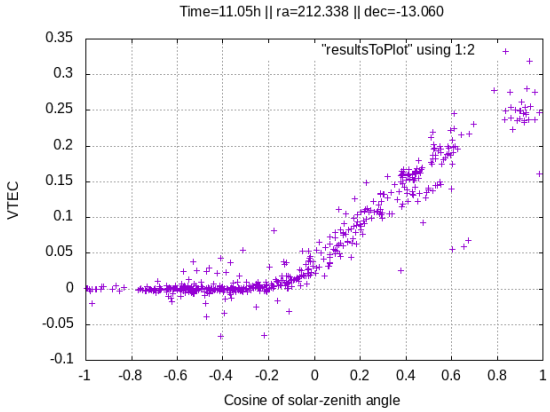


Figure 3.4: VTEC value as a function of the solar-zenith angle cosine

where a first approach of the algorithm will be presented to detect the flare without knowing the location of its source.

Chapter 4

Brute Force Approach

In the previous chapter the correlation between the solar-zenith angle's cosine and the estimated VTEC value was studied. Here, we aim to provide a first, brute force approach, of the *Blind GNSS Search of Extraterrestrial EUV Sources (BGSEES)* algorithm to estimate the location of a EUV source. This approach is done as a first approximation to the problem to see more clearly how the algorithm will work, regardless of its performance.

For this first approach the Sun is used as the source we are trying to find (to check the corectness of the solution). It considers possible Sun locations (with a certain degree of precision) and checks the fitness of each to consider which could be the real location.

4.1 Correlation

As seen in the previous chapter, there exists a correlation between the VTEC value and the cosine of the solar-zenith angle.

The aim of the algorithm is to study the correlation for each of the possible Suns locations considered and yield a fitness indicator for each of them.

The main idea behind the search is that the higher the correlation, the more accurate the estimated location should be, compared to the Sun's real location.

The results will be discussed in the last section of this chapter to see if the previous expectations are true.

Computation

The correlation between two independent variables is defined as follows:

$$r_{xy} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum(x_i - \bar{x})^2}\sqrt{\sum(y_i - \bar{y})^2}} \quad \text{for} \quad i \in (0, n) \tag{4.1}$$

Although optimization is not the aim of this section, the previous formula would require passing the data twice: first to compute the mean of the cosine and VTEC and second to compute the coefficient itself. The formula can also be expressed as follows:

$$r_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}} \quad \text{for} \quad i \in (0, n) \tag{4.2}$$

Which can be implemented with a single pass algorithm, as opposed to the former. Because of this we decided to initially start with this one.

4.2 Algorithm

The algorithm works as follows: the spike of VTEC value throughout the day is found by finding the epoch with the maximum mean VTEC of all IPPs for that epoch[1]. The data is then filtered by that epoch (only the info of IPPs for that epoch will be used for the computations) and the algorithm starts considering possible Suns. Furthermore, because the peak of the flare has a VTEC value of 0.4, a cutoff filter is performed to with a value of 0.7, so that interferences do not cause incorrect calculations.

There are $360 * 180 = 64800$ possible Sun's. Thus, in order to test the algorithm, the factor *STEP* is used which defines the step between the angles of possible Suns. A step of 1 degree would mean considering all 64800. The smaller the step, the more Suns will be considered.

For each of these possible Suns, the VTEC value and the cosine of the solar-zenith angle ($\cos\chi$) are computed for every IPP in that epoch. Each of these Suns yields a data set with the aforementioned variables that can be plotted to obtain images such as the one studied at the end of the previous section. This two variables are used for computing the correlation coefficient for every considered Sun.

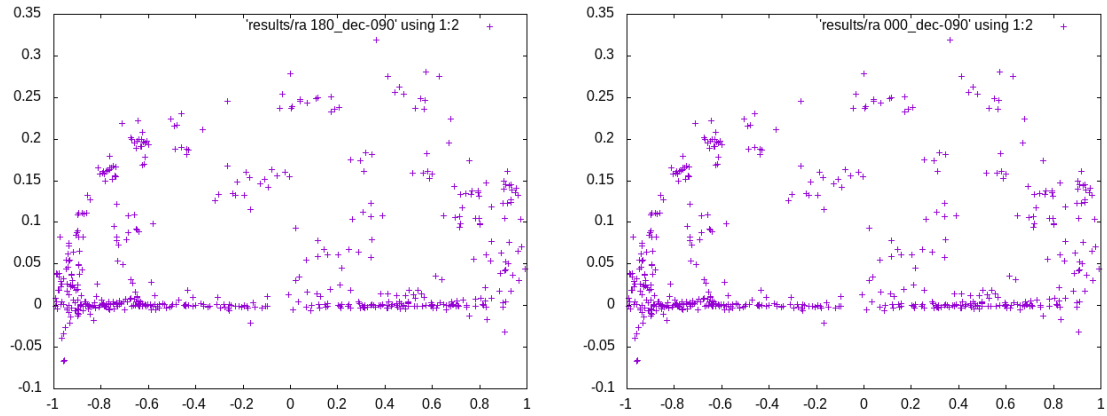
The following is the pseudocode for the brute force approach of the algorithm. Which returns the Sun that has yielded the highest correlation coefficient.

[1]The use of this method for finding the best epoch is discussed later in the chapter using the results of the algorithm

Algorithm 1 Brute Force Approach

```
1: procedure MAIN
2:     epoch ← findSpikeInData()
3:     filterDataByEpoch(epoch)
4:     bestSun ← nil
5:     for ra = 0; ra <= 360; ra+ = STEP do
6:         for dec = −90; dec <= 90; dec+ = STEP do
7:             currentSun ← computeCorrelationPossibleSun(ra, dec)
8:             if currentSun.correlation > bestSun.correlation then
9:                 bestSun ← currentSun
10: return bestSun
```

One thing that has to be taken into consideration is that all possible locations that have a declination of 90° or -90° yield the same results (figure 4.1).



(a) ra=180°, dec=-90°

(b) ra=0°, dec=-90°

Figure 4.1: Two examples of possible Sun locations

Using a *diff* tool we can observe that although the data files used to plot the results [2] have different values, the resulting plots are exactly the same image.

This is due to the fact that declinations 90° and -90° correspond to the poles, with the Sun exactly on top or below the Earth, and therefore all possible right ascensions are exactly the same position.

[2]The data is $cos\chi$ for the X axis and VTEC for the Y axis

4.2.1 Implementation

Iterating over the possible Suns

The main loop of the algorithm considers: for each possible declination, each possible right ascension (taking into consideration the special case of the poles) and uses said declination and right ascension to compute the correlation coefficient, saving the one with the highest value.

```
for (int dec = -90; dec <= 90; dec += step) {
  if (dec != -90 and dec != 90) {
    for (int ra = 0; ra <= 360; ra += step) {
      pearsonCoefficient = computeCorrelation(&ra, &dec);
      if (pearsonCoefficient > maxCoefficient) {
        maxCoefficient = pearsonCoefficient;
        location = "[" + ra + ", " + dec + "]";
      }
    }
  }
  else {
    int ra = 0;
    pearsonCoefficient = computeCorrelation(&ra, &dec);
    if (pearsonCoefficient > maxCoefficient) {
      maxCoefficient = pearsonCoefficient;
      location = "[" + ra + ", " + dec + "]";
    }
  }
}
```

Listing 4.1: Main loops

Computing the correlation

This is the main code of the Fortran function *computeCorrelation(ra, dec)* called for every possible *(ra, dec)* pair considered in the previous loop. Other auxiliary functions appear in the *computeCorrelation(ra, dec)* function such as *openFile()* (opens the file for reading) or *toRadian()* (converts the degrees from the input to radians, used by Fortran's trigonometric functions). These functions are not included for readability.

computeCorrelation(ra, dec) reads every line of the file that contains the information of the IPPs filtered by the found epoch and computes both the solar-zenith angle and the VTEC using the same procedures seen in the previous chapter:

```
1 double precision function traverseFile (raSunIn, decSunIn)
2   implicit none
3   integer, intent(in) :: raSunIn, decSunIn
4   double precision :: raIPP, decIPP, raSun, decSun, mapIon, d2Li,
     cosX, vtec
5   double precision :: sumx = 0, sumy = 0, sumxy = 0, sumx2 = 0,
     sumy2 = 0
6   double precision :: rxyPearson
7   integer :: i = 0
8
9   sumx = 0
10  sumy = 0
11  sumxy = 0
12  sumx2 = 0
13  sumy2 = 0
14  i = 0
15
16  raSun = raSunIn
17  decSun = decSunIn
18  raSun = toRadian(raSun)
19  decSun = toRadian(decSun)
20  call openFile()
21  do while (1 == 1)
22    read (1, *, end = 240) raIPP, decIPP, mapIon, d2Li
23    raIPP = toRadian(raIPP)
24    decIPP = toRadian(decIPP)
25    vtec = estimateVTEC(mapIon, d2Li)
26    cosx = computeSolarZenithAngle (raIPP, decIPP, raSun, decSun)
27    if (cosx > CORRELATION_THRESHOLD) then
28      call updateCorrelationParameters (cosx, vtec, sumx, sumy,
     sumxy, sumx2, sumy2)
29      i = i + 1
30    end if
31  end do
32  240 continue
33  close(1)
34  rxyPearson = computePearsonCoefficient(i, sumx, sumy, sumxy,
     sumx2, sumy2)
35  return
36 end function traverseFile
```

Listing 4.2: Correlation computation

As can be seen in the code, the line (IPP) is only considered if $\cos\chi$ is higher than a *"correlation threshold"* ($-0.1°$ in this case). This is done because we want to study only the "part" of the ionosphere where the Sun is having an effect. This can be seen in figure 6.1, where we observed that the VTEC value remained the same from $\cos\chi = -1$ (180°) to $\cos\chi = 1$ (0°).

Once the two variables are computed (*vtec* and *cosx*, in the code), the necessary

summations for computing the correlation are updated each iteration:

- $\sum x_i$ and $\sum y_i$
- $\sum x_i y_i$
- $\sum x_i^2$ and $\sum y_i^2$

The following code is the function called every iteration that updates this summations:

```
1 subroutine updateCorrelationParameters (x, y, sumx, sumy, sumxy,
     sumx2, sumy2)
2   implicit none
3   double precision, intent(in) :: x, y
4   double precision :: sumy, sumy2, sumxy, sumx2, sumx
5
6   sumx = sumx + x
7   sumy = sumy + y
8   sumxy = sumxy + x*y
9   sumx2 = sumx2 + x*x
10  sumy2 = sumy2 + y*y
11
12  return
13 end subroutine updateCorrelationParameters
```

Listing 4.3: Updating the summations

Finally, once all the IPPs have been processed, the previous summations are used to compute the Pearson Correlation Coefficient using equation 4.2, this is the value the function returns to the C++ code.

```
1 double precision function computePearsonCoefficient (n, sumx, sumy,
      sumxy, sumx2, sumy2)
2   implicit none
3   integer, intent(in) :: n
4   double precision, intent(in) :: sumx, sumy, sumxy, sumx2, sumy2
5   double precision :: rxyPearsonCoefficient
6
7   numerator = n*sumxy - sumx*sumy
8   denominator = sqrt(n*sumx2-sumx*sumx)*sqrt(n*sumy2-sumy*sumy)
9
10  rxyPearsonCoefficient = numerator/denominator
11  return
12 end function computePearsonCoefficient
```

Listing 4.4: Computing the correlation coefficient using the summations

Compiling

The C++ and Fortran compilers allow us to compile both languages and their libraries together. With this, the main part of the algorithm can be implemented

using C++ which can then call Fortran for the parts that require heavy numerical computation.

This can be done by compiling the object of the Fortran code using the $-c$ flag, and then linking it with the C++ code using the $-lgfortran$ flag so that the standard Fortran libraries are included:

gfortran fortranFunctions.f90 -c -o functions.o
g++ functions.o bruteForce.cc -o bruteForce.x -lgfortran

4.2.2 Mean VTEC as a reliable indicator of the moment of the flare

The first part of the algorithm was, without going into the actual computations regarding the position of the IPPs, the possible Sun locations, etc, finding out when to perform the study, that is, detecting a spike in the VTEC content throughout the provided data range. In the previous chapter we already knew the specific moment of the flare: 11.05h, and could work based on this information, but the first step of the algorithm has to determine which moment is going to be studied.

For each epoch[3] we computed the mean VTEC of all IPPs and returned the epoch which had the highest VTEC mean.

To see if the mean VTEC could be used as a reliable indicator, the algorithm was tested with all available epochs of the data set, in order to study the effect of this indicator in the resulting estimation of the source's location, before studying the one with the highest coefficient in detail.

To do this we sorted the different epochs available in our data set by their mean VTEC, inserting them into a priority queue.

The following loop traverses the data and computes the mean VTEC of each epoch, inserting it in a priority queue:

[3]In our data set the epochs ranged from 10.5 to 11.5, that is, 10:30AM to 11:30AM with a sampling rate of 1/120 hours or 30 seconds

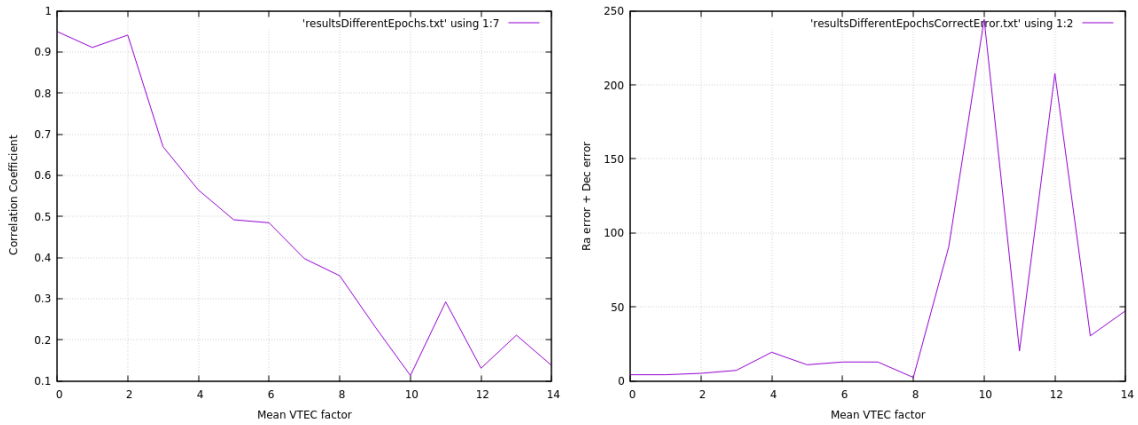
```
1   double epochIn, vtecIn, raIPPIn, latIPPIn;
2   int n = 0;
3
4   data >> epochIn >> vtecIn >> raIPPIn >> latIPPIn;
5   double totalEpochVTEC = vtecIn;
6   double previousEpoch = epochIn;
7   while (data >> epochIn >> vtecIn >> raIPPIn >> latIPPIn) {
8     totalEpochVTEC += vtecIn;
9     n++;
10    if (previousEpoch != epochIn) {
11      insertCandidate(previousEpoch, totalEpochVTEC/n);
12      previousEpoch = epochIn;
13      totalEpochVTEC = 0;
14      n = 0;
15    }
16  }
```

Listing 4.5: Finding a VTEC spike

The following table presents the 10 best epochs from best to worst and their results. Figure 4.2 shows a visual representation of the evolution of the correlation coefficient and its total error (right ascension error + declination error) as the mean VTEC of the epoch decreases (using 15 different epochs).

Epoch	RA	Dec	RA Error	Dec Error	Correlation Coefficient
11.05	213.75	-10.3125	1.412	2.7475	0.949659
11.075	215.625	-12.1875	3.287	0.8725	0.910935
11.0417	210.938	-9.375	1.4005	3.685	0.941128
11.1167	211.875	-19.6875	0.463	6.6275	0.669265
11.0333	225.938	-18.75	13.5995	5.69	0.564307
11.025	219.375	-16.875	7.037	3.815	0.49188
11.0917	216.562	-21.5625	4.2245	8.5025	0.48441
11.1333	214.688	-23.4375	2.3495	10.3775	0.397166
11.1917	211.875	-11.25	0.463	1.81	0.355636
11.3667	185.625	-76.875	26.713	63.815	0.23138

Table 4.1: Results for different epochs



(a) Correlation coefficient (b) Error of the estimated source

Figure 4.2: Correlation and error of the solution as the mean VTEC decreases

As we can see, the correlation rapidly decreases from 1 (almost linear) to almost 0. The error, on the other hand, does not experience much change for the first epochs but finally increases considerably, because despite the correlation coefficient the estimation can still be done. Past a certain correlation, however, the estimation is harder to perform and the error spikes.

4.2.3 Results

Executing the algorithm with a STEP of 10°, this is the output of the execution:

```
1 [C++: Finding a spike in the VTEC distribution]
2   -> Spike found: 11.05
3 [AWK: Filtering all data by best epoch: 11.05]
4 [C++ -> Fortran: Finding the Person coefficients for possible Suns]
5   -> Input degree step: 10
6 [631 possible Suns considered]
7 [C++: Results]
8   -> Largest correlation coefficient: 0.926959
9   -> Estimated Sun's location: [ra=210, dec=-10]
```

Listing 4.6: Brute force approach algorithm output

As we can see, the possible Sun with the highest correlation coefficient (0.9269) has a location with a right ascension of 210° and a declination of -10°. Considering that for the epoch we are working with the Sun position was 212.338° and −13.060° as the right ascension and declination, respectively, we can see that the estimated Sun's location returned by the algorithm is close to the real one, using a step of 10°.

It can be interesting to see how the computation time grows as more precision is demanded from the algorithm, and if the precision of the results does as well. The following table shows this relation for some input cases:

Step	Considered Suns	Correlation	Estimated location	Time
100	5	0.695358	[ra=200, dec=10]	867ms
50	25	0.695358	[ra=200, dec=10]	303ms
25	106	0.866293	[ra=225, dec=-15]	820ms
12	436	0.92287	[ra=216, dec=-6]	956ms
6	1771	0.934663	[ra=216, dec=-12]	1s 385ms
3	7141	0.937349	[ra=213, dec=-12]	7s 169ms
1	64621	0.939114	[ra=214, dec=-11]	1m 9s 564ms

Table 4.2: Results using different STEP values

As we can see this approach provides the expected results, but has a large computational complexity that increases with the precision we demand.

Furthermore, we can see that a problem appears: in the last two cases there is an increase in the correlation coefficient as expected, but the estimated Sun location does not improve, it is actually less accurate than the previous one. This could be caused by other interferences in the VTEC measurement.

In the next chapter, an optimization is presented for the algorithm to perform these computations, aiming to reduce its complexity.

Chapter 5

Decreasing search range method

In the previous chapter we saw that the BGSEES algorithm for detecting the location of an EUV source (in this case, the Sun) is possible with a first, brute force approach. Here, the algorithm is studied in more detail considering a different method aiming to increase its precision and reduce its computational complexity.

5.1 Decreasing the range of the search

As we saw in the previous chapter, to increase the precision of the algorithm, the *step* with which we iterate over the possible angles (right ascension and declination) is reduced. This causes more possible Suns to be considered. For example, with a step of one degree, we consider all possible right ascensions [0,360] and declinations [-90,90]: $360 * 180 = 64800$ Suns, minus the $2 * 360 - 2 = 718$ right ascensions we do not consider[1] because as we have seen right ascensions for declinations of -90 and 90 are the same location.

We want to have the highest precision possible without having to consider all $64800 - 718 = 64082$ possibilities by progressively reducing the search range.

This first method works as follows: the entire possible range is considered with a large starting step (e.g 60). Once the best Sun is found within this range, the precision is increased (the step is decreased) and the search range is reduced. This way the precision is increased but the number of considered possibilities remains similar each iteration of the algorithm.

[1] We do not consider the 360 right ascensions of the two poles (-90 and 90), hence 2*360, but we do consider the two poles themselves (with any valid right ascension)

5.2 Pseudocode

The following is the pseudocode[2] for the algorithm using this method:

Algorithm 2 Search range decrease

```
1: procedure MAIN
2:     epoch ← findSpikeInData()
3:     filterDataByEpoch(epoch)
4:     bestSun ← nil
5:     r ← defaultRange()
6:     for step = initStep; step >= min; step /= 2 do
7:         for ra = r.lowerRa; ra <= r.upperRa; ra+ = step do
8:             for dec = r.lowerDec; dec <= r.upperDec; dec+ = step do
9:                 currentSun ← computeCorrelationPossibleSun(ra, dec)
10:                if currentSun.correlation > bestSun.correlation then
11:                    bestSun ← currentSun
12:                    r ← newRange(bestSun, step)
13: return bestSun
```

5.3 Implementation

This first piece of code is the main loop of the method, which starts with a default range of ra=[0, 360], dec=[-90, 90] and is reduced every iteration based on the current best Sun's estimated location.

Furthermore, because an increase in the precision of the tested location does not necessarily imply an improvement in the solution, the new candidate is inserted into a *priorityQueue* ordered by the coefficient to assure that the method returns the best one found throughout the entire execution (it is inserted into a priority queue for debugging purposes, but simply saving the best solution would be more efficient).

[2]The code for the special cases of -90 and 90 degree declinations is not included for more readability. It is included in the implementation, in the following section.

```
1 void TraverseGlobe::decreasingSTEP() {
2   int rangeSize = 3;
3   int initStep = 60;
4   sourceInfo currentSun;
5   searchRange range = setRange(currentSun, true, initStep,
     rangeSize);
6   for (double step = initialStep; step >= 0.5; step /= 2) {
7     currentSun = considerPossibleSuns(step, range, plotData);
8     bestSuns.push(currentSun);
9     range = setRange(currentSun, false, step, rangeSize);
10  }
11 }
```

Listing 5.1: Decreasing the range and increasing the precision

The *setRange* function sets a new range based on the estimated location that depends on the precision used for the values and checks that valid range values are returned.

```
1 searchRange setRange(sourceInfo sun, bool defaultR, double step,
     int rangeSize) {
2   searchRange r;
3   if (defaultR) {
4     r.lowerRa = 0;
5     r.upperRa = 360;
6     r.lowerDec = -90;
7     r.upperDec = 90;
8   }
9   else {
10    double rRange = step*rangeSize;
11    double dRange = step*rangeSize;
12    r.lowerRa = sun.ra - rRange >= 0 ? sun.ra - rRange : 0;
13    r.upperRa = sun.ra + rRange <= 360 ? sun.ra + rRange : 360;
14    r.lowerDec = sun.dec - dRange >= -90 ? sun.dec - dRange : -90;
15    r.upperDec = sun.dec + dRange <= 90 ? sun.dec + dRange : 90;
16  }
17  return r;
18 }
```

Listing 5.2: Setting the new range based on the estimated source location

Finally, the *considerPossibleSuns* function has the same functionality as the one from the previous chapter, it iterates over the possible locations, this time, however, it does so over the given range, rather than just the default one.

The names of the lower and upper bound variables have been changed (*uRa* instead of *upperRa*, for example) for readability.

```
1 sourceInfo considerPossibleSuns(double step, searchRange range) {
2   FortranController fc;
3   double pearsonCoefficient;
4   int i = 0;
5   sourceInfo bestSun;
6   bestSun.coefficient = -23;
7   bestSun.location = "salu2";
8
9   for (double dec = range.lDec; dec <= range.uDec; dec += step) {
10    if (dec != -90 and dec != 90) {
11      for (double ra = range.lRa; ra <= range.uRa; ra += step) {
12        pearsonCoefficient = fc.computeCorrelation(&ra, &dec);
13        if (pearsonCoefficient > bestSun.coefficient) {
14          bestSun.coefficient = pearsonCoefficient;
15          bestSun.ra = ra;
16          bestSun.dec = dec;
17        }
18      }
19    }
20    else {
21      //Do only once
22      double ra = 0;
23      pearsonCoefficient = fc.computeCorrelation(&ra, &dec);
24      if (pearsonCoefficient > bestSun.coefficient) {
25        bestSun.coefficient = pearsonCoefficient;
26        bestSun.ra = ra;
27        bestSun.dec = dec;
28      }
29    }
30  }
31  return bestSun;
32 }
```

Listing 5.3: Iterating over possible locations within the given range

Finally, the *computeCorrelation* calls the Fortran code (the same used in the Brute Force chapter) to compute the correlation

Figure 5.1 is a visual representation of how the algorithm works. The X and Z axis are the possible right ascensions and declinations (the possible solutions to our problem) and the Y axis is the correlation coefficient.

As we can see in the plot, as we increase the precision of the search, the range decreases, this can be seen because each point of the plot is each of the locations considered by the algorithm: the density of considered solutions increases as the algorithm gets closer to the local maxima, the correct solution we are looking for.

Another change that could be done to improve this method's performance would be to adopt a sort of *"Dynamic Programming"* strategy in order to avoid repeated calculations (the new range will always be inside the previous range). However, the problem is that because we are dealing with a different precision for the right

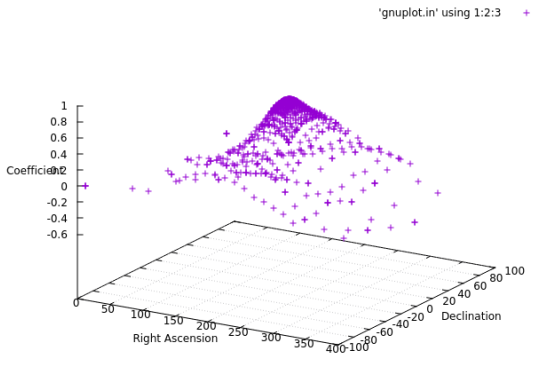


Figure 5.1: All visited candidates of the solution space

ascension and declination values every loop, the algorithm is not working with the exact same value.

5.4 Linear fitting: discarding outliers

In the previous chapter we used a simple method to discard outliers for the VTEC value (using a cutoff value between -0.7 and 0.7). However, another approach to discard outliers was possible: linear fitting. The aim is to find the line that fits best the relation between both variables (cosine and VTEC) by means of linear regression to discard samples that do not fit in it.

We can see that this procedure has been used before in figure 3.2 from the paper *"GNSS measurement of EUV photons flux rate during strong and mid solar flares"*[9].

Manuel Herández-Pajares, the author, provided the Fortran program that performs this computation. After integrating it with the rest of the code, figure 5.2 shows the result of testing it with the flare used in the previous chapter:

The main problem of this method without it, the correlation could be calculated in a single pass of the data. Now, on the other hand, the algorithm needs multiple iterations: first, the solar-zenith angle cosine for each IPP is calculated; then, the outliers are discarded, and finally, the filtered data is traversed one last time to compute the correlation.

Here is the new code in order to compute the correlation for each possible location:

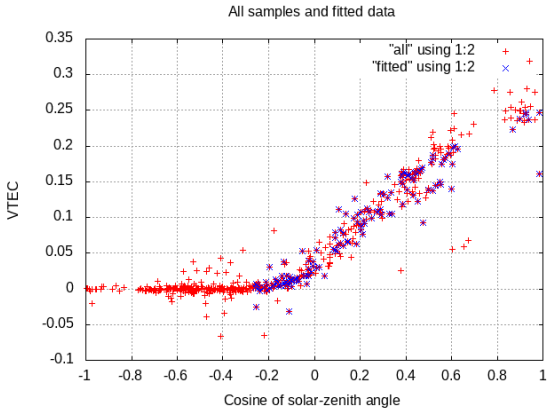


Figure 5.2: All data (red) and fitted samples (blue)

```
1 double computeCorrelation(double* ra, double* dec) {
2   FileManager fileManager;
3   int sigma = 1;
4   int iterations = 6;
5   computecosinesofcurrentsourcefortran_(ra, dec);
6   fileManager.discardOutliersLinearFitFortran(sigma, iterations);
7   return computecorrelationfortran_(ra, dec);
8 }
```

Listing 5.4: Discarding outliers and computing the correlation

As we can see, the data is traversed **at least** 3 times now (more depending on the number of iterations).

5.5 Results

Executing the algorithm with both methods for discarding outliers using the data set from the previous chapter we obtain:

```
1 Estimation error: 1.86776 degrees
2 Execution time: 1.07601 seconds
```

Listing 5.5: Decreasing range using a cutoff value for outliers

As we can see, the algorithm provides a better estimation than the brute force approach. Which took more than minute yielding a result with a precision of 1 degree.

Testing the algorithm using the linear fit approach to discard outliers we obtain the following result:

```
1 Estimation error: 3.57674 degrees
2 Execution time: 37.5578 seconds
```

Listing 5.6: Decreasing range using linear fit for outliers

For this data set in particular, the solution has a similar error (although the linear fit version does not improve with respect to the one with a basic filter) and the execution time is increased considerably. This comparison will be studied in more detail in the *Results* chapter, where different data sets will be used.

Seeing the increase in computational complexity of this method when discarding outliers using linear fitting, we decided to focus on a different method that would rely only on the data itself, instead of considering the many possible locations of the source.

Chapter 6

Least Squares method

As we have seen in previous chapters the overall process to determine the location of the source is studying the correlation between the VTEC value and the solar-zenith angle (or source-zenith angle, speaking in general terms) for a possible location.

That is, for an IPP with a location and an associated VTEC value, given the location of a possible source, compute the cosine between them, and see that the closer the cosine is to 1 (or 180°), the higher the VTEC value.

The idea that we wanted to test with this method was finding the location of the source by performing the inverse operation: having the VTEC, location of the IPP and correlation (1, assuming a near-linear correlation), obtaining the source's right ascension and declination.

6.1 The system of equations

The source-zenith cosine between the source and the IPP was computed using the following equations:

$$unitVectorIPP = \begin{bmatrix} X' \\ Y' \\ Z' \end{bmatrix} = \begin{bmatrix} \cos\delta_g * \cos\alpha_g \\ \cos\delta_g * \sin\alpha_g \\ \sin\delta_g \end{bmatrix} \tag{6.1}$$

$$unitVectorSource = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} \cos\delta_s * \cos\alpha_s \\ \cos\delta_s * \sin\alpha_s \\ \sin\delta_s \end{bmatrix} \tag{6.2}$$

$$\cos\chi = unitVectorIPP \cdot unitVectorSource \tag{6.3}$$

Having the VTEC and source-zenith cosine, we could find the correlation of the two variables, expecting that the real source would yield a near-linear correlation.

Visually, we can see the relation between VTEC and the computed cosine in figure 6.1, obtained in chapter 3 when studying the a specific case for the Sun.

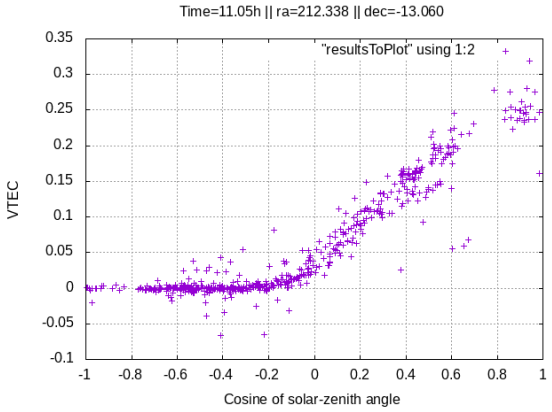


Figure 6.1: VTEC as a function of the solar-zenith angle's cosine

As we have previously seen for this case, there appears to be a linear relation starting around $\cos\chi = -0.1$ between the two studied parameters. Therefore, we could define this linear relation as a **straight line** ($y = mx + b$) by expressing the estimated VTEC value (ΔV) as a function of the source-zenith cosine (6.4).

$$\Delta V = a\cos\chi + b \tag{6.4}$$

Because the cosine is computed using the previous equations (6.1, 6.2, 6.3). It can be expressed as follows, the dot product of both unit vectors:

$$\cos\chi = XX' + YY' + ZZ' \tag{6.5}$$

Where X', Y', Z' are the components of the IPP's unit vector obtained from equation 6.1, and X, Y, Z are the unknowns of our equation: the components of the source's unit vector, which could be used to easily find the right ascension and declination of the source by using trigonometric operations.

However, finding the value of these unknowns is the challenge of this method. Taking the cosine as 6.5 we can express the linear function as:

$$\Delta V = aXX' + aYY' + aZZ' + b \tag{6.6}$$

Because a and b are unknowns as well as X, Y and Z, we can group them as follows:

$$\Delta V = \alpha X' + \beta Y' + \gamma Z' + b \tag{6.7}$$

Our aim after solving the previous equation would be obtaining the values of X, Y and Z. We can see that:

$$\sqrt{\alpha^2 + \beta^2 + \gamma^2} = \sqrt{a^2(X^2 + Y^2 + Z^2)} = \sqrt{a^2} = a \tag{6.8}$$

Because X, Y and Z are the components of a unit vector[1]. The previous allows us to, once we know the values of α, β and γ, obtain X, Y and Z by doing:

$$\frac{\alpha}{\sqrt{\alpha^2 + \beta^2 + \gamma^2}} = \frac{\alpha}{a} = \frac{aX}{a} = X \tag{6.9}$$

In our data we can find, for each IPP: ΔV, X', Y', Z' (because we have the right ascension and declination of the point).

For each of these IPPs, we have an equation of the form $\Delta V = \alpha X' + \beta Y' + \gamma Z' + b$ and, therefore, we have an overdetermined system of equations, with more equations (unknown, depends on the input data) than variables (four: α, β, γ and b)

Knowing how to obtain X, Y and Z from α, β and γ, we can now focus on solving the system of equations to obtain the latter unknowns .

Because we have an overdetermined system of equations, the solution can be approximated using the **Least Squares** approach.

Least Squares is a method to estimate the solution of an overdetermined system of equations (more equations than unknowns, as in our case), by minimizing the sum of the squared residuals. Residuals will be studied in more detail in the lasts sections of this chapter.

The method estimates the solution of a matrix system of the form $y = AX$ using the following equation:

$$X = (A^T A)^{-1} A^T y \tag{6.10}$$

Our system can be represented in matrix form as follows:

$$\begin{bmatrix} \Delta V_0 \\ \Delta V_1 \\ . \\ . \\ . \\ \Delta V_n \end{bmatrix} = \begin{bmatrix} X'_0 & Y'_0 & Z'_0 & 1 \\ X'_1 & Y'_1 & Z'_1 & 1 \\ . & . & . & . \\ . & . & . & . \\ . & . & . & . \\ X'_n & Y'_n & Z'_n & 1 \end{bmatrix} \begin{bmatrix} \alpha \\ \beta \\ \gamma \\ b \end{bmatrix} \tag{6.11}$$

Where y is the VTEC, A the components of the IPP, and X the solution to the system, α, β, γ and b.

With the Least Squares method we could obtain X, Y and Z using the solution to the system (α, β, γ). Taking into consideration equation 6.2, the values of the

[1] $\sqrt{(X^2 + Y^2 + Z^2)} = 1$

right ascension and declination angles can be obtained by performing the inverse trigonometrical operations, that is:

Because we know that $Z = \sin(\delta_s)$ we the declination is obtained by simply computing the arc sinus:

$$\delta_s = \arcsin(Z) \tag{6.12}$$

But we have two operations that involve the declination, $X = \cos\delta_s * \cos\alpha_s$ and $Y = \cos\delta_s * \sin\alpha_s$. Both involve $\cos\delta_s$ so we can equate them:

$$\frac{Y}{\sin\alpha_s} = \frac{X}{\cos\alpha_s} \tag{6.13}$$

$$\alpha_s = \arctan2(Y, X) \tag{6.14}$$

The $\arctan2(Y, X)$ function is a Fortran function that computes the tangent but using two numbers. Instead of relying only on one angle (which would cause ambiguity regarding the quadrant of the angle) this function uses both angles to determine the exact value, which suits our situation.

Using these two operations, the algorithm finally yields our solution: the right ascension and declination estimated from using all the IPPs' information as an overdetermined system.

Furthermore, another possibility would be executing the algorithm multiple iterations. Using the estimated location yielded by each iteration, the algorithm would compute the cosine of the angle between the estimated solution and the IPP (each line) and discard it considering a certain cosine threshold: -0.1 in particular, the cosine in which the linear relation starts (day hemisphere in the case of the Sun) as we have seen in previous chapters. The results of the algorithm with and without iterations are discussed in the last section.

The main advantage of this method over the one introduced in the previous chapter is that it does not need to compute the correlation (which requires passing the data set once) every time a location is considered. The data set itself (the) file is only traversed once.

6.2 Pseudocode

The following is the pseudocode of the algorithm for a single iteration of the algorithm:

The algorithm first stores the necessary data in the arrays and uses them to compute the system solution. The estimated location of the source is then obtained from the system solution by means of the equations presented in the previous section.

Algorithm 3 Least Squares method

```
1: procedure MAIN
2:     for i each line in file do
3:         y(i) ← vtec
4:         A(i,0) ← cos(dec)*cos(ra)
5:         A(i,1) ← cos(dec)*sin(ra)
6:         A(i,2) ← sin(dec)
7:         A(i,3) ← 1
8:     sytemSolution ← (A^T A)^{-1} A^T y
       return obtainEstimatedPosition(sytemSolution)
9: procedure obtainEstimatedPosition(systemSolution)
10:    a ← sytemSolution(0)
11:    b ← sytemSolution(1)
12:    g ← sytemSolution(2)
13:    mod ← sqrt(a^2 + b^2 + g^2)
14:    X ← a/mod
15:    Y ← b/mod
16:    Z ← g/mod
17:    dec ← arcsin(Z)
18:    ra ← arctan(X, Y)
       return (ra, dec)
```

6.3 Implementation

For the case of no iterations, the algorithm can simply be implemented by storing each line of information in an array. On the other hand, when discarding outliers using the result of the previous iteration, two possible implementations exist:

- Using the same static array for all iterations, but storing 0s in the rows of unused IPP information
- Using a dynamical, allocatable array, storing only the information that will be used

When using dynamical arrays in Fortran these have to be allocated/deallocated, the following is the function that adds a new row to the array storing the information, implemented for our case with 4 columns and an unknown number of rows.

```
1  subroutine addRowToArray(array, elem0, elem1, elem2, elem3)
2    implicit none
3    integer :: i, oldSize
4    double precision, intent(in) :: elem0, elem1, elem2, elem3
5    double precision, dimension(:,:), allocatable :: array
6    double precision, dimension(:,:), allocatable :: tmpArray
7
8    if(allocated(array)) then
9      ! Allocate one more row to the tmpArray
10     oldSize = size(array,1)
11     allocate(tmpArray(0:oldSize, 0:3))
12
13     ! Copy the original content of the array
14     tmpArray(0:oldSize,0:3) = array(0:oldSize,0:3)
15
16     ! Append the new row
17     tmpArray(oldSize,0) = elem0
18     tmpArray(oldSize,1) = elem1
19     tmpArray(oldSize,2) = elem2
20     tmpArray(oldSize,3) = elem3
21
22     ! Free the previous array and store the new data in it
23     deallocate(array)
24     call move_alloc(tmpArray, array)
25   else
26     allocate(array(0:0, 0:3))
27     array(0,0) = elem0
28     array(0,1) = elem1
29     array(0,2) = elem2
30     array(0,3) = elem3
31   end if
32 end subroutine addRowToArray
```

Listing 6.1: Adding a new row to a two dimensional array

The function first checks if the array is already allocated, in which case the previous array is copied to a new one with one more row and then the new elements are stored in the it with the new space. If it has not been allocated yet, a new array is created but with one single row.

Because of this, each time a new row is added, the processed data up until that point has to be traversed again (when copying the content of the array to the new one).

On the other hand, using a static array allows us to store the data in one single pass (simply storing 0s if we are not going to use the data), consequently, using the approach with dynamic arrays has more time complexity.

Both methods yield exactly the same results, as the rows that contain 0s are not used in the matrix computations, because of this, the algorithm works with the static version, to assure that there is only a single pass of the data.

The following is the main function of the algorithm, which stores the data in the array reading it from the input file (listing 6.3), computes the solution to the system (listing 6.4.2), and then using that solution obtains the estimated location of the source (listing 6.3).

```
1   subroutine leastSquares(iteration, solutionRa, solutionDec)
2     implicit none
3     integer, intent(in) :: iteration
4     double precision :: solutionRa, solutionDec
5     double precision, dimension(0:numRows) :: matrixVTEC
6     double precision, dimension (0:numRows, 0:3) :: matrixIPP
7     double precision, dimension (0:3) :: solution
8
9     call storeMatrixData(matrixVTEC, matrixIPP, iteration,
      solutionRa, solutionDec)
10    call matrixComputations(solution, matrixIPP, matrixVTEC)
11    call obtainSourceLocation(solution, solutionRa, solutionDec)
12
13  end subroutine leastSquares
```

Listing 6.2: Main Least Squares function

The static storeMatrixData function traverses the file and stores the information, if the method is working with multiple iterations, it also checks using the past iteration's estimation whether the IPP point is valid or not. If it is, the components are computed with function computeComponentsIPP() (listing 6.3), otherwise, 0s are stored for that row.

```
1   subroutine storeMatrixData(matrixVTEC, matrixIPP, iteration,
     solRa, solDec)
2     implicit none
3     integer, intent(in) :: iteration
4     double precision, intent(in) :: solRa, solDec
5     double precision, dimension(0:numRows) :: matrixVTEC
6     double precision, dimension (0:numRows, 0:3) :: matrixIPP
7     double precision :: vtec, raIPP, decIPP
8     double precision :: xIPP, yIPP, zIPP
9     integer :: i, validSample
10
11    call openFile(inputFileName)
12
13    do i = 0, numRows
14      read (1, *, end = 240) vtec, raIPP, decIPP
15      validSample = 1
16      if (iteration /= 0) then
17        validSample = checkOutlier(solRa, solDec, raIPP, decIPP)
18      end if
19      if (validSample == 1) then
20        call computeComponentsIPP(raIPP, decIPP, xIPP, yIPP, zIPP)
21      else
22        vtec = 0
23        xIPP = 0
24        yIPP = 0
25        zIPP = 0
26      end if
27      matrixVTEC(i) = vtec
28      matrixIPP(i, 0) = xIPP
29      matrixIPP(i, 1) = yIPP
30      matrixIPP(i, 2) = zIPP
31      matrixIPP(i, 3) = 1
32    end do
33    240 continue
34    close(1)
35  end subroutine storeMatrixData
```

Listing 6.3: Storing the data from the input file

```
1   subroutine computeComponentsIPP(ra, dec, xIPP, yIPP, zIPP)
2     implicit none
3     double precision, intent(in) :: ra, dec
4     double precision :: raRad, decRad
5     double precision, intent(out) :: xIPP, yIPP, zIPP
6
7     raRad = toRadian(ra)
8     decRad = toRadian(dec)
9
10    xIPP = cos(decRad)*cos(raRad)
11    yIPP = cos(decRad)*sin(raRad)
12    zIPP = sin(decRad)
13  end subroutine computeComponentsIPP
```

Listing 6.4: Compute the components of the IPP's unit vector

After storing all the necessary data in the arrays, the algorithm perform the calculations of equation 6.10. The function to compute the inverse is the only matrix operation that is not implemented by Fortran's standard library, so the *LAPACK (Linear Algebra PACKage)* library for numerical linear algebra[19], which implements it, is used by the algorithm for this computation.

```
1   subroutine matrixComputations(solution, A, Y)
2     implicit none
3     double precision, dimension (0:3), intent(out) :: solution
4     double precision, dimension (0:numRows), intent(in) :: Y
5     double precision, dimension (0:numRows, 0:3), intent(in) :: A
6     double precision, dimension (0:3, 0:numRows) :: transposedA
7     double precision, dimension (0:3, 0:3) :: covMat
8
9     transposedA = transpose(A)
10    covMat = inv(matmul(transposedA, A))
11    solution = matmul(matmul(covMat, transposedA), y)
12  end subroutine matrixComputations
```

Listing 6.5: Function matrixComputations to solve the system

After the solution to the system has been obtained, the degrees are obtained as explained in the previous sections, and using Fortran's *atan2()* function:

```
1   subroutine obtainSourceLocation(solution, solRa, solDec)
2     implicit none
3     double precision, dimension (0:3), intent(in) :: solution
4     double precision, intent(out) :: solutionRa, solutionDec
5     double precision :: a, b, g, mod, radianRa, radianDec
6     double precision :: X, Y, Z
7
8     a = solution(0)
9     b = solution(1)
10    g = solution(2)
11    mod = sqrt(a*a + b*b + g*g)
12    X = a/mod
13    Y = b/mod
14    Z = g/mod
15    radianRa = datan2(Y,X)
16    radianDec = dasin(Z)
17    if (radianRa < 0) then
18      radianRa = radianRa + 2*PI
19    end if
20    solRa = toDegree(radianRa)
21    solDec = toDegree(radianDec)
22  end subroutine obtainSourceLocation
```

Listing 6.6: Obtaining the source's location using the system's solution

The solRa and solDec variables are the ones that are either returned to the C++ controller or used perform another iteration of the algorithm.

6.4 Results

Here the results of the algorithm with both methods for discarding outliers are studied, using the data set from the previous chapter.

6.4.1 Single iteration

These are the results of the algorithm with a single iteration, that is, solving the equation system once and using all the available data (with the cutoff value for the VTEC introduced in the previous chapter):

```
1     Estimation error: 4.57509 degrees
2     Execution time: 0.00851647 seconds
```

Listing 6.7: One iteration of the Least Squares method

As we can see, the error is there has been a slight increase in the error compared to the Decreasing Range method, but the execution time has been greatly reduced.

6.4.2 Multiple iterations: narrowing the search

As mentioned before, a possibility to improve the result could be iterating using the estimated value of the algorithm. This is done using the *checkOutlier()* function when storing the matrix data (listing 6.3). The function implements the method to discard IPPs discussed earlier in the chapter: computing the cosine of the angle between the estimated solution and the IPP and discard it considering a cosine threshold: -0.1, to work only IPPs from the day hemisphere.

```
1   integer function checkOutlier(solutionRa, solutionDec, raIPP,
     decIPP)
2     implicit none
3     double precision, intent(in) :: solutionRa, solutionDec, raIPP,
      decIPP
4     double precision :: sourceZenithAngle
5     integer :: validSample, returnValue
6
7     sourceZenithAngle = computeSourceZenithAngle (solutionRa,
     solutionDec, raIPP, decIPP)
8     if (sourceZenithAngle >= COSINE_THRESHOLD) then
9       validSample = 1
10    else
11      validSample = 0
12    end if
13    returnValue = validSample
14    return
15  end function checkOutlier
```

Listing 6.8: Function checkOutlier to discard outliers

The following table shows the results of using 10 iterations with this method:

As we can see, multiple iterations do not present an improvement in the estimation. This is caused because in each iteration IPPs are discarded and therefore the system of equations has less information to compute the estimation.

Because there does not appear to be any improvement, another option could be to use the estimation error obtained from the covariance matrix as an indicator of how precise the estimation is. This is obtained by adding the covariances of the the components of the system's solution. These covariances are the elements of the diagonal from the matrix $(A^T A)^{-1}$ from equation 6.10. This option using the *LS estimation error* (not the estimation error, obtained from comparing our estimation to the real position of the source) will be tested with more data sets in the results chapter.

6.4.3 Multiple iterations: residual sum

Another possibility would be to discard IPPs by using the residual sum of the Least Squares estimation (not the one we obtain comparing our solution to the

Iteration	Total estimation error (degrees)	Time (seconds)
1	4.57509	0.0134369
2	7.98606	0.00955695
3	10.9058	0.00946149
4	5.50332	0.0102698
5	7.25672	0.0149682
6	8.48851	0.0115219
7	8.46854	0.0136026
8	10.206	0.014326
9	6.23355	0.0140396
10	7.83679	0.0156941

Table 6.1: 10 iterations of the Least Squares method

real position), rather than by discarding the night hemisphere. This value can be obtained using the real VTEC values (from the input data) with the ones we would obtain using the least squares estimated model.

$$\sigma^2 = \frac{\sum_{i=0}^{m}(y_i - f(i))^2}{m} \tag{6.15}$$

Which can then be used to discard IPPs based on the following comparison.

$$|(y_i - f(i)| <= 3\sigma \tag{6.16}$$

In this case the algorithm does not consider the estimation error to keep the best possible solution, but rather iterates and returns the result of the last iteration.

Iteration	Total estimation error (degrees)	Time (seconds)
1	4.57509	0.00756712
2	4.71719	0.010272
3	4.7942	0.0120997
4	4.47241	0.0105663
5	4.47009	0.0125787

Table 6.2: 5 iterations of the Least Squares method using the residual sum of squares to discard outliers

As with the previous iteration method, we are discarding rows and therefore the Least Squares method reduces its precision, therefore yielding a larger error. How-

ever, the error seems to decrease after iteration 4 and slightly improve the original solution for this case in particular, but whether it improves the solution or not for other cases will be studied in the results chapter.

Although for this specific case the method does not present an improvement in comparison to the Decreasing Range method, the Least Squares method is significantly faster in terms of execution, and will be studied in more detail, where more data sets will be used to see which yields the best results.

Chapter 7

Other methods and optimizations

In the previous chapters two methods to estimate the location of a flare have been presented.

Here, other possible optimizations are considered that could be used to, in the future, extend the algorithm and perhaps improve its performance and accuracy.

While some are completely different methods, others are optimizations that can be built on top of any approach.

7.1 Hill Climbing

Using the decrease range method in chapter 6 we could see the plot of all the possibilities the algorithm considers:

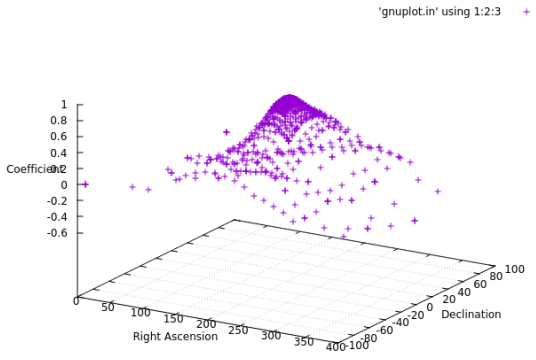


Figure 7.1: All visited candidates of the solution space

In this solution space, there appears to be a "hill" (our solution) so an attempt to solve the problem using a *Hill Climbing* approach was also considered.

Hill Climbing is an heuristic search algorithm that moves through the solution space of a problem by finding the best neighbour of the current solution until it can not progress any more. Seeing the previous figure we thought it could be interesting to test this method for our problem, so a simple greedy algorithm was implemented as a first attempt to test if the method could work.

The following is the implementation of such test, which checks its surrounding neighbours and moves to the best one (the objective function is the correlation) until the solution can not be improved any further.

```
1   // Starting position
2   sourceInfo current;
3   current.ra = 160;
4   current.dec = -20;
5   int i = 0;
6   // Loop with limit or until no progress can't be made
7   while (++i < 100) {
8     vector<sourceInfo> candidates = getNeighbourList(current);
9     sourceInfo newCandidate = getBestCandidate(candidates);
10    if (newCandidate < current) {
11      break;
12    }
13    current = newCandidate;
14  }
15
```

Listing 7.1: Hill Climbing

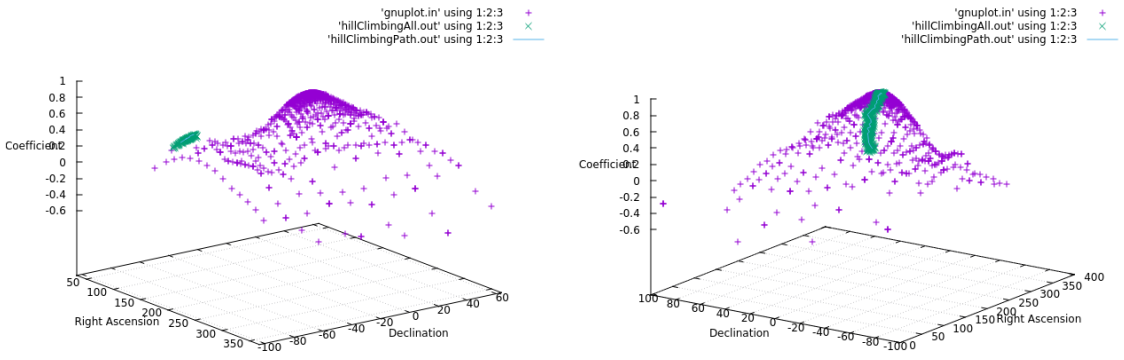
The following figure shows the results of the execution for two different starting states. For both of them, the algorithm ran until it could not progress any further (wasn not interrupted by the iteration limit). The plots contain both the possibilities considered by the decrease range method (in purple, the same plot as 7.1), and the path taken by the greedy algorithm (in green).

Visually, we can see that the number of considered possibilities is inferior and the top is reached for case (b), but a problem appears: **a local maxima**.

If we take a starting right ascension of 160° and a declination of -20°, the algorithm takes a path that gets to the top of our solution, yielding similar results to the previous, decrease range method.

However, with a starting right ascension of 100° and a declination of -60°, the algorithm finds a local maxima on its way and cannot progress to the real best solution.

As a result, the Hill Climbing approach is not reliable for all cases, considering that local maxima may exist for our type of problem, leading to incorrect estimations in some cases.



(a) Start: ra=100°, dec=-60°

(b) Start: ra=160°, dec=-20°

Figure 7.2: Paths taken by the Hill Climbing algorithm

7.2 Simulated Annealing

Simulated Annealing is an heuristic search algorithm that approximates the optimal solution of a problem, similar to Hill Climbing. The main difference is the use of probability to accept a solution as better than the current one, to be able to explore other areas of the solution space.

A possible solution to the problem of local maxima seen in the previous section would be using the Simulated Annealing algorithm so that other parts of the solution space could be explored, finding other paths that might lead to our desired solution, instead of only focusing on one path that could get stuck in a local maxima.

Although it might yield better results, simulated annealing approximates the optimal solution, but does not guarantee finding the global maxima.

7.3 OpenMP

OpenMP (Open Multi-Processing) is an specification for compiler directives and library routines available for C, C++ and Fortran used for parallel programming.

Using OpenMP would not be an entirely different method, but rather an optimization that could be used with all the presented methods to try to parallelize some regions of the code. It would be feasible for our case considering it is available for both C++ and Fortran, the main languages used for the algorithms.

An example of how multi-processing could be used for the algorithm would be parallelizating the computation of the correlation in the decrease range method: each thread could read a part of the input and compute the necessary variables used to compute the final value of the correlation. Because these variables are sums, each

thread could handle a part of the input and finally we could compute the total sum using OpenMP's reduction clause (which sums the values from each thread at the end of the partial loop).

These two possibilities (using OpenMP and Simmulated Annealing) could be implemented in the future if more tests were done with the algorithm, but for now we decided to focus on the methods from previous chapters.

Chapter 8

Results: solar flares

In this chapter different data sets will be studied using the two presented methods for the *BGSEES* algorithm: the **Least Squares** and **Decrease Range** methods.

The aim of the chapter is to test them against each other and using different parameters to see which method yields the best results.

The algorithm is tested using data from Solar flares, for these, the ti files of days when a flare had taken place were used to compare the results. The data was filtered around the time of the flare (30 minutes before and after, if there was an exact moment in time).

Below is the list of times (*year.day.seconds*) of the different flares we studied. The seconds are either an exact moment in time or a range used in the plots of the papers the flares are listed in.

These are the flares listed in the paper *"GNSS measurement of EUV photons flux rate during strong and mid solar flares"*[9]

- 2003.301.39777
- 2003.308.71000-71100
- 2005.020.24200-24400
- 2011.210.44134

And those listed in *"GPS as a solar observational instrument: Real-time estimation of EUV photons flux rate during strong, medium, and weak solar flares"*[22]

- 2001.347.51800-52200
- 2002.196.72240
- 2005.258.30990
- 2012.066.4400-4700

- 2012.130.50600-51000
- 2012.297.11600-12000

To perform this study, the best epoch within the given range is found using the mean VTEC, as shown in chapter 5. The data is then filtered using this epoch and the algorithm is executed using the necessary parameters, outputing the **execution time** of the algorithm and the **absolute error** of the estimation, obtained by computing the angle between correct Sun position[1] and the estimated location, using the same operations we have used in previous chapters to compute the angle.

```
#!/bin/bash
strings=(
  '2003.301,36000,41400'
  '2011.210,44134'
  [..All the filenames..]
  '2012.297,11600,12000'
)
tiDataFolder="/home/mbdavid2/Documents/dataTi/"
for i in "${strings[@]}"; do
  dataInfo="$i"

  # Split the information
  arrayInfo=(${dataInfo//,/ })

  # Use the range if specified, compute it otherwise
  if [ ${#arrayInfo[@]} = 2 ]; then
    let lowerLimit="${arrayInfo[1]}"-1800
    let upperLimit="${arrayInfo[1]}"+1800
  else
    let lowerLimit="${arrayInfo[1]}"
    let upperLimit="${arrayInfo[2]}"
  fi

  # Name the file according to the parameters
  tiDataFile="ti.""${arrayInfo[0]}"
  outputFileName="$tiDataFile"."$lowerLimit"-"$upperLimit"

  # Filter and compress
  zcat "$tiDataFolder""originals/""$tiDataFile"
  | gawk -v lower="$lowerLimit" -v upper="$upperLimit"
  '{/a/; if ($3 >= lower/3600 && $3 <= upper/3600) {print $0;}}'
  > "$tiDataFolder""$outputFileName"
  gzip -f "$tiDataFolder""$outputFileName" # -f to force overwrite
done
```

Listing 8.1: Filtering the ti file

[1]The correct Sun position at that moment is obtained from the data, it is one of the many fields the ti files contain

The ti files that contain data for the entire day are filtered using the previous bash script, which has a list with the information of each file to be filtered: the name of the original file and the upper and lower limits of time (or a specific moment used to compute the limits). It then filters each file using a simple AWK one-line script that checks the field with the time.

The study is divided in three categories, based on the method used to discard outliers from the input data:

- Using the data from **all IPPs** without filtering out any outliers
- Using a **cutoff value** for **all the VTEC data** that will be used for the computations of the algorithm
- Using **linear fit** for the Decreasing Range method to discard outliers and **multiple iterations** for the Least Squares method to try to improve the solution, both presented in their respective chapters.
- Using data from **multiple epochs** rather than just one.

8.1 Using all available data

For this first test, all the data from the ti file is used: in the case of the Decreasing Range (DR) method, all the VTEC values that could be outliers are used to compute the correlation, and in the case of the Least Squares (LS) method, all are used for the equations of the system.

Data set	Error (°)	Time (s)
2001.347	113.813	1.03879
2002.196	83.5147	0.26934
2003.301	24.6405	1.07813
2003.308	128.59	0.971644
2005.020	20.1031	0.916863
2005.258	91.3298	0.498707
2011.210	90.5716	1.46812
2012.066	133.236	2.30571
2012.130	162.888	2.49292
2012.297	78.487	0.789705
Total	927.174	11.8299

Data set	Error (°)	Time (s)
2001.347	106.064	0.0175844
2002.196	66.2043	0.0158694
2003.301	42.2689	0.0154119
2003.308	55.4949	0.326298
2005.020	124.218	0.14858
2005.258	111.073	0.0264138
2011.210	98.776	0.0558694
2012.066	64.186	0.0143492
2012.130	73.1871	0.0321778
2012.297	47.0189	0.00680915
Total	788.491	0.659363

Table 8.1: Estimation error and execution time for different data sets using the DR (left) and LS (right) methods without any filter

The main problem of this method is that outliers can cause the computation of the mean to be unstable, which causes the algorithm to use incorrect epochs.

Furthermore, the outliers can cause numerical instability in some of the methods' computations, if one has a large value, for example, the computation of the correlation relies on the sum of the square of the VTEC value, so the total can lead to incorrect results because of the rapid increase of this variable.

Additionally, the LS method is significantly faster: all data sets together add up to a total execution time of less than one second, whilst the DR method needs that time or even more for almost each of the data sets.

While in all previous sections a direct VTEC filter has been used, using all data was tested too to compare the results.

8.2 Direct VTEC filter

Although this is a very simple approach to discard outliers, we decided to test it because the value of the Delta VTEC does not usually surpass values such as 0.7, only some IPPs might present values like this, due to other interferences in the satellite data. The flare from the day 301 of 2003, for example, studied in previous chapters, is one of the most powerful flares ever recorded, and the peak value of the Delta VTEC is 0.4.

These are the results of the execution using a cutoff value of 0.7. This value was selected as it is the one that yielded the best results in a range of 0.3 to 1:

Data set	Error (°)	Time (s)
2001.347	3.53947	1.03243
2002.196	27.6877	0.287287
2003.301	3.93239	1.36538
2003.308	131.366	0.891328
2005.020	64.8737	0.551884
2005.258	48.7806	1.00214
2011.210	126.204	1.23266
2012.066	75.1081	1.85402
2012.130	56.7937	2.42187
2012.297	1.46042	2.86145
Total	539.746	13.5005

Data set	Error (°)	Time (s)
2001.347	3.41832	0.0114944
2002.196	46.578	0.00492704
2003.301	4.57509	0.00724225
2003.308	141.865	0.378145
2005.020	38.3263	0.129509
2005.258	1.88011	0.0103007
2011.210	38.5213	0.0603164
2012.066	70.1063	0.0251154
2012.130	9.26238	0.0518934
2012.297	3.00704	0.0392889
Total	357.54	0.718233

Table 8.2: Estimation error and execution time for different data sets using the DR (left) and LS (right) methods with a cutoff filter

As we can see, there has been a significant improvement for both methods in some of the flares with low error values (although some still present a high error), and because this filter is applied when performing the filtering by time of the data, it does not have an impact on the complexity of the execution.

Because the direct filter yields significantly better results for both cases, the next sections use it for filtering the first traversal of the data (when the necessary data is obtained from the ti file), before performing an additional filter to attempt to discard outliers.

8.3 Decreasing range: linear fit

This approach, introduced in chapter 5, finds the straight line that fits the data set and discards outliers based on a sigma parameter. This filter can be performed several times (the number of iterations). The results of the decreasing range method using a sigma of 3 and 4 iterations (the best combination of those values in a range from 1 to 10) is shown below:

Data set	Error (º)	Time (s)
2001.347.gz	3.66657	45.4151
2002.196.gz	30.9237	42.6182
2003.301.gz	3.57674	51.7222
2003.308.gz	79.3679	43.6955
2005.020.gz	68.911	37.3023
2005.258.gz	22.6375	50.5795
2011.210.gz	19.7344	55.8291
2012.066.gz	84.3076	31.9302
2012.130.gz	21.896	42.326
2012.297.gz	0.679492	53.7369
Total	335.701	455.155

Table 8.3: Results for different data sets using linear fit with the DR method

While this approach improves upon the solution of the Least Squares method (from 357.54 to 335.701), it requires a significantly higher execution time: each execution takes almost a minute, which is more than the total execution time for all data sets using the same method in the previous section.

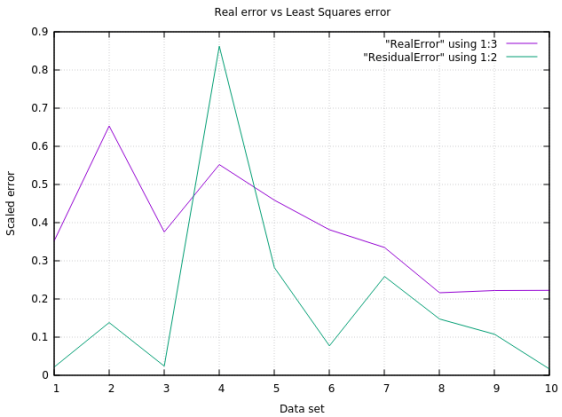
8.4 Least Squares: Iterations

8.4.1 Discarding by source position

In chapter 6 iterations did not provide better results for a specific case, but we decided to test it using all data sets in case there was any overall improvement. These are the results of executing the algorithm with 10 iterations, saving the result of the best iteration. The best iteration is that which has the least estimation error, given by the covariance matrix (as seen in 6.4.2):

Data set	Error (°)	Time (s)
2001.347	3.41832	0.0194832
2002.196	46.578	0.00818968
2003.301	4.57509	0.0171783
2003.308	141.865	0.366494
2005.020	38.3263	0.141508
2005.258	1.88011	0.0177967
2011.210	38.5213	0.0704044
2012.066	70.1063	0.049831
2012.130	9.26238	0.0711422
2012.297	3.00704	0.0592826
Total	357.54	0.82131

(a) Results for different data sets



(b) Comparison of the real error (purple) and the estimated Least Squares error (green)

Figure 8.1: Results of LS method with 10 iterations based on the estimated LS error

As we can see, the results in terms of estimation error are exactly the same as the ones seen in table 8.2 with a slight increase in execution time due to the number of iterations. Because there is no improvement with consecutive iterations, the algorithm just keeps the best result from the first one, hence the same results.

Despite the fact that using this approach does not improve the results, it can be interesting to see how the estimated LS error evolves compared to the real error for all the studied data sets, figure 8.1b shows a comparison for all the tested data sets.

As we have seen in previous sections, some of the studied flares are hard to detect for both methods, due to their intensity. We can observe that both functions have similar spikes: if the solution has a large error, it is because the Least Squares method could not provide a good solution with the available data.

8.4.2 Discarding by the residual sum of squares

In the Least Squares chapter a slight improvement could be observed when discarding outliers based on the residual sum of squares of each iteration, but only for a specific flare. These are the results of the total error and total execution time for all 10 data sets using up to 30 iterations.

Iterations	Error (°)	Time (s)
2	375.793	0.741631
3	458.501	0.752343
4	454.244	0.736867
5	422.8	0.744048
6	447.083	0.767054
7	445.304	0.813639
8	457.769	0.80455
9	401.549	0.87425
10	385.327	0.824476
11	370.055	0.838879

Iterations	Error (°)	Time (s)
12	387.599	0.893248
13	371.906	0.893959
14	376.933	0.910624
15	365.973	0.934418
...	...	...
22	376.933	0.939168
23	365.973	0.915974
...	...	...
29	376.933	0.95647
30	365.973	1.16539

Table 8.4: Total estimation error and total execution time for different data sets using LS method with iterations

As we can see, the total error increases significantly once the algorithm starts iterating but the results slowly improve (and the execution time increases) until iterations 14 and 15, where the algorithm periodically yields the same results every two iterations. Still, no iteration presents any improvement in comparison to the first one: 357.54 degrees, so using a single iteration appears to be the best option for the Least Squares method.

8.5 Using multiple epochs

Although this change did not come up when testing the data sets from the Sun but those from far-away stars (in the next chapter), using multiple epochs resulted in better estimations with the Sun as well so it is included in this chapter.

As we have seen when reducing the number of samples by discarding oultiers with the Least Square method, the results did not improve due to less equations being used for the system. Because of this, we decided to test the algorithm using data from more than one epoch, that is, when filtering by the time of the best found epoch, also filtering by the time of the second best one (or third), increasing the number of samples used by the algorithm. These are the results of using the two best epochs rather than just the best one, and only using a cutoff filter:

Data set	Error (°)	Time (s)
2001.347	6.47413	2.14455
2002.196	23.1275	0.59226
2003.301	1.86704	2.0248
2003.308	94.6148	1.19993
2005.020	72.4195	1.18676
2005.258	44.2516	1.92844
2011.210	11.4821	2.54239
2012.066	89.4625	3.97582
2012.130	20.645	4.37908
2012.297	4.82785	4.78413
Total	369.172	24.7582

Data set	Error (°)	Time (s)
2001.347	4.49127	0.161254
2002.196	8.30599	0.00555821
2003.301	4.25894	0.00930464
2003.308	129.135	0.363268
2005.020	40.0696	0.13032
2005.258	12.3223	0.011427
2011.210	6.11407	0.0623802
2012.066	43.7512	0.0282298
2012.130	7.72546	0.0567187
2012.297	3.8804	0.0432351
Total	260.055	0.871695

Table 8.5: Estimation error and execution time for different data sets using the DR (left) and LS (right) methods with a cutoff filter and using data from two epochs

Although using linear fit to discard outliers with the DR method seemed to improve the results in previous sections, using it with multiple epochs does not present any improvement (the total error of the DR with linear fit for two epochs is 400.146). But it is still an improvement in comparison to the DR method using only the cutoff filter. Regarding the LS method, using multiple iterations with two epochs reduces the error, but not iterating at all is again the combination that yields the lowest error values, a total of 260.055 degrees. Although some of the flares are detected with a slightly higher error, the total is significantly lower.

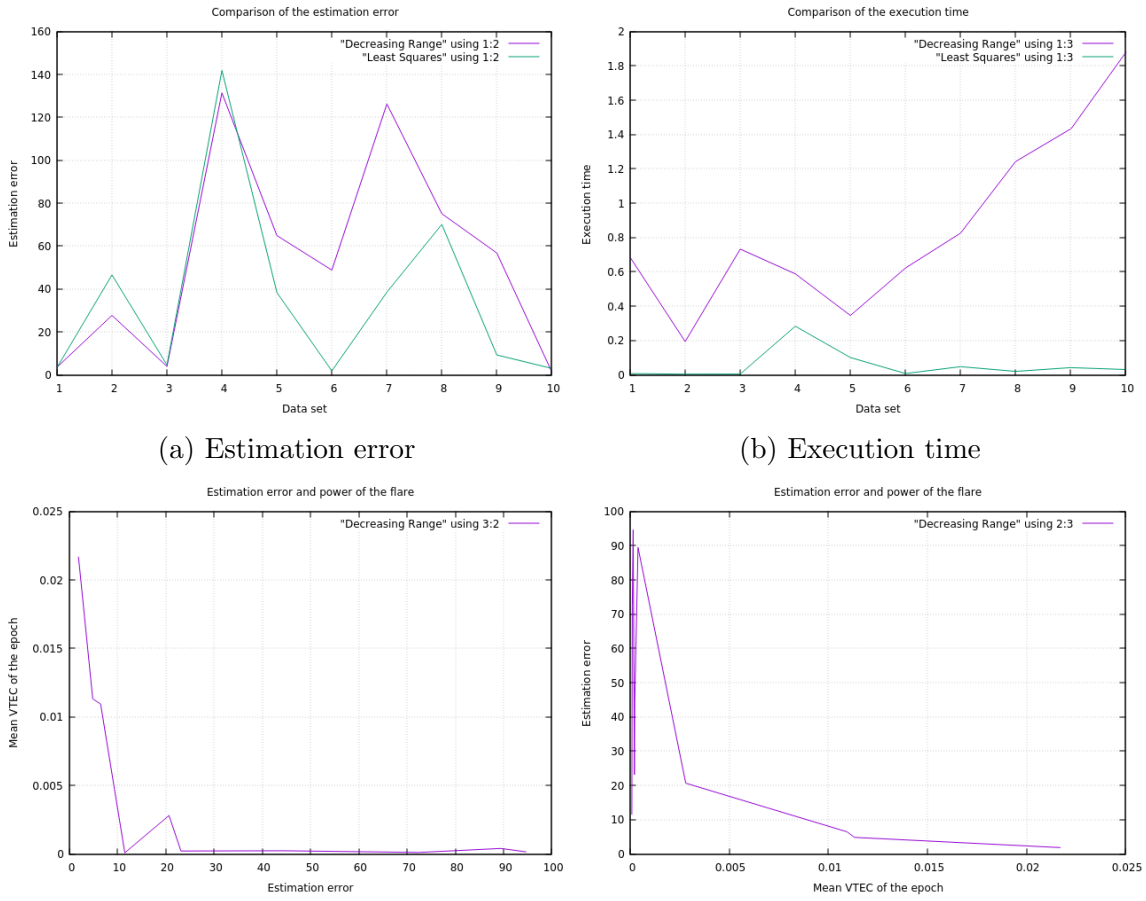
Furthermore, using 3 epochs instead of 2 resulted in a total error of 390.954 for the DR method and 294.689 for the LS method, and there is no improvement

beyond using 3 epochs. This change was very important for some tests presented in the next chapter.

8.6 Discussion

In this chapter we have seen that working with the entire data set of IPPs does not yield good results, there is too much noise and the results differ considerably from the real position of the Sun. Considering that this is a new method to detect flares, however, this could have been the case in general (the detection could not have been possible, or at least not with enough precision), but when using a direct filter for the VTEC values we have seen that some of the data sets yield results with errors as small as 4 degrees.

It can be interesting to see, however, how some data sets still have a high error, due to the nature of the flare (perhaps it was not sufficiently strong to have an impact on the ionosphere), figure 8.2a compares the Least Squares and Decreasing Range methods' error of the direct filter approach.



(a) Estimation error

(b) Execution time

(c) Mean VTEC as a functiondsadasr

(d) dsadasdMean VTEC as a functiondsadas

Figure 8.2: Comparison of the estimation error and execution time for the Least Squares and Decreasing Range methods

Furthermore, the correlation between the intensity of the flare and the error of the estimation can be seen in figure 8.2c if we plot the mean VTEC of the best epoch that the algorithm uses (representative of the strength of the flare) as a function of the estimation error. Or the inverse (figure 8.2d. These two plots are representative of how the estimation error increases as the power of the flare decrease.

Regarding the performance of the algorithm, figure 8.2b compares the execution time of both the LS (green) and DR (purple) methods. The DR method takes more time for all data sets and there would be an even greater difference would we compare the method using linear fit.

In conclusion, the Decreasing Range method (using linear fit to discard outliers) and the Least Squares method (using a direct filter) provide good results for some of the data sets with powerful flares, therefore, these two methods will. However, the combination that yields the best results is the Least Squares method using a direct filter and data from the two best epochs rather than just one.

Therefore, this will be the main method used to study stellar flares in the next chapter, although the other presented options will be taken into consideration as well.

Chapter 9

Stellar flares

9.1 Study on the feasibility of stellar flare detection

Before starting to adapt the algorithm for detecting solar flares to this scenario, a study was conducted parallel to its development, to see if the energy from flares originating in far-away stars could be detected using the already existing method, namely the GNSS Solar Flare Activity Indicator (GSFLAI) algorithm [9].

To study if this was feasible, the existing algorithm was executed with certain candidates of flares and GRBs to see if they could be detected.

The project supervisor, Manuel Hernández-Pajares, who as mentioned before has previously performed several studies on the subject, performed these tests with the GSFLAI algorithm. This algorithm takes into consideration the location of the source (the Sun) to see if there is a relation between an increase in the VTEC of the ionosphere and the solar-zenith angle to determine if this increase is caused by a solar flare [9]. The aim was changing the location of the source (the Sun) to that of the star being studied to see if had any effect on the ionosphere.

However, because of its large execution time, the objective was to:

- Find a database for possible candidates, several online archives with information about previously recorded Gamma Ray Bursts were considered.
- Select an appropriate source of this pool of candidates by writing a quick script that yielded an ordered list of the best candidates based on certain factors, instead of selecting a random source.

9.1.1 Sources of data and possible candidates

The three main databases we considered for the study were:

- The GRB collection website of Jochen Greiner, scientist at the Max-Planck-Institute for extraterrestrial Physics (MPE) [16], which offers a collection of detected GRBs by different telescopes and observatories.
- The Magnetar Outburst Online Catalog (MOOC), developed by the Institute of Space Sciences (CSIC-IEEC, Barcelona) [14]. We also had the pleasure to meet one of the leaders of this project, Nanda Rea, and discuss
- The Neil Gehrels Swift Observatory website and archive by the National Aeronautics and Space Administration (NASA), Goddard Space Flight Center [7] which contains an archive of detected GRBs by the Swift observatory and is constantly updated.

Because of the layout of the website and how the data could be accessed, the option with which we started was the Swift Database, as the data could be visualized in an HTML table and was easily accessible.

9.1.2 The Neil Gehrels Swift Observatory and its data

The Swift Observatory is a NASA mission with international participation, designed to observe GRBs and their afterglows to study topics such as the origins of GRBs or what they can reveal about the early stages of the universe [21]. The observatory is equipped with three main instruments that work with each other to study GRBs [5] [7]:

- The **Burst Alert Telescope (BAT)**, tasked with detecting the GRBs and computing their positions. This triggers the spacecraft to point the other telescopes to the burst so it can be studied in more detail.
- The **X-ray Telescope (XRT)**, used for studying the X-ray radiation and taking images of the bursts which in turn help increase the accuracy of the location estimation.
- The **UV/Optical Telescope (UVOT)**, which serves a similar purpose to the XRT, but studies the ultraviolet band of the spectrum.

For each detected GRB, the data obtained by the different telescopes is given. The parameters that are relevant to our study and determine the fitness of each of the candidates are:

- The **name of the burst**, given by the date it was detected. For example, the GRB named 190220A was detected the 20th of February of 2019.
- The **Universal Time (UT)** of the detection, that is, hh:mm:ss of the day given by the name.

- The **fluence** detected by the BAT component, in units of keV.
- The **UVOT magnitude**, measured by the UV Telescope.
- The location that triggered the detection, given as **Right Ascension (Ra)** and **Declination (Dec)**.

9.1.3 Objective function

Our main goal in this section was to obtain a list of GRB candidates ordered from more to less probable to be detected by the algorithm based on their fitness. To obtain this score we had to define an objective function, taking into consideration two factors:

- The **strength** of the burst, given by the UVOT magnitude. If this value was not available (as it was the case with many of the candidates) the BAT fluence was considered as its strength. This values were already given by the archive and no additional computations were required.
- The **angle between the burst and the Sun**, this was an important factor because bursts having an effect on the night hemisphere should be more noticeable than those hitting the day one, where the Sun has a bigger influence.

The final score is computed by adding the previous factors. While the angle ranged from 0 to 180, the strength had smaller values, giving more weight to the angle in the final score.

Computing the angle

As mentioned before, the Swift archive gives us the Right Ascension (Ra) and Declination (Dec) where the source is thought to be located.

The location of the Sun, on the other hand, is unknown. But we do know the time when the burst was detected.

The supervisor, Manuel Hernández-Pajares, provided me with an algorithm which takes date (year, month, day and UT) and a planet of the Solar System (or the Sun, our case) as the input and returns its location in the celestial sphere, that is, its Ra and Dec.

This algorithm belongs to the **Starlink Project** (Rutherford Appleton Laboratory), which provided open-source software like the one at hand to astronomical institutions. Although it was shut down in 2005, the code is still available and we could use it for our study [1].

Knowing the location of the GRB and that of the Sun: the cosine of the angle between both can be computed and used as a parameter for the objective function.

9.1.4 Obtaining the data

Regarding the scrapping of the website to parse the data and obtain this ordered list, **Python** was chosen because the problem required a quick implementation, and Python's libraries offered a great tool to develop a simple solution as quick as possible.

In the script, the website with the table of bursts (see figure x) is scrapped using Python's **BeautifulSoup** library, which has an HTML and XML parser that allows us to easily select and obtain data from a given website.

Insertion sort was used so we could insert every considered GRB into a list of sorted candidates as we were traversing the table.

The best 5 candidates of the resulting sorted table[1], are shown here:

Name	Ra	Dec	UVOT	BAT	Date	Angle	Score
190202A	166.506	9.393	V=17.94	60	2019.02.02	148.79	166.69
190129B	117.285	1.257	n/a	n/a	2019.01.29	158.11	158.11
181228A	49.831	13.212	V¿19.1	100	2018.12.28	134.41	153.41
181010A	52.574	-23.023	V¿19.4	6.9	2018.10.10	133.22	152.22
190219A	189.686	76.606	n/a	39	2019.02.19	111.74	150.74

Table 9.1: Best results from the Swift GRB database

Sadly, none of the GRBs provided good results, an example of the result of the execution of the GSFLAI algorithm is shown in figure 9.1

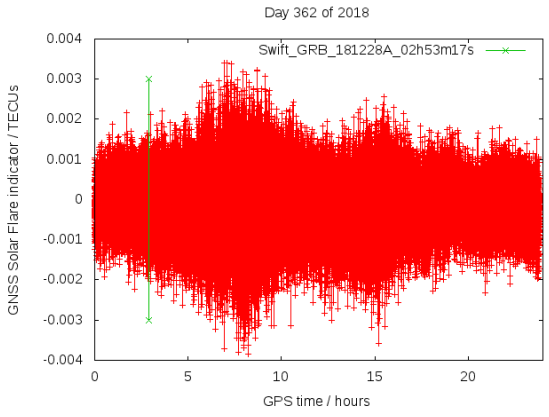


Figure 9.1: Result of the GSFLAI algorithm for the 181228A GRB

As we can see, although a small peak can be seen at the moment of the flare,

[1]until the moment of these tests

there are many other (and more significant) spikes throughout the plot, which does not indicate a clear detection of the flare.

9.2 Testing the BGSEES algorithm

As we have seen detecting stellar flares is a really challenging task. Even when knowing the location of the star and making the algorithm focus on that direction no clear results have been obtained. The aim of the project was to provide algorithms to detect Solar flares without knowing the location of the Sun so that it could be expanded to try this method without focusing on the direction of the star, but rather studying the available data and trying to find the source.

For this chapter two stellar flares were studied to test the algorithm and whether it could be possible to detect them using this method.

9.2.1 Discarding the day hemisphere

One of the first things that needed to be implemented in order to study stellar flares was a way to avoid the interferences of the Sun. Because the Sun would have a greater effect on the Earth's ionosphere, the day hemisphere is discarded.

This leaves the algorithm with only half the data, which could cause potential flares to be missed if they were having an effect on the day hemisphere.

This is done by simply taking the IPP's and Sun's right ascension and declination from the ti file and computing the cosine of their angle as has been done in previous chapters, using equations 3.2, 3.3 and 3.4.

If we want to discard the day hemisphere, then the IPP has to be ignored if the cosine of the angle, which has a range of [-1,1], is smaller than -0.2 (the point at which a linear correlation between the variables can be observed, as seen in chapter 3 (figure 3.4).

The following is the AWK script that performs this check for every IPP if we indicate the algorithm that it has to discard the hemisphere:

```
1  function checkValidIPP(raSun, decSun, raIPP, decIPP) {
2    return unitVectorsCosine(raSun, decSun, raIPP, decIPP) <= -0.2
3  }
4
5  function unitVectorsCosine(raSource, decSource, raIPP, decIPP) {
6    return (sin(decIPP)*sin(decSource) + cos(decIPP)*cos(decSource)
     *cos(raIPP - raSource));
7  }
```

Listing 9.1: Discarding the day hemisphere

The function returns true (the IPP is valid) if it is in the night hemisphere, that is, the cosine of its angle with the Sun is less or equal than -0.1.

With this change in the algorithm, it should be ready to try to detect stellar flares, rather than flares from the Sun. The following are some cases of strong stellar flares detected by dedicated telescopes that have been presented in research papers, using the information of them and with the help of the authors in some cases, the algorithm is tested to see if it can work with stellar flares or these are too weak to be detected without a dedicated telescope.

9.2.2 Proxima Centauri

The first studied flare is presented in the paper *"The first naked-eye superflare detected from Proxima Centauri"* (Howard, Ward S., et al.) [13], which, as the title of the paper suggests, was a flare so powerful it could even be seen with the naked eye. However, the flare could not be detected by pointing directly at it (knowing the location). This star is located in the Centaurus constellation, 4.22 light-years away

The flare was detected the 18th of March of 2016, with the flare peaking at **8:32 UT**. Having the ti file from this day in particular, 2016.078, we filtered the time range to obtain one hour surrounding that specific moment. Because this was a more challenging scenario, instead of finding the moment of the peak and only focusing on that time, the algorithm iterates over all epochs to see if any of them provided good results.

Taking into consideration that Proxima Centauri has a right ascension of 217.4294° and a declination of −62.67948° [25], the algorithm was executed using only the night hemisphere to try to detect this flare, using this location to calculate the error of the estimation.

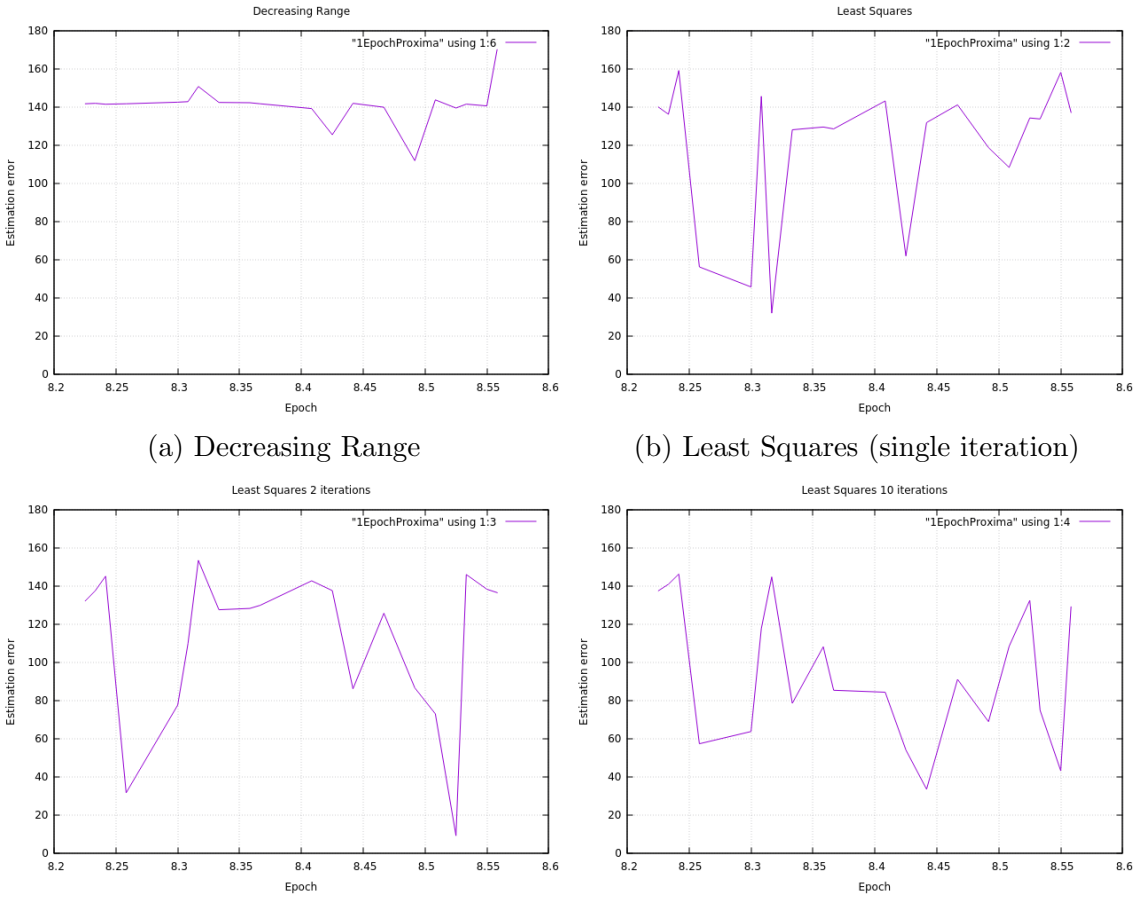
All of the following tests used a direct VTEC filter of 0.7. Furthermore, the day hemisphere is discarded using the code shown in the previous section.

Result of the 20 best epochs

The first tests consisted of computing the results for the **20 best epochs** using multiple combinations, the ones that yielded the best results where: Least Squares with 1, 2 and 10 iterations and Decreasing Range without Linear fit.

Figure 9.2 shows plots of the results of these tests in which the evolution of the error throughout the time range can be seen.

Although there is a lot of variation and spikes, some information can be drawn from this results. The Least Squares method using 2 iterations (figure 9.2c) shows a significant decrease in the estimation error with only 8.99495 degrees of error. This could be a coincidence but it is an interesting result because it is from the exact moment of the flare (8.525h). A decrease can be seen too in the results of the Decreasing Range method (figure 9.2a), although the error is still high and far



(a) Decreasing Range

(b) Least Squares (single iteration)

(c) Least Squares (2 iterations)

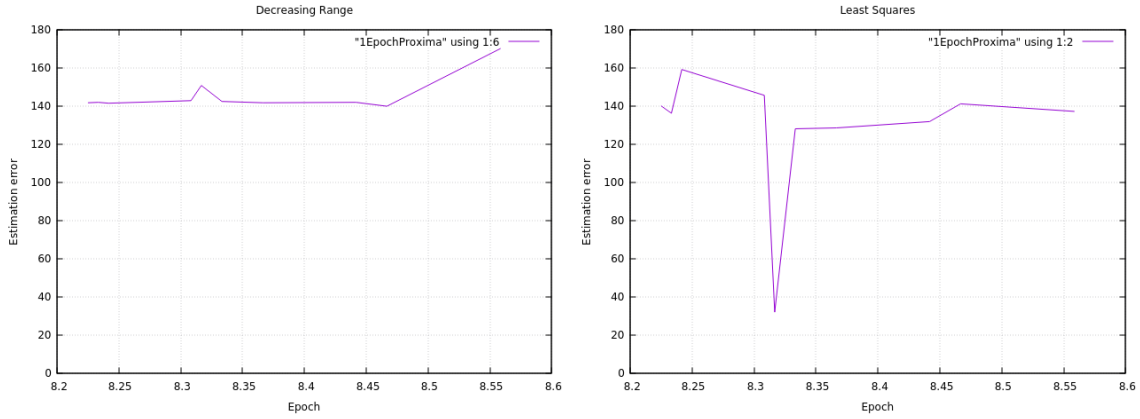
(d) Least Squares (10 iterations)

Figure 9.2: Estimation error surrounding the moment of the flare using 20 epochs (Proxima Centauri)

from the correct answer. The others present a lot of variation, because of this, we decided to instead use only the 10 best epochs.

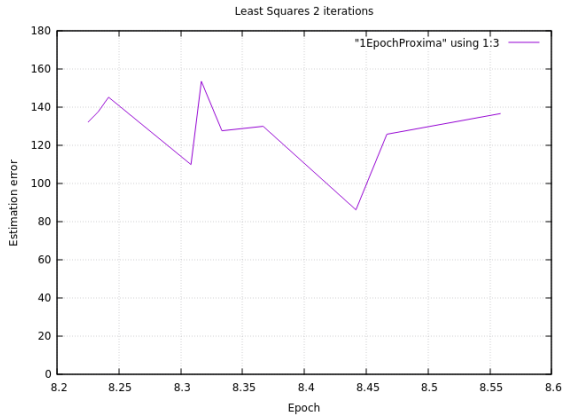
Result of the 10 best epochs

Because the estimation is not precise there appears a lot of variation if many epochs are used (many possible solutions) and not much information can be obtained from the results and the plot, however, if the number of used epochs is ten, we expected the change in the estimation to be seen more clearly. In this case, the Least Squares method with 3 iterations gave better results than using 10 iterations:

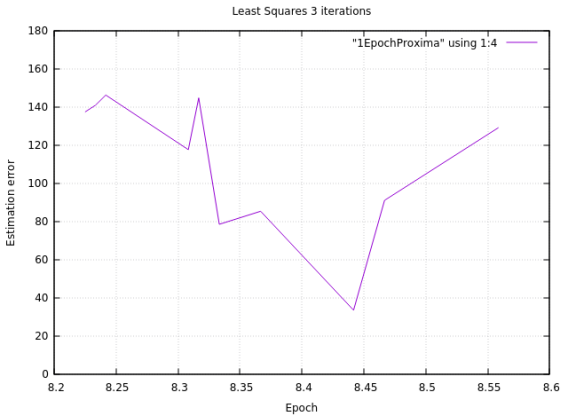


(a) Decreasing Range

(b) Least Squares (single iteration)



(c) Least Squares (2 iterations)



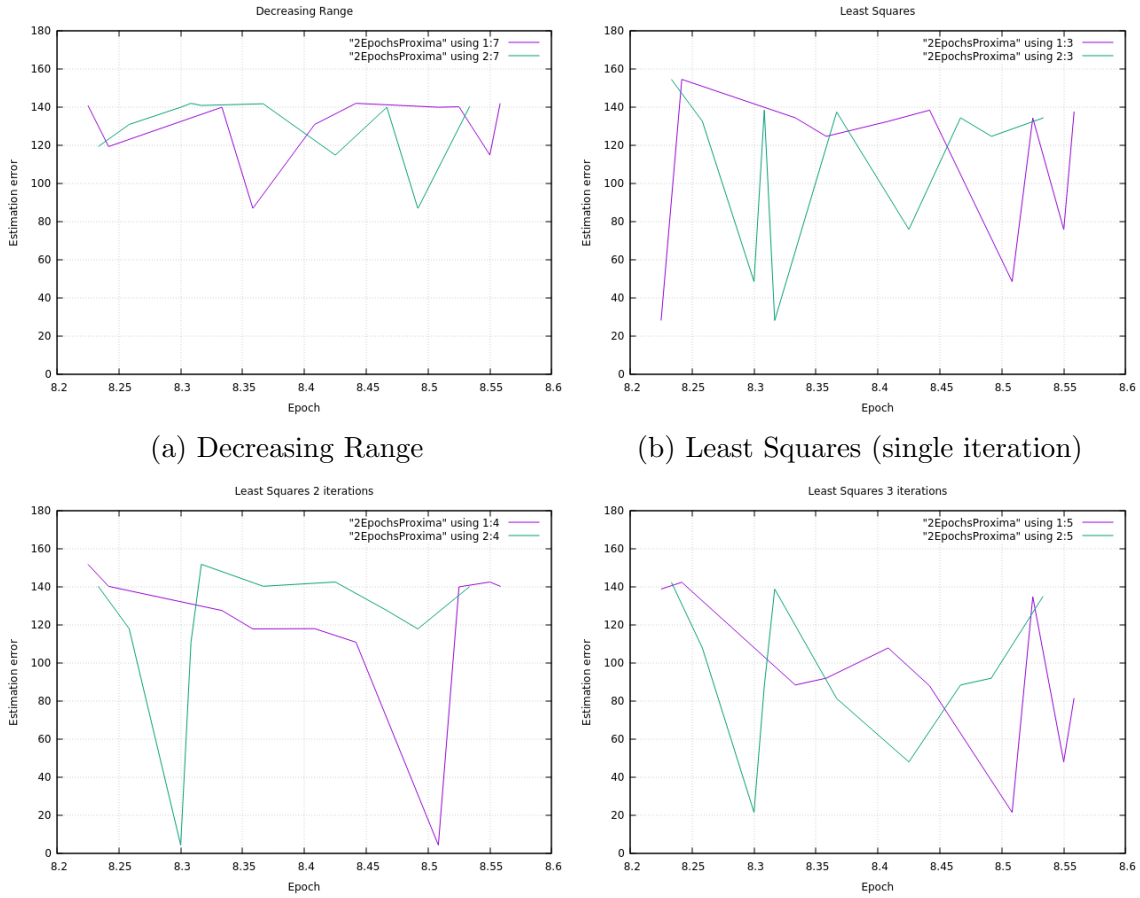
(d) Least Squares (3 iterations)

Figure 9.3: Estimation error surrounding the moment of the flare using 10 epochs (Proxima Centauri)

Although there is less noise in the plots and inverse spikes can be seen more clearly, none of them coincide with the exact moment of the flare and the results of the best epochs (ordered by their mean VTEC) are not the best results of the algorithm (in the previous test with 20 epochs, in fact, we have seen a result of only 8.99495 degrees for the moment of the flare).

Result of the 20 best epochs using 2 simultaneously

In the previous chapter the option of using the 2 best epochs was introduced, which yielded improved results for the tests done with the Sun. Using this method provides better results for this case as well. However, because the epochs are not ordered by time, but by their fitness, the 2 used epochs to compute a solution are not consecutive in time. Because of this the results of these tests show two functions: the results of the algorithm using the first of the 2 used epochs (purple) and the results using the second epoch (green).



(a) Decreasing Range

(b) Least Squares (single iteration)

(c) Least Squares (2 iterations)

(d) Least Squares (3 iterations)

Figure 9.4: Estimation error surrounding the moment of the flare using 20 epochs in groups of 2 (Proxima Centauri)

As we can see, although we obtain good results for some cases, such as the Least Squares method using 2 iterations, which gives a solution with only **4.15255 degrees of error** when using the epochs 8.5083h and 8.3h (the two inverse spikes seen in figure 9.4c). This time, the spike can also be seen more clearly with the

Decreasing Range method (9.4a) and the other Least Squares executions (9.4b and 9.4c).

The results for the Proxima Centauri flare are promising considering that the estimation that only has 4.15255 degrees of error is obtained when using data from an epoch near the moment of the event (8.5083h), leading us to believe that the algorithm has been slightly sensitive to the flare.

9.2.3 NGTS J121939.5-355557

Another studied stellar flare is presented in the paper *"Detection of a giant flare displaying quasi-periodic pulsations from a pre-main-sequence M star by the Next Generation Transit Survey"* (Jackman, James AG, et al.) [15].

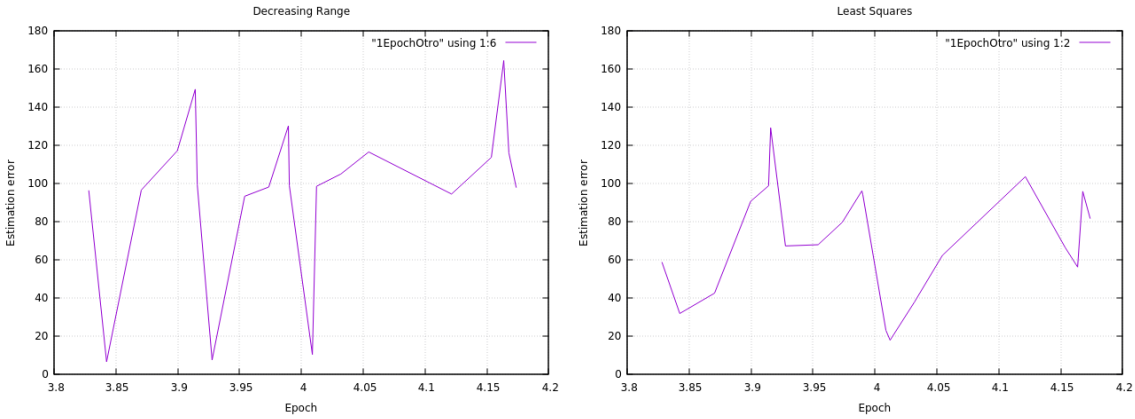
Although the date was specified in the paper (2016.032), the specific moment of the flare was not indicated. We got in touch with the authors of the paper, who kindly provided us information about the moment of the flare. The event was detected only at the moment of the peak (04:00 UT), so the start and end of the flare were unknown. Because of this, we worked with one hour of data surrounding the peak.

Several tests were performed with this flare. Generally speaking, none of the results provided an exact location estimation but we could observe some interesting results as with the previous flare.

Using the same parameters as the previous section (discarding the night hemisphere, using two epochs and using a direct VTEC filter), we proceeded to perform the same tests as the previous flare.

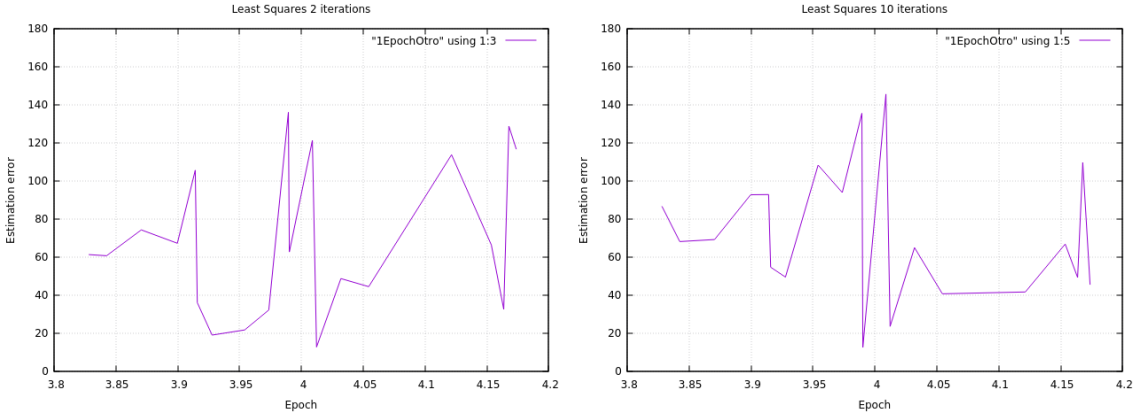
Result of the 20 best epochs

Although the results obtained present a lot of variation and spikes as with Proxima Centauri, we can notice that there is a significant improvement in the solution for the moment of the flare in all four cases from figure 9.5. However, other epochs present low estimation errors as well, with values as low as 12.5647 degrees.



(a) Decreasing Range

(b) Least Squares (single iteration)



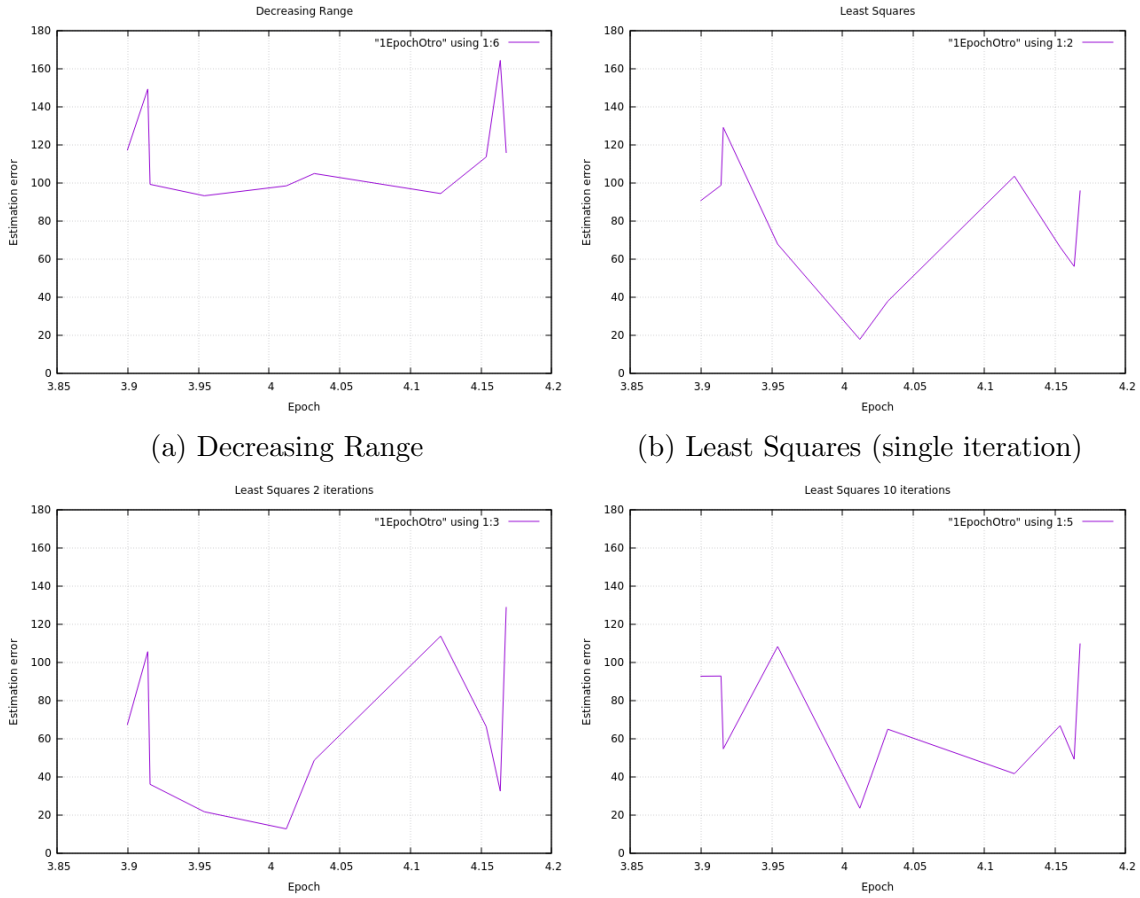
(c) Least Squares (2 iterations)

(d) Least Squares (10 iterations)

Figure 9.5: Estimation error surrounding the moment of the flare using 20 epochs (NGTS)

Result of the 10 best epochs

With the previous flare we performed the tests by plotting only the 10 best epochs to try to see the results more clearly, which we did in this case as well:



(a) Decreasing Range

(b) Least Squares (single iteration)

(c) Least Squares (2 iterations)

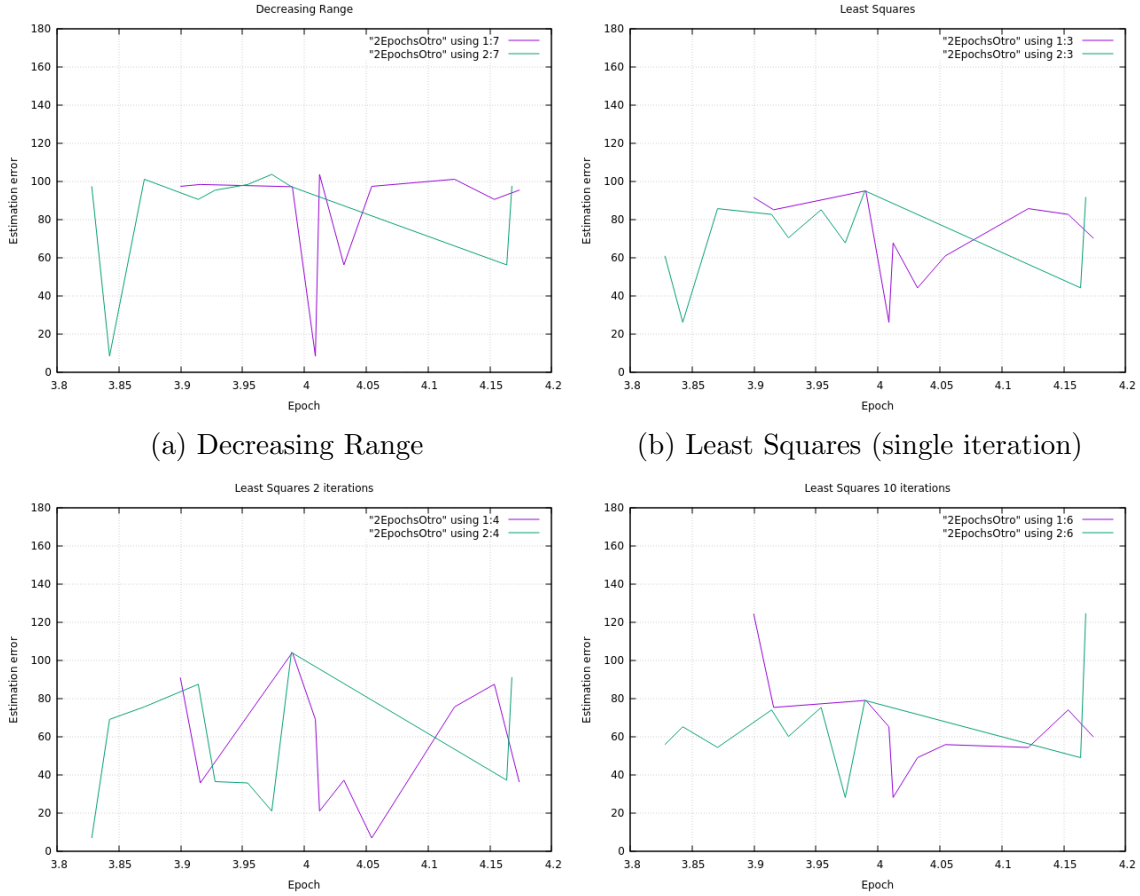
(d) Least Squares (10 iterations)

Figure 9.6: Estimation error surrounding the moment of the flare using 10 epochs (NGTS)

Although there is a high error for the Decreasing Range method (9.6a), we can see a sort of depression in the plot surrounding the moment of the flare. This depression is more clear in the other three tests using the Least Squares method (9.6b, 9.6c and 9.6d), yielding the best results near the moment of the flare.

Result of the 20 best epochs using 2 simultaneously

So far, the best results (both for solar and stellar flares) have been obtained when using 2 epochs simultaneously, and this method had interesting results for this flare as well.



(a) Decreasing Range

(b) Least Squares (single iteration)

(c) Least Squares (2 iterations)

(d) Least Squares (10 iterations)

Figure 9.7: Estimation error surrounding the moment of the flare using 20 epochs in groups of 2 (NGTS)

Similarly to the previous flare, the plot of the first used epoch (purple) is the one that presents a clear inverse spike at the moment of the flare, which is very noticeable in the Decreasing Range method (9.7a), in which the error of the estimation remains high (slightly below 100 degrees) surrounding the flare and is greatly reduced at the moment of the peak.

Both flares present a significant reduction in the estimation error at the exact moment of the flare, which could be due to the algorithm being able to estimate a

better solution due to the strength of the flare at that moment in time, leading us to think that the algorithm might be slightly sensitive to these two stellar flares.

Chapter 10

Conclusions

The main aim of the project was developing an algorithm capable of detecting solar flares without considering the location of the Sun, so that this could then be applied to the challenging scenario of stellar flares.

In the first chapters we studied the main factor that has to be taken into consideration in the algorithm: the correlation between the source-zenith angle's cosine and the VTEC value.

Then, an first approach of algorithm was developed to traverse the globe and consider possible Suns, the Decreasing Range method, which choose the best candidate based on its correlation.

An alternative method, the Least Squares method, was also developed. This approach solved an equation system to estimate the source's location, and yielded similar results.

Finally, both methods were tested with several solar flares using different parameters to study which one yielded better estimations, which provided positive results regarding the detection of solar flares. Using this information, the algorithm was tested with stellar flares as well.

This was a very challenging task due to stellar flares having a weaker effect on the ionosphere, but the algorithm has provided interesting results with the studied cases, where it seems to be sensible to the event, considering the estimation error is reduced at the moment of the flare.

10.1 Future work

The results obtained when studying the detection of stellar flares have presented an opportunity to proceed with the project, by improving on it and using the algorithm to test more flares to see if similar results are obtained.

In the next weeks more tests will be performed with other cases and a brief research paper presenting the results and new proposed methods will be written to

be submitted to the *"Geophysical Research Letters"* journal.

Furthermore, some of the proposals presented in chapter 7 will be implemented as well, such as the parallelization of some parts of the algorithm for increased performance.

In conclusion, the aim of the project was to develop an algorithm to be able to detect solar flares without knowing the location of the Sun to try to detect stellar flares and the tests performed with flares from outside the Solar System have provided promising results that have encouraged us to keep working on the project, which we hope can be of help to the scientific community in the future.

Chapter 11

Annexes

Poner aqui codigo de filemanager, debugger, makefile...?

Bibliography

[1] (Archived website) The Starlink Project. `https://web.archive.org/web/20120123062045/http://starlink.jach.hawaii.edu/starlink/WelcomePage`. [Online; accessed 10-March-2019].

[2] International GNSS Service Website. `http://www.igs.org/about`. [Online; accessed 8-March-2019].

[3] J. M. Dow, R. E. Neilan, and C. Rizos. The international GNSS service in a changing landscape of global navigation satellite systems. *Journal of geodesy*, 83(3-4):191–198, 2009.

[4] L.-L. Fu and A. Cazenave. *Satellite altimetry and earth sciences: a handbook of techniques and applications*, volume 69. Elsevier, 2000.

[5] N. Gehrels, G. Chincarini, P. Giommi, K. Mason, J. Nousek, A. Wells, N. White, S. Barthelmy, D. Burrows, L. Cominsky, et al. The swift gamma-ray burst mission. *The Astrophysical Journal*, 611(2):1005, 2004.

[6] N. Gehrels and S. Razzaque. Gamma-ray bursts in the swift-fermi era. *Frontiers of Physics*, 8(6):661–678, 2013.

[7] Goddard Space Flight Center (NASA). Swift GRBs Archive. `https://swift.gsfc.nasa.gov/archive/grb_table.html/`. [Online; accessed 1-March-2019].

[8] C. J. Hegarty and E. Chatre. Evolution of the Global Navigation Ssatellite System (GNSS). *Proceedings of the IEEE*, 96(12):1902–1917, 2008.

[9] M. Hernández-Pajares, A. García-Rigo, J. M. Juan, J. Sanz, E. Monte, and A. Aragón-Àngel. GNSS measurement of EUV photons flux rate during strong and mid solar flares. *Space Weather*, 10(12):1–16, 2012.

[10] M. Hernández-Pajares, J. Juan, J. Sanz, R. Orus, A. Garcia-Rigo, J. Feltens, A. Komjathy, S. Schaer, and A. Krankowski. The igs vtec maps: a reliable source of ionospheric information since 1998. *Journal of Geodesy*, 83(3-4):263–275, 2009.

[11] M. Hernández-Pajares, J. M. Juan, J. Sanz, À. Aragón-Àngel, A. García-Rigo, D. Salazar, and M. Escudero. The ionosphere: effects, gps modeling and the benefits for space geodetic techniques. *Journal of Geodesy*, 85(12):887–907, 2011.

[12] B. Hofmann-Wellenhof, H. Lichtenegger, and E. Wasle. *GNSS–global navigation satellite systems: GPS, GLONASS, Galileo, and more*. Springer Science & Business Media, 2007.

[13] W. S. Howard, M. A. Tilley, H. Corbett, A. Youngblood, R. P. Loyd, J. K. Ratzloff, N. M. Law, O. Fors, D. Del Ser, E. L. Shkolnik, et al. The first naked-eye superflare detected from proxima centauri. *The Astrophysical Journal Letters*, 860(2):L30, 2018.

[14] Institute of Space Sciences (CSIC-IEEC. Magnetar Outburst Online Catalog. `http://magnetars.ice.csic.es/#/parameters`. [Online; accessed 3-March-2019].

[15] J. A. Jackman, P. J. Wheatley, C. E. Pugh, D. Y. Kolotkov, A.-M. Broomhall, G. M. Kennedy, S. J. Murphy, R. Raddi, M. R. Burleigh, S. L. Casewell, et al. Detection of a giant flare displaying quasi-periodic pulsations from a pre-main-sequence m star by the next generation transit survey. *Monthly Notices of the Royal Astronomical Society*, 482(4):5553–5566, 2018.

[16] Jochen Greiner, Max-Planck-Institute for extraterrestrial Physics. GRB collection. `http://www.mpe.mpg.de/~jcg/grbgen.html`. [Online; accessed 3-March-2019].

[17] D. Martínez Cid. First study on the feasibility of stellar flares detection with gps. B.S. thesis, Universitat Politècnica de Catalunya, 2016.

[18] A. P. Mitra. Ionospheric effects of solar flares. In *Astrophysics and space science library*, volume 46, 1974.

[19] Netlib. LAPACK library. `http://www.netlib.org/lapack/`. [Online; accessed 29-April-2019].

[20] S. W. P. C. N. Oceanic and A. Administration). Ionosphere. `https://www.swpc.noaa.gov/phenomena/ionosphere`. [Online; accessed 25-March-2019].

[21] P. W. Roming, T. E. Kennedy, K. O. Mason, J. A. Nousek, L. Ahr, R. E. Bingham, P. S. Broos, M. J. Carter, B. K. Hancock, H. E. Huckle, et al. The swift ultra-violet/optical telescope. *Space Science Reviews*, 120(3-4):95–142, 2005.

[22] T. Singh, M. Hernandez-Pajares, E. Monte, A. Garcia-Rigo, and G. Olivares-Pulido. GPS as a solar observational instrument: Real-time estimation of EUV photons flux rate during strong, medium, and weak solar flares. *Journal of Geophysical Research: Space Physics*, 120(12):10–840, 2015.

[23] S. Solar Center. The Earth's Ionosphere. `http://solar-center.stanford.edu/SID/activities/ionosphere.html`. [Online; accessed 13-March-2019].

[24] Solar System Exploration, NASA Science. Reference Systems. `https://solarsystem.nasa.gov/basics/chapter2-2/`. [Online; accessed 5-March-2019].

[25] WikiSky. Proxima Centauri location. `http://server3.wikisky.org/starview?object_type=1&object_id=2392473`. [Online; accessed 3-June-2019].