UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Geodésica, Cartográfica y Topográfica

Evaluación de efectos del cambio climático empleando técnicas de geodesia espacial: Caso de variación de en altura de estaciones permanentes GNSS por equilibrio isostático

Trabajo Fin de Grado

Grado en Ingeniería Geomática y Topografía

AUTOR/A: Martínez Montes, Carlos

Tutor/a: Anquela Julián, Ana Belén

Cotutor/a: Marqués Mateu, Ángel

CURSO ACADÉMICO: 2021/2022

ESCUELA TÉCNICA SUPERIOR
DE INGENIERÍA GEODÉSICA
CARTOGRÁFICA Y TOPOGRÁFICA

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Evaluación de efectos del cambio climático empleando técnicas de geodesia espacial: Caso de variación en altura de estaciones permanentes GNSS por equilibrio isostático

Autor: Carlos Martínez Montes

Tutores: Ángel Marqués Mateu y Ana Belén Anquela Julián

Fecha: 30/08/2022

“El presente documento ha sido realizado completamente por el firmante; no ha sido entregado como otro trabajo académico previo y todo material tomado de otras fuentes ha sido convenientemente entrecomillado y citado su origen en el texto, así como referenciado en la bibliografía”

Resumen

Este proyecto trabaja la meta 13.1 del ODS13, acción por el clima. Generando una herramienta que permitirá “fortalecer la resiliencia y la capacidad de adaptación a los riesgos relacionados con el clima y los desastres naturales en todos los países”. Para contribuir con este objetivo, se ha trabajado con la variación de coordenadas de estaciones permanentes pertenecientes al IGS. Estas coordenadas pertenecen a un ajuste armonizado de la serie temporal 1 de enero del año 1995 hasta el 1 de enero del año 2020.

En la realización del estudio se emplea una herramienta de desarrollo propio; una aplicación informática creada al efecto que permitirá el procesamiento sistemático de un gran volumen de información obtenido de las estaciones permanentes. El desarrollo informático permitirá además crear una librería para el lenguaje Python que facilitará en un futuro a otros usuarios la obtención de información de cualquier estación permanente mundial en el lapso de tiempo deseado.

Para la realización de este proyecto se ha seguido una metodología que consta de los siguientes puntos:

- Descarga de datos
- Análisis y tratamiento de la información
- Obtención de resultados: mapas y gráficos
- Creación de una clase (estructura informática de la información) de forma similar a la estructura de cualquier librería Python

Índice de figuras

Índice de tablas

Índice

1. Introducción

1.1. Finalidad

La finalidad del proyecto se separa en dos ámbitos. Por un lado, comprobar si debido los efectos del cambio climático, las estaciones permanentes GNSS cercanas a los polos, ha habido una aceleración en su variación en altura por la pérdida de hielos en los últimos años. Por el otro, automatizar este método de trabajo desarrollando una herramienta multiplataforma que en el futuro pueda emplear cualquier usuario que desee realizar este tipo de estudios lo pueda hacer de una forma sencilla y eficiente.

1.2. Descripción de antecedentes

Este proyecto está basado en un proyecto anterior “Estudio de la evolución temporal de la posición de las estaciones GNSS cercanas al Polo Norte” realizado por Lourdes Monfort Bolufer como proyecto final para la obtención de su título de Ingeniera en Geodesia y Cartografía. Las diferencias con el anterior proyecto se encuentran en el mayor periodo de tiempo estudiado y en la creación de una herramienta multiplataforma de desarrollo propio para la obtención y tratamiento de los datos necesarios para el estudio.

1.3. Conceptos previos:

1.3.1. Estaciones permanentes:

Las estaciones permanentes GNSS están compuestas por un receptor geodésico capaz de captar las señales de las diferentes constelaciones a su alrededor, más un sistema informático para procesar la información. Las observaciones realizadas por los receptores son almacenadas como ficheros de datos y publicados posteriormente.

El uso principal (datos recopilados y publicados por las estaciones permanentes de GPS) lo proporcionan los profesionales que utilizan receptores de satélites GPS para las mediciones, quienes a través del procesamiento posterior de las observaciones pueden mejorar la precisión (y, por lo tanto, la precisión de las coordenadas obtenidas), en algunos casos. hasta centímetros, y un enlace a un único sistema de referencia geodésica (georreferenciación).

También el uso de estas como parte de una red de estaciones permanentes sirven para realizar un cálculo de correcciones, para que puedan hacer uso de ella cualquier usuario de la tecnología GNSS y hasta los dispositivos tecnológicos de uso común denominados navegadores las usan para mejorar su precisión de los 10m hasta alcanzar los 1-2m.

En cambio, sí se habla del uso científico de estaciones, que fue en primer lugar, la razón de las primeras estaciones de este tipo, se encuentra una gran variedad de campos desde estudios atmosféricos, observar la geodinámica del planeta y observaciones muy precisas de tiempo.

1.3.2. Observaciones GNSS

Los aparatos de recepción GNSS permiten el registro de una serie de observables que se obtienen del rastreo de las diferentes señales enviadas por los satélites GNSS. Ahora se procederá a enumerar los observables básicos que un receptor de este tipo puede generar:

- En primer lugar, el receptor guarda el tiempo en el que recibe las señales utilizando el reloj interno del mismo. Al hacer esto todas las observaciones realizadas por el receptor están referenciadas temporalmente sin importar la señal GNSS recibida por parte de los satélites.
- Por otro lado, el receptor recibe las ondas portadoras de los satélites GNSS. Para ello realiza una medición de los ciclos enteros de las fases generadas par las señales GNSS registradas mediante la comparación de estas con unas replicadas por el mismo receptor.
- También realiza un cálculo de la pseudodistancia entre la antena del receptor y el satélite GNSS. Su cálculo se realiza por medio del tiempo que tarda en llegar la señal desde el satélite hasta el receptor. Se multiplica este tiempo por la velocidad de la luz en el vacío obteniendo así la pseudodistancia.

1.3.2.1. Efemérides

Las efemérides son los ficheros de la posición orbital de un satélite en concreto generados en un instante concreto del tiempo, solo pudiéndose obtener tras realizar una observación desde una estación permanente y procesar los datos obtenidos.

1.3.3. Ficheros SINEX

Los ficheros SINEX son los ficheros donde a través de una corrección realizada a posteriori se almacenan las ubicaciones precisas de todas las estaciones GNSS que hay en el mundo. Esta posición se registra diariamente por diferentes organismos que se encargan de realizar estos cálculos como son el IGS o la NASA, entre otros. Para ello utilizan los datos obtenidos de estas para realizar una red geodésica y por medio de métodos estadísticos como mínimos cuadrados realizar una corrección en las observaciones obteniendo en algunos casos errores por debajo de los 2mm en latitud, longitud y altura.

1.3.4. Red mundial del IGS

El IGS es una organización científica voluntaria fundada en enero de 1994. Inicialmente se denominó Servicio GPS Internacional renombrándose en el año 2005 a *International GNSS Service* (IGS) una vez ingresaron otros sistemas como GALILEO (planificado por Unión Europea) y GLONASS en la escena del posicionamiento global. Actualmente el IGS es un servicio de la Asociación Internacional de Geodesia (IAG) y participan unas 200 organizaciones de aproximadamente 80 países, siendo su misión principal capturar, archivar y distribuir datos y productos asociados GPS y GLONASS de alta calidad. Para ello el International GNSS Service emplea datos de observaciones de estaciones de referencia GNSS de múltiples organismos o instituciones de todo el mundo, formado de este modo una red mundial de unas 300 estaciones de referencia GNSS, en varias de ellas incluso en tiempo real, a partir de la cual realiza un seguimiento continuo de los satélites de las constelaciones GNSS.

Los datos son utilizados posteriormente para proporcionar los siguientes productos:

- órbitas y relojes precisos;
- parámetros de rotación terrestre;
- datos troposféricos e ionosféricos; y
- coordenadas y velocidades.

Esta información permite dar apoyo a mediciones geodésicas de alta precisión utilizadas en diversas actividades científicas y prácticas tales como la geodesia, geodinámica, ingeniería, investigaciones oceanográficas y atmosféricas, y levantamientos y cartografía.

Además, los productos del IGS son utilizados en la mejora y densificación del Marco de Referencia Terrestre Internacional (ITRF), infraestructura mantenida por el Servicio Internacional de Sistemas de Referencia y Rotación Terrestre (IERS). Este hecho es de especial relevancia en el cálculo de coordenadas de redes de estaciones de referencia GNSS

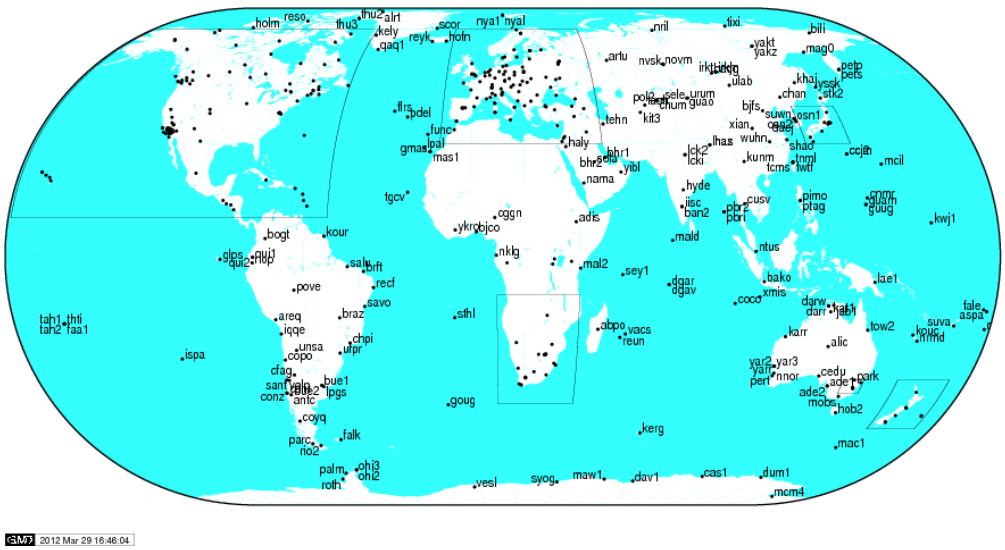


Figura 1: Red de estaciones IGS

Las informaciones de rastreo de los satélites se ponen a disposición diariamente en archivos Rinex. Todos los puntos generan por lo menos un archivo con mensajes de observación y navegación, mientras que algunos puntos generan también datos meteorológicos.

Aunque la mayor parte de los productos son empleados por instituciones académicas y de investigación, las órbitas y relojes precisos de los satélites y los datos RINEX son muy utilizados en aplicaciones de topografía, como por ejemplo en servicios de Posicionamiento Puntual Preciso (PPP)

Los archivos que el sitio del IGS pone a disposición son los siguientes:

- Rinex (observación GPS y Glonass, datos meteorológicos y de navegación)
- Sinex (posición y velocidad de las estaciones)
- Hatanaka (observaciones GPS y Glonass)
- sp3 (órbita de los satélites GPS y Glonass)
- erp (parámetros de rotación de la Tierra)
- clock Rinex (reloj de la estación y de los satélites)
- Ionex (ionosfera)
- Tropo Sinex (estimativa de atraso cenital)
- site log (histórico del punto)
- Antex (calibración de la antena)

Es importante destacar que todos los productos del IGS son de libre disponibilidad para cualquier propósito, sea éste científico o comercial, de acuerdo a la política de “datos abiertos” que mantiene la organización. Una lista completa de los productos puede encontrarse en: http://www.igs.org/components/prods.html.

Para la descarga de los datos, debemos tener en cuenta las siguientes consideraciones:

1.3.4.1. Identificación de la estación

En la Figura anterior aparece el conjunto de estaciones de la red IGS. La estación viene identificada de forma única por una sigla de cuatro letras. Designado el punto de interés, el archivo log de la estación puede bajarse desde el link *ftp://igscb.jpl.nasa.gov/igscb/station/log*, con el siguiente formato:*ssssmmyy.log*, donde:

- *ssss* ,es el código de la estación
- *mm* ,es el mes de creación o actualización del archivo de log
- *yy* ,el año.

1.3.4.2. Identificación del centro de datos

Antes de bajar los datos, se deben verificar los posibles Centros de Datos (global, regional, etc.) que están realizando el rastreo de la estación en estudio en el período de tiempo deseado. Los datos están disponibles en el link http://igscb.jpl.nasa.gov/components/data.html, y los datos anuales presentados se encuentran en el formato *globyyyy.syn* y los mensuales en el formato *globmmyy.syn*. Los datos de los centros se encuentran en el formato *center_name.syn* para el año corriente y en el *center_name.syy* para cualquier año.

1.3.4.3. Descarga de los datos

Para bajar los datos, hay que acceder al Centro de Datos vía ftp o http, dirigirse al directorio correspondiente y descargar los archivos que sean necesarios.

1.3.5. Estaciones permanentes EUREF

EUREF es el nombre con el que se conoce a la Subcomisión de Marcos de Referencia en Europa, integrada en la Comisión 1 de Marcos de Referencia perteneciente a la Asociación Internacional de Geodesia. Los objetivos de EUREF son la definición, implantación y mantenimiento de un marco de referencia geodésico común para todo el continente europeo. La Red GNSS Permanente de EUREF está formada por:

- una red de estaciones de referencia GNSS (Global Navigation Satellite Systems, como GPS, GLONASS, Galileo, Beidou,..) en Funcionamiento continuo,
- centros de datos que proporcionan acceso a los datos de la estación,
- centros de análisis que analizan los datos GNSS,
- centros de productos o coordinadores que generan los productos de la EPN,
- y una Dirección Central que se encarga del seguimiento y manejo diario de la EPN

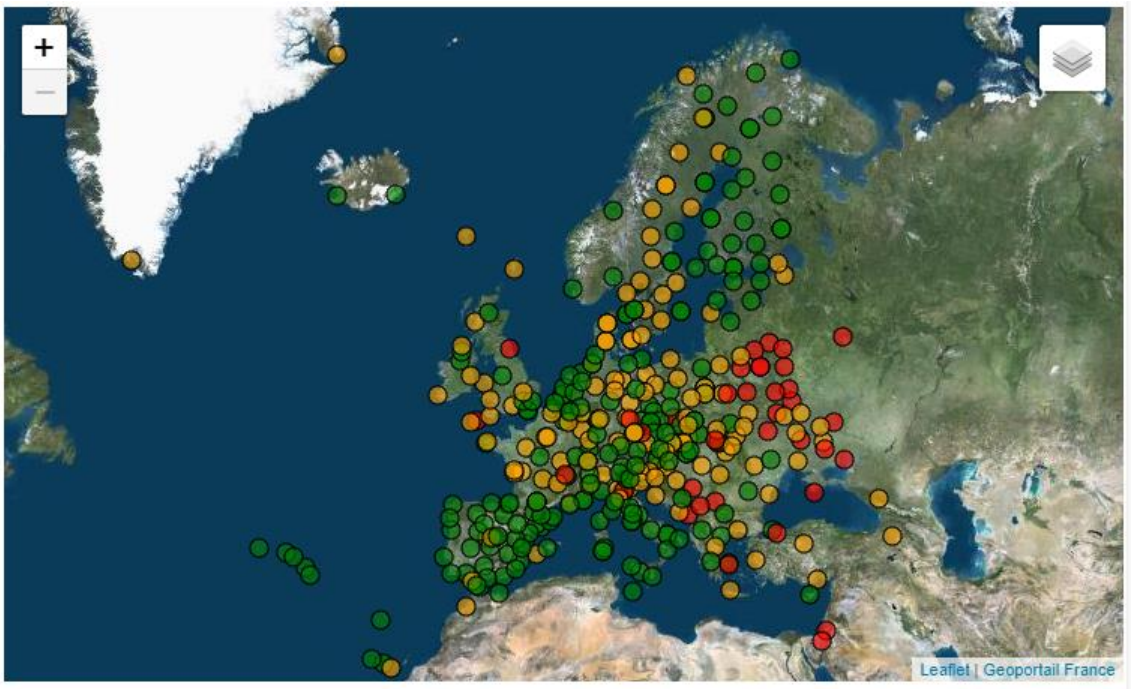


Figura 2: Mapa EUREF

Todas las contribuciones a la EPN son voluntarias, con más de 100 agencias/universidades europeas involucradas. La EPN opera bajo estándares y objetivos internacionales bien definidos y suscritos por sus colaboradores. Estos objetivos garantizan la calidad a largo plazo de los productos EPN

El objetivo principal de la EPN es proporcionar acceso al Sistema de Referencia Terrestre Europeo 89 (ETRS89), que es el sistema de

coordenadas GNSS preciso estándar en toda Europa. Con el apoyo de EuroGeographics y avalado por INSPIRE (D2.8.I.1 Data Specification on Coordinate Reference Systems), el ETRS89 constituye la columna vertebral de los datos de geolocalización en el territorio europeo, tanto a nivel nacional como internacional.

La EPN brinda acceso al ETRS89 al poner a disposición del público los datos de seguimiento del GNSS, así como las posiciones, velocidades y parámetros troposféricos precisos de todas las estaciones de la EPN. Basándose en estos productos, la EPN también contribuye al seguimiento de las deformaciones tectónicas en Europa y apoya el seguimiento del clima a largo plazo, la predicción numérica del tiempo y el seguimiento de las variaciones del nivel del mar.

EUREF trabaja en estrecha colaboración con diferentes organismos cartográficos, catastrales y registros de la tierra de cada uno de los países europeos siendo éstos los que a la postre trasladan a sus respectivos países las definiciones del sistema de referencia geodésico adoptado por EUREF.

1.3.6. Sistema de Referencia

En este trabajo se ha utilizado el Sistema de coordenadas geodésicas WGS84 de EPSG:4326.

De este modo, será necesario la conversión de coordenadas geocéntricas a geodésicas. A pesar de tratarse de una cuestión habitual no existe una solución estándar por lo que se emplearán las fórmulas citadas en el artículo por Luis Garcia-Asenjo Villamayor desarrolladas en Meyer, T.H.: “Introduction to Geometrical and Physical Geodesy: foundations of geomatics”

Como describe Luis Garcia-Asenso en su artículo “*Se trata de una solución basada exclusivamente en conversiones de coordenadas, rotaciones y traslaciones que no deforman las figuras ni introduce variaciones en las distancias manteniendo intacta la geometría de los puntos levantados*” L.:“ **Conversión entre coordenadas geodésicas y coordenadas locales**”, Ed. Universidad Politécnica de Valencia

1.3.7. Transformación de coordenadas cartesianas geocéntricas a geodésicas.

Para esta transformación se ha seguido el proceso inverso de conversión de coordenadas locales a geodésicas. Las fórmulas empleadas son las siguientes:

- Incrementos de coordenadas locales respecto al origen $(x_0, y_0, z_0)^T$ del sistema local:

$$\begin{pmatrix} \Delta x \\ \Delta y \\ \Delta z \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} - \begin{pmatrix} x_0 \\ y_0 \\ z_0 \end{pmatrix}$$

- Rotación α de los incrementos $(\Delta x, \Delta y, \Delta z)^T$ para convertirlos a incrementos en un sistema local $(e, n, u)^T$ orientado al norte geodésico y origen $(0,0,0)^T$:

$$\begin{pmatrix} e \\ n \\ u \end{pmatrix} = \begin{pmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \Delta x \\ \Delta y \\ \Delta y \end{pmatrix}$$

- Conversión de los incrementos $(e, n, u)^T$ a incrementos de coordenadas ECEF empleando para ello las coordenadas geodésicas del origen $(\varphi_0, \lambda_0, h_0)^T$

$$\begin{pmatrix} \Delta X \\ \Delta Y \\ \Delta Z \end{pmatrix} = \begin{pmatrix} -\sin\lambda_0 & -\sin\varphi_0\cos\lambda_0 & \cos\varphi_0\cos\lambda_0 \\ \cos\lambda_0 & -\sin\varphi_0\sin\lambda_0 & \cos\varphi_0\sin\lambda_0 \\ 0 & \cos\varphi_0 & \sin\varphi_0 \end{pmatrix} \begin{pmatrix} e \\ n \\ u \end{pmatrix}$$

- A los Incrementos $(\Delta X, \Delta Y, \Delta Z)^T$ se le suman las coordenadas ECEF del origen $(X_0, Y_0, Z_0)^T$ obtenidas a partir de las geodésicas $(\varphi_0, \lambda_0, h_0)^T$ mediante las expresiones:

$$\begin{aligned} X_0 &= (\nu_0 + h_0)\cos\varphi_0\cos\lambda_0 \\ Y_0 &= (\nu_0 + h_0)\cos\varphi_0\sin\lambda_0 \\ Z_0 &= \left[(1-e^2)\nu_i + h_0\right]\sin\varphi_0 \end{aligned}$$

, en las que $\nu_0 = \frac{a}{\sqrt{1-e^2\,sin^2\varphi_0}}$ es el radio de curvatura de la sección normal del primer vertical y los parámetros *a* y *e* son respectivamente el semieje mayor y la primera excentricidad del elipsoide de referencia.

- Finalmente, las coordenadas ECEF se convierten a coordenadas geodésicas mediante:

$$\varphi = arctg\frac{Z + e'^2 b sen^3\theta}{p - e^2 a\cos^3\theta} \qquad \lambda = arctg\frac{Y}{X} \qquad h = \frac{p}{\cos\varphi} - \nu$$

$$\theta = arctg\frac{Za}{pb} \qquad e'^2 = \frac{a^2 - b^2}{b^2} \qquad p = \sqrt{X^2 + Y^2}$$

, en las que a y b son respectivamente el semieje mayor y menor del elipsoide de referencia.

1.3.8. Placas tectónicas

Las placas tectónicas son las estructuras por las cuales se conforma nuestro planeta. Son grandes masas de roca de la corteza terrestre que se dividen en varias

secciones y que se desplazan muy lentamente impulsadas principalmente por el calor interno del planeta.

En términos geológicos se comportan como una placa rígida de roca sólida en la superficie de la tierra (litosfera), de un grosor entre 15 y 200 km, flotando sobre roca ígnea y fundida que conforma el centro del planeta (astenosfera).

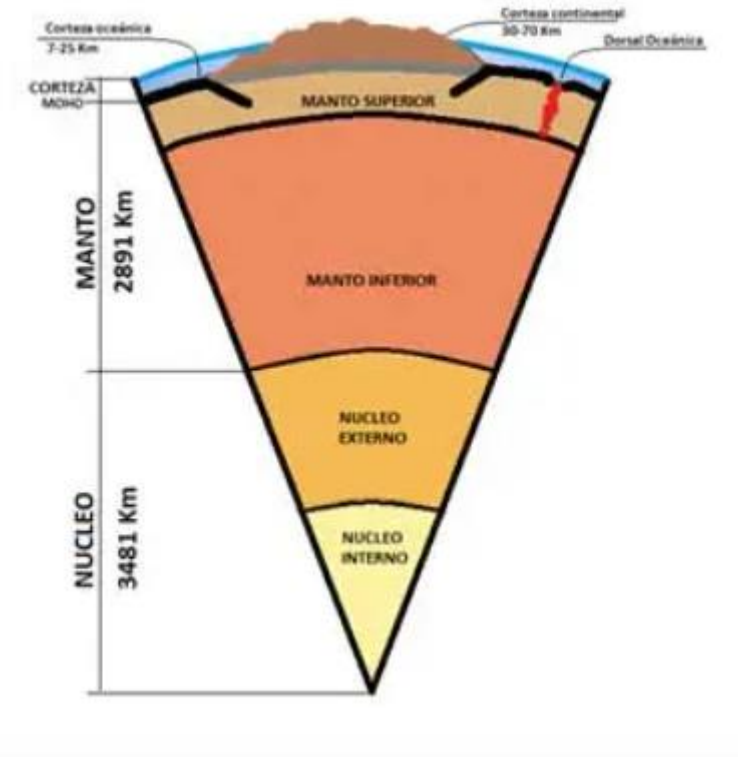


Figura 3: Estructura de la tierra

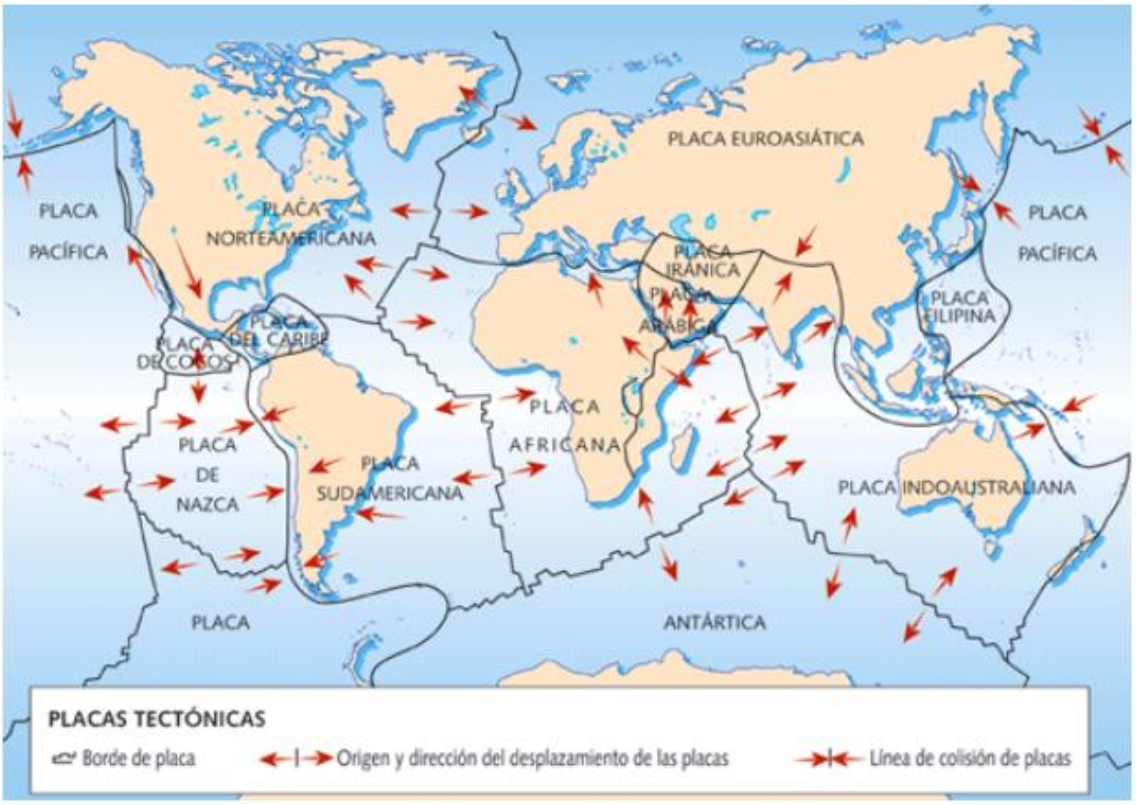


Figura 4: Distribución superficial de las placas tectónicas

Hay varias placas principales que suelen recibir el nombre del continente en el que se encuentran y otras muchas secundarias, más pequeñas, pero no menos importantes.

La actividad geológica tiene su origen en la interacción de las placas cuando se acercan o se separan. El movimiento crea tres tipos de límites tectónicos:

- Límites convergentes donde las placas se acercan unas a otras de modo que cuando colisionan la corteza se comba dando lugar a las cordilleras que continúan elevándose mientras el choque continúa. Estos límites también tienen lugar cuando una placa oceánica se hunde bajo otra continental en un proceso denominado “subducción”. La placa superior se eleva dando lugar a sistemas montañosos y la inferior se funde provocando erupciones volcánicas.
- Límites divergentes en los que el magma surge en la superficie desde las profundidades del manto de la tierra separando dos o más placas y renovando el fondo oceánico; así, una dorsal marcaría la separación entre placas.
- Límites transformantes en los que dos placas friccionan una con otra a lo largo de las denominadas fallas de desgarre. Estos límites no crean fenómenos espectaculares como montañas u océanos, pero pueden dar lugar a terremotos devastadores como el de 1906 en San Francisco por la Falla de San Andrés.

1.3.9. Mareas terrestres

Las mareas, tanto de las partes líquidas como sólidas y gaseosas, son las manifestaciones de los efectos que sobre un cuerpo origina la acción de las fuerzas que derivan de un campo potencial.

Dada su influencia sobre el comportamiento de las coordenadas de las estaciones es importante conocer su concepto básico.

Son el resultado de la combinación de cuatro efectos gravitatorios claramente observables sobre la masa de agua de los océanos:

- la atracción gravitatoria de la Luna (nuestro satélite natural, situado a un promedio de 300.000 Km de la Tierra)
- la atracción gravitatoria del Sol (situado a unos 150.000.000 Km de la Tierra)
- la velocidad de rotación de la Tierra (aproximadamente 360º/día)
- la presencia de los continentes

El origen de la fuerza de las mareas se debe a que la Tierra es un cuerpo extenso y el campo gravitatorio que producen la Luna o por el Sol no es homogéneo en todo el planeta, ya que hay zonas que están más cerca y otros más alejados de dichos cuerpos celestes.

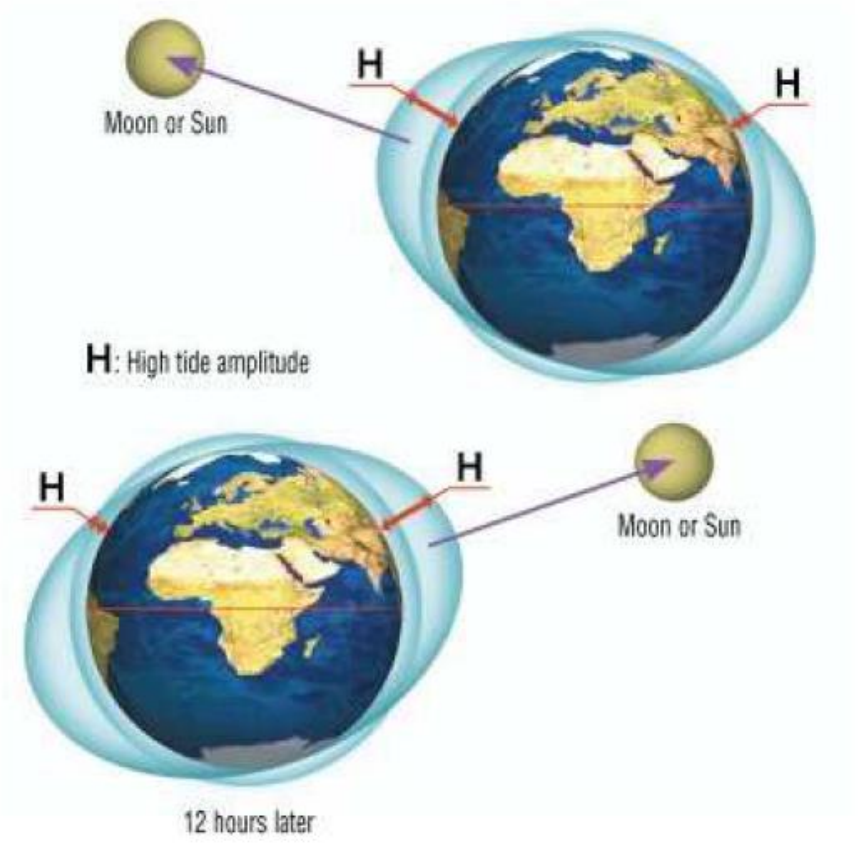


Figura 5: Mareas terrestres

En el caso de las mareas terrestres, habría que hacer una pequeña distinción con las mareas oceánicas. La tierra al no ser completamente rígida, debido a la propia heterogeneidad de la corteza y a la viscosidad y elasticidad del planeta, generan deformaciones de tipo elástico que afectan a la amplitud de este fenómeno.

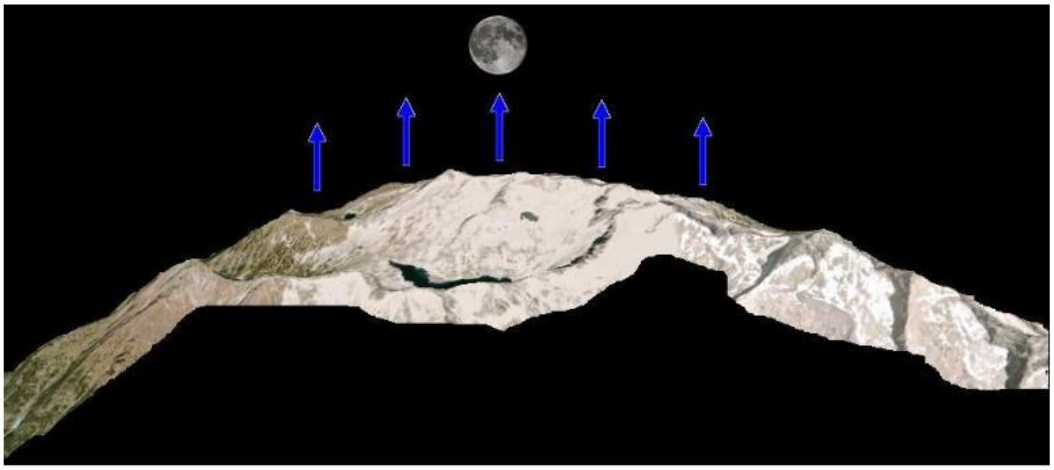


Figura 6: Movimientos terrestres influenciados por la luna

Actualmente se han desarrollado gran variedad de instrumentación científica capaz de detectar los efectos de las mareas terrestres.

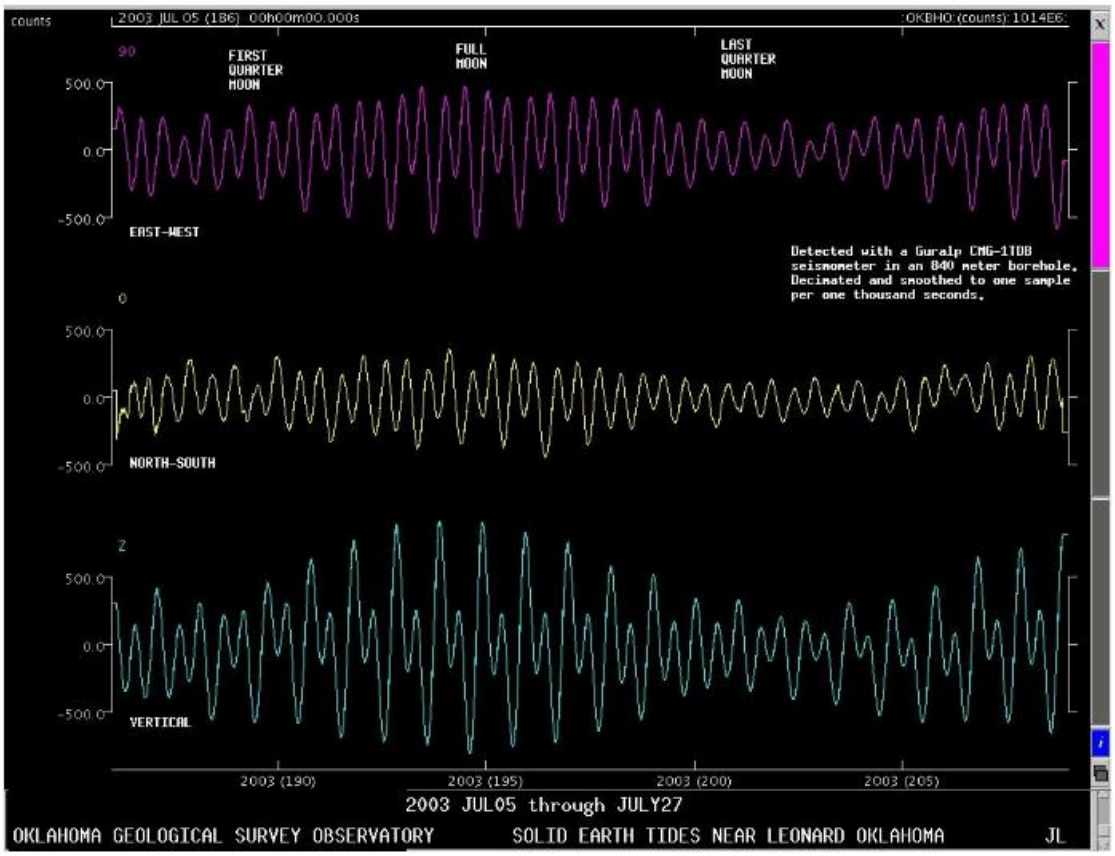


Figura 7: Estudio mareas terrestres cerca de Leonardo (Oklahoma)

En España disponemos de dos ejemplos de laboratorios donde se realizan mediciones de la marea terrestre, el Laboratorio Geodinámico del Valle de los Caídos (Madrid) y el Laboratorio Subterráneo de Canfranc que es una instalación excavada en la roca, 850 metros de profundidad en la vertiente españolas de los Pirineos Aragoneses.En el primer caso, en servicio desde 1975, disponemos de registros de ciclos completos de marea de 20 años.

La magnitud de los efectos de marea es de varias decenas de cm, llegando en algunas zonas del planeta a más de 30cm, aunque en el ecuador debido a la acción del Sol, suelen estar en torno a 15cms.

Por último, en ciertas ocasiones, tanto las mareas terrestres como las oceánicas, se las han relacionado a la actividad volcánica y sísmica, aunque sigue estudiándose si hay una verdadera relación entre estas.

1.3.10. Movimientos isostáticos

El término **isostasia** (del griego *ísos*, 'igual',*stásis*, 'paralización') se refiere a la condición de equilibrio gravitacional de la zona externa de la geosfera. Esta parte del planeta que incluye la corteza y el manto presenta diferencias de altitud compensando así las diferencias de densidad entre las distintas áreas. La condición de equilibrio se resuelve con movimientos verticales (epirogénicos) y se fundamenta en el principio de Arquímedes.

Este equilibrio puede romperse por distintos procesos como los movimientos tectónicos o la fusión de casquetes glaciales que pueden alterar la densidad de la corteza terrestre.

La isostasia contribuye a conformar el relieve de la Tierra. Partiendo de que los continentes son menos densos que el manto y la corteza oceánica, los pliegues de la corteza continental acumulan gran cantidad de materiales en regiones concretas. Una vez finaliza el ascenso en una zona, comienzan los procesos erosivos en la corteza. Los materiales van depositándose fuera de la cadena montañosa perdiendo peso y volumen. Compensando esta pérdida, se elevan las raíces y dejan en la superficie materiales sometidos a un mayor proceso metamórfico convertidos en granito. El granito en superficie forma escudos y macizos más rígidos que no se pliegan ante una nueva orogenia y que acaban rompiéndose dando lugar a un relieve fallado. Estos bloques en los que se fragmenta el escudo también tienden a alcanzar el equilibrio isostático y los reajustes entre bloques dan lugar a pequeños terremotos.

Ya en 1735 Pierre Bouguer realiza observaciones de la deflexión de la vertical menores a las esperadas, fenómeno que también verificó posteriormente George Everest en un levantamiento topográfico de la India. De estos resultados surgió la idea de que alguna compensación, con un contraste negativo de densidad, debía existir debajo de la topografía visible lo que condujo al concepto de isostasia que asume el equilibrio de una columna de la tierra hasta cierto *nivel de compensación*.

La condición de equilibro isostático se plantea como:

$$\int_{-T_o}^{H} \rho \, dz = Cte.$$

Donde *To* es la profundidad de compensación, *H* la altura de la topografía y ρ la densidad. La expresión establece que hay un nivel de compensación *To* por encima del cual el peso de todas las columnas imaginarias de corteza y manto es constante. La condición se cumple en la tierra para valores de *To* de pocos cientos de kilómetros. Si el peso de dos columnas fuera distinto, el manto se desplazaría hasta equilibrarlos, alcanzando el equilibrio isostático.

Ante el desconocimiento de las densidades del interior terrestre se desarrollaron dos modelos casi de manera simultánea; el modelo de Henry Pratt propone una profundidad de compensación constante *To* y variaciones de la topografía asociadas a cambios laterales en la densidad, mientras que el de George Airy asume densidad constante y una profundidad de compensación variable.

Actualmente existen tres modelos isostáticos: Pratt-Hayford, Airy-Heinskanen y Vening Meinesz, los cuales se pueden consultar (Heiskanen & Moritz, 1985).

2. Objetivos

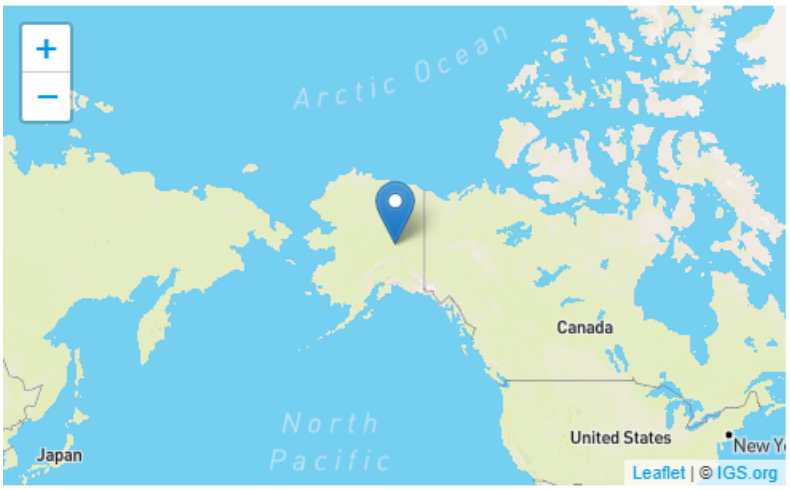
Como objetivos generales se tiene en primer lugar el estudio de la influencia que tiene el efecto del cambio climático sobre el equilibrio isostático de la corteza terrestre, a partir de la variación de coordenadas de estaciones permanentes GNSS. En segundo lugar, la creación de un método para automatizar el proceso de captura y análisis de la información. La consecución de estos objetivos requiere las siguientes tareas:

- Desarrollar una aplicación informática multiplataforma que permita procesar un gran volumen de información de forma eficiente, mediante el lenguaje de programación Python.
- Facilitar a terceros el método de adquisición y procesamiento de datos en una librería accesible desde Python que sistematice y simplifique el proceso en otros trabajos de investigación estructurando la información en una “clase” y desarrollando las funciones de programa necesarias.
- Desarrollar un método de geovisualización que optimice el análisis de resultados (caso para placa euroasiática).
- Análizar distintos métodos de geovisualización de resultados (gráficas y video animado).

3. Datos

- Ficheros SINEX de la web de la NASA https://cddis.nasa.gov/archive/gnss/products/ desde el año 1995 hasta el 2020.
- Estaciones permanentes GNSS estudiadas en el proyecto:

- Estación FAIR.

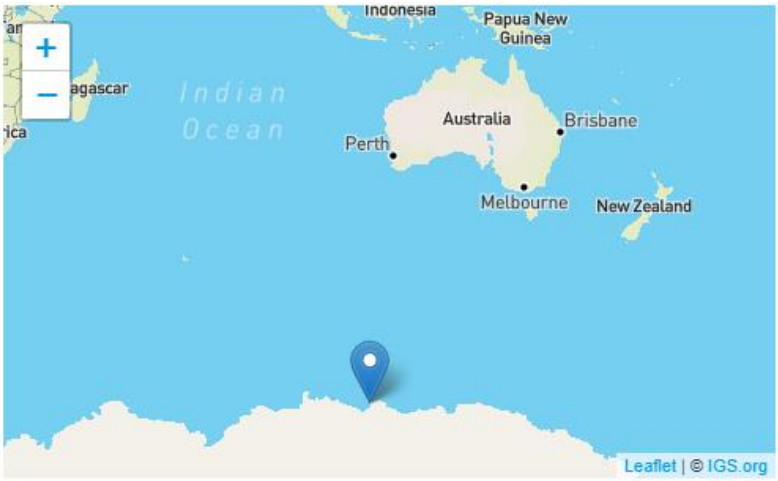


Country/Region	Fairbanks, United States
Latitude, Longitude	64.978, -147.499
Elevation	319.1771 m

Figura 8: Estación FAIR localización

Figura 9: Foto de la estación FAIR

- Estación CAS1.

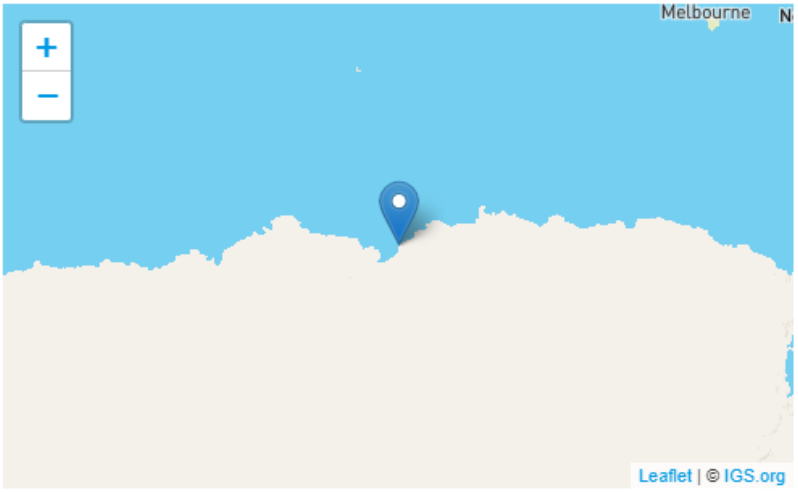


Country/Region	Casey, Antarctica
Latitude, Longitude	-66.283, 110.520
Elevation	22.6 m

Figura 10: Estación CAS1 localización

Figura 11: Foto de la estación CAS1

- Estación DAV1.



Country/Region	Davis, Antarctica
Latitude, Longitude	-68.577, 77.973
Elevation	44.5 m

Figura 12: Estación DAV1 localización

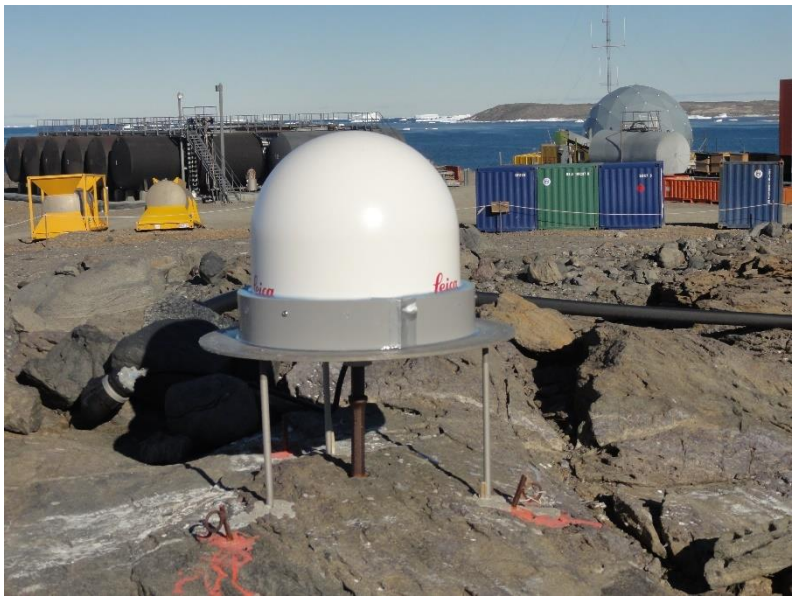
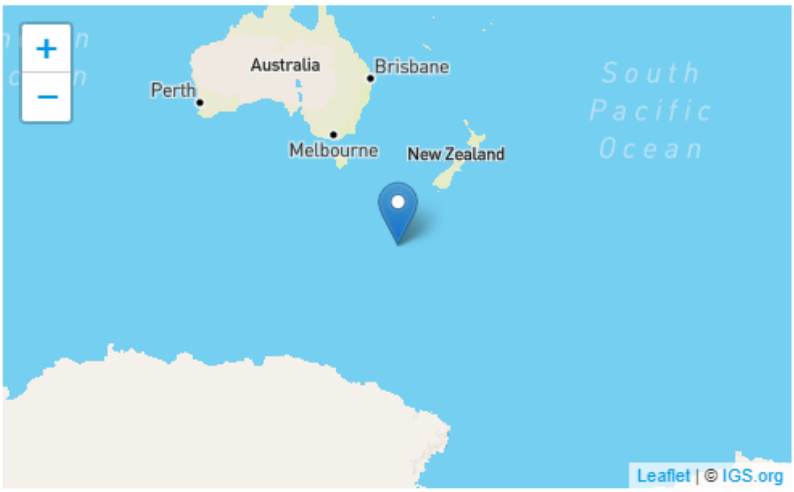


Figura 13: Foto de la estación DAV1

- Estación MAC1.



Country/Region	Macquarie Island, Australia
Latitude, Longitude	-54.499, 158.936
Elevation	-6.7 m

Figura 14: Estación MAC1 localización

Figura 15: Foto de la estación MAC1

- Estación MAW1.

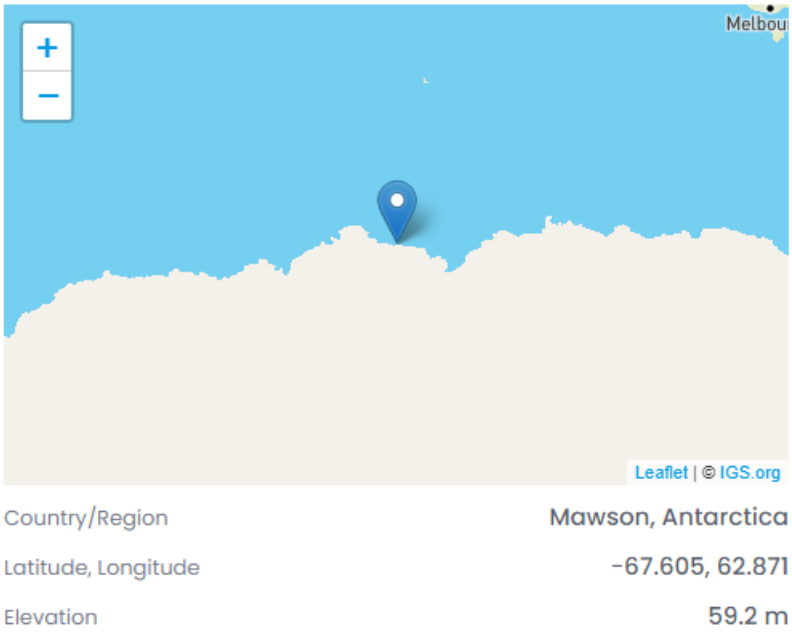


Figura 16: Estación MAW1 localización

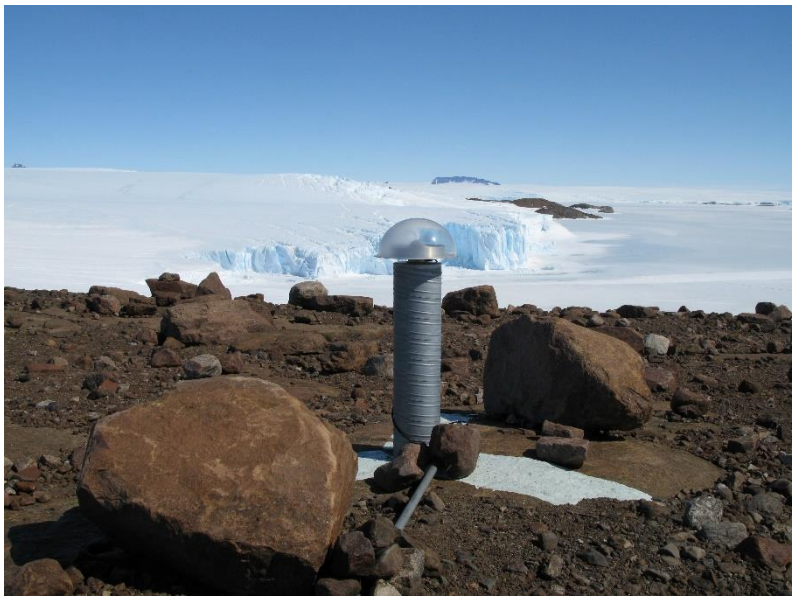
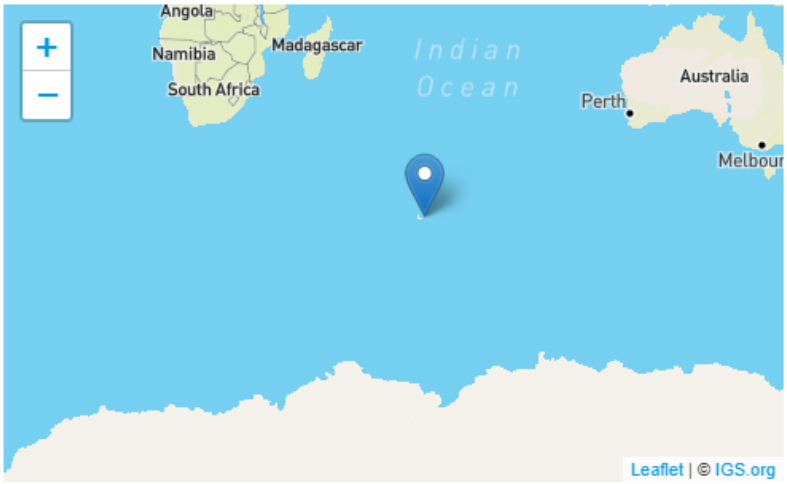


Figura 17: Foto de la estación MAW1

- Estación KERG.



Country/Region	Port aux Francais, French Southern Territories (the)
Latitude, Longitude	-49.351, 70.256
Elevation	73.0 m

Figura 18: Estación KERG localización

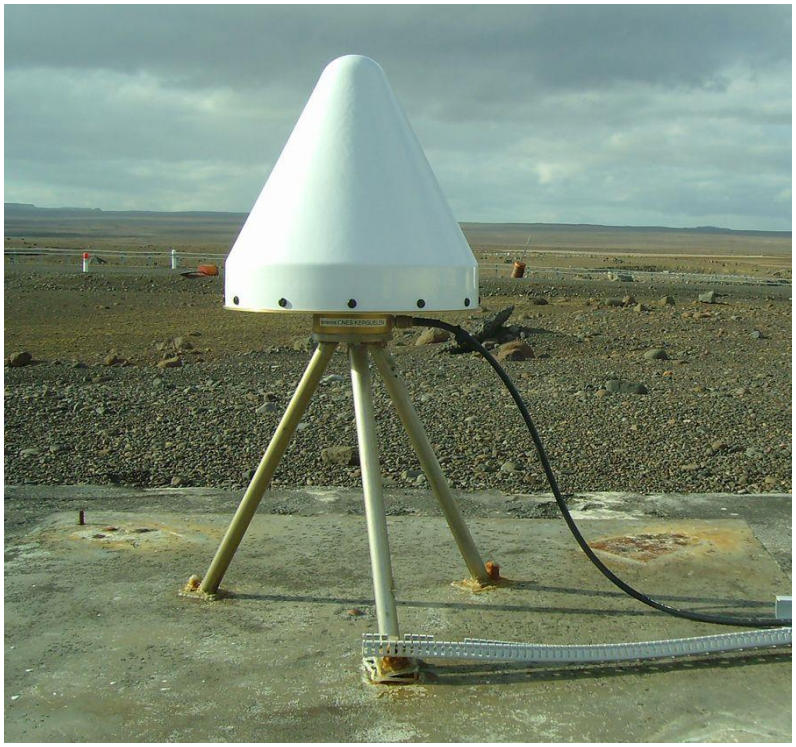
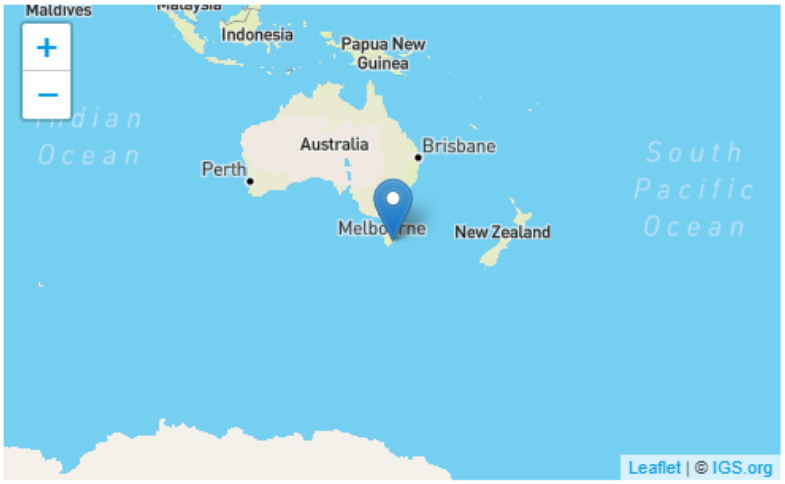


Figura 19: Foto de la estación KERG

- Estación HOB2.



Country/Region	Hobart, Australia
Latitude, Longitude	-42.805, 147.439
Elevation	41.0 m

Figura 20: Estación HOB2 localización

Figura 21: Foto de la estación HOB2

- Estación PERT.

Country/Region	Perth, Australia
Latitude, Longitude	-31.802, 115.885
Elevation	12.7 m

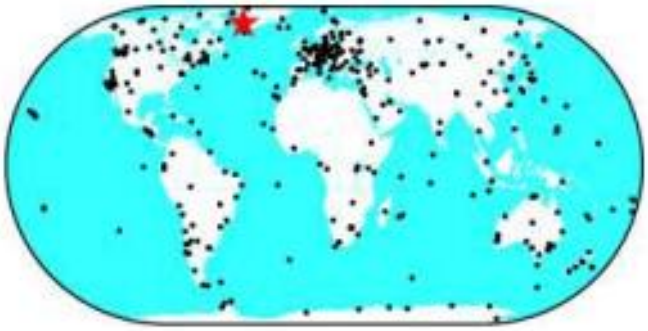
Figura 22: Estación PERT localización

Figura 23: Foto de la estación PERT

- Estación KELY.

RESEÑAS ESTACIONES IGS

ID	**KELY**	*Latitud*	+ 66º59'14",7067
País	Greenland	*Longitud*	- 50º56'41".4199
Localidad	Kangerlussuaq	*Altura*	229,800
Placa tectónica	North American	*EUREF*	Sí
Domo IERS	43005M002	*Datos*	11 años

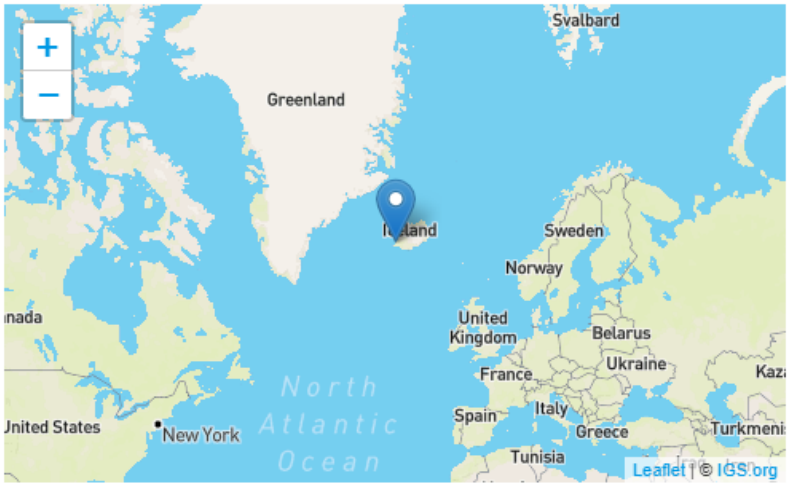


Organismo	JPL (Jet Propulsion Laboratory)
Contacto	dstowers@jpl.nasa.gov

Imágenes IGS

Figura 24: Reseña KELY

- Estación REYK.

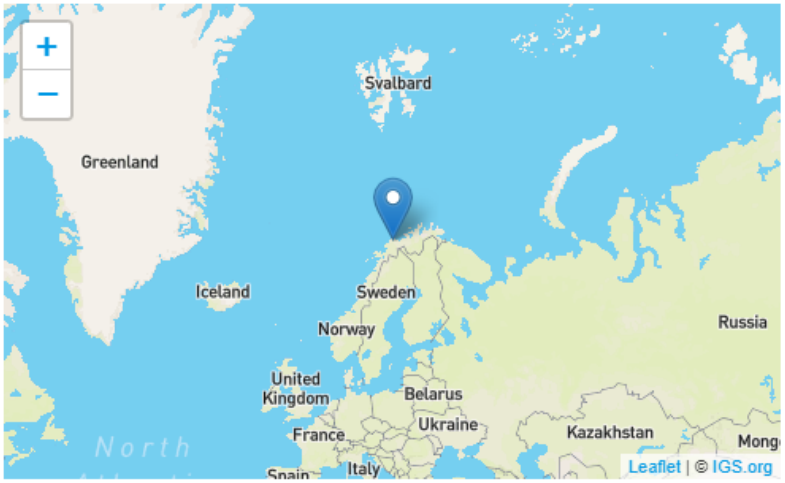


Country/Region	Reykjavik, Iceland
Latitude, Longitude	64.139, -21.955
Elevation	93.1 m

Figura 25: Estación REYK localización

Figura 26: Foto de la estación REYK

- Estación TRO1.



Country/Region	Tromsoe, Norway
Latitude, Longitude	69.663, 18.940
Elevation	138.0 m

Figura 27: Estación TRO1 localización

Figura 28: Foto de la estación TRO1

- Estación SCH2.

Country/Region	Schefferville, Canada
Latitude, Longitude	54.832, -66.833
Elevation	498.20 m

Figura 29: Estación SCH2 localización

Figura 30: Foto de la estación SCH2

- Estación NYAL.

Country/Region	Ny-Alesund, Norway
Latitude, Longitude	78.930, 11.865
Elevation	79.0 m

Figura 31: Estación NYAL localización

Figura 32: Foto de la estación NYAL

.

4. Metodología

De modo esquemático podemos resumirla con el siguiente mapa conceptual:

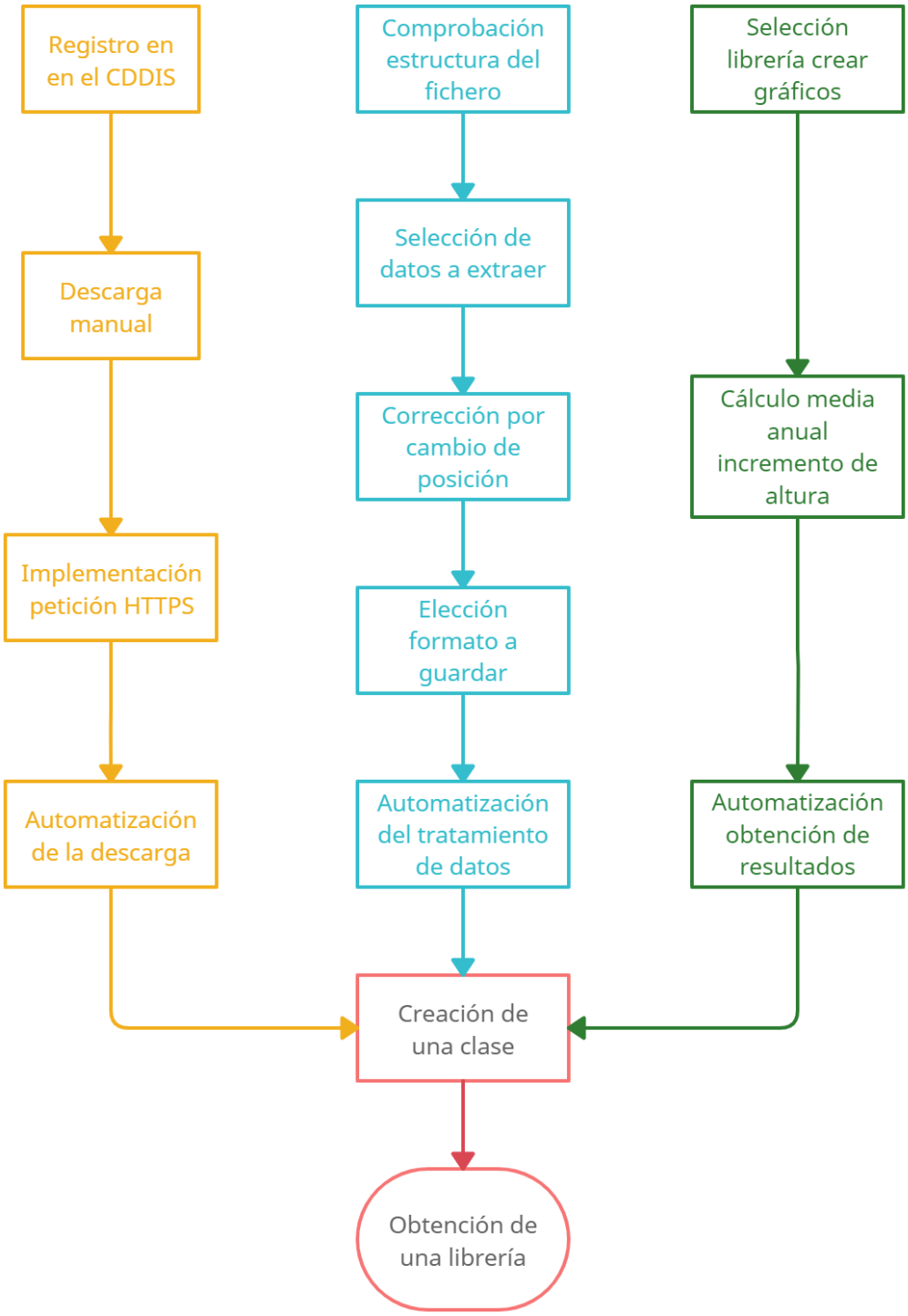


Figura 33: Mapa conceptual de la metodología

En la realización de este proyecto se han seguido una serie de pautas como se observa en el mapa anterior. Se ha dividido el trabajo en cinco procesos con sus distintas fases.

Además, cabe mencionar que la aplicación está hecha para todas estaciones permanentes encontradas en los ficheros repro2 para cada semana GPS. Como se verá posteriormente un apartado se centrará en las estaciones localizadas en la placa euroasiática ya que se tiene una mayor densidad de puntos que permitirá un mejor análisis.

4.1. Proceso 1: Descarga de datos

La descarga se realiza desde la página web de la nasa https://cddis.nasa.gov/archive/gnss/products/. El proceso se agiliza con el empleo del lenguaje de programación Python que selecciona los datos, gestiona la descarga y almacena en ficheros locales, evitando así la búsqueda y descarga manual. En el código, se emplea una petición HTTP que permite obtener cualquier fichero de un directorio concreto dentro de la web de la nasa.

https://cddis.nasa.gov/Data_and_Derived_Products/CDDIS_Archive_Access.html .

Para realizar la petición previamente se ha definido un listado con la selección de estaciones a estudio y para cada una de las semanas de la serie temporal que interesan, por ejemplo:

https://cddis.nasa.gov/archive/gnss/products/1899/repro3/

De las distintas soluciones disponibles en la web, se ha optado por la solución repro3 con cálculos en postproceso que tienen en cuenta la problemática asociada a las incidencias en el IGS, así como los cambios de antena, de receptor o cambios de sistema de referencia.

A su vez se ha seleccionado por medio de una comparación entre una cadena de caracteres los datos correspondientes a la solución proporcionada por el IGS en el día 7 de modo que tengamos la diferencia de datos sea de una semana.

Esta petición se ha modificado para poder realizar el proceso de descarga de forma automática. La estructura de esta petición se encuentra en el anexo 1.

4.2. Proceso 2: Análisis y tratamiento de datos

Dentro de este proceso se realiza el siguiente tratamiento con los datos:

- Comprobación estructura del fichero: El formato del fichero de la NASA es comprimido, por ello se ha buscado la librería “Gzip” para su descompresión. Del mismo modo para la estructura de datos “.Z” se ha localizado la librería “unlzw3” que permite el tratamiento en nuestra aplicación.
- Del fichero descomprimido seleccionamos la información necesaria. En la sección “SITE/IDE” obtenemos el código y descripción de la estación.

Tabla 1: datos "SITE/IDE

*CODE	PT	__DOMES__	T	_STATION DESCRIPTION__	_LONGITUDE_	_LATITUDE__	HEIGHT_
00NA	A	59975M001	P	Darwin Supreme Court B	130 50 38.4	-12 27 59.9	104.9
0ARK	A	M	P	0ARK	016 57 45.6	58 28 59.8	37.6
0BIS	A	M	P	0BIS	011 53 30.9	57 43 30.2	131.5
0BOD	A	M	P	0BOD	017 03 31.2	57 14 49.0	45.5
0FRL	A	M	P	0FRL	011 54 43.9	57 39 0.9	112.7
0HDG	A	M	P	0HDG	017 56 2.7	59 13 18.2	107.4
0HNA	A	M	P	0HNA	017 52 48.2	60 32 5.6	42.3

De la sección “SOLUTION/ESTIMATE” obtenemos coordenadas geocéntricas X,Y,Z, el periodo de descarga y su error correspondiente.

Tabla 2: datos "SOLUTION/ESTIMATE

*INDEX	_TYPE_	CODE	PT	SOLN	_REF_EPOCH__	UNIT	S	___ESTIMATED_VALUE___	__STD_DEV__
1	STAX	00NA	A	1	18:003:43200	m	2	-4.07366253168645e+06	3.59659e-03
2	STAY	00NA	A	1	18:003:43200	m	2	4.71206464365688e+06	3.99937e-03
3	STAZ	00NA	A	1	18:003:43200	m	2	-1.36787408002471e+06	1.61142e-03
4	STAX	0ARK	A	1	18:003:43200	m	2	3.19690745187798e+06	1.15362e-03
5	STAY	0ARK	A	1	18:003:43200	m	2	9.75114902294180e+05	6.51339e-04
6	STAZ	0ARK	A	1	18:003:43200	m	2	5.41410144161527e+06	1.80726e-03
7	STAX	0BIS	A	1	18:003:43200	m	2	3.34079127209417e+06	1.21488e-03
8	STAY	0BIS	A	1	18:003:43200	m	2	7.03523116633426e+05	6.21990e-04
9	STAZ	0BIS	A	1	18:003:43200	m	2	5.36955931805510e+06	1.82310e-03

Esta selección de datos se realiza mediante un filtrado por medio del encabezado de la sección correspondiente. Después, ya que el fichero se estructura como un fichero con columnas de ancho fijo, seleccionamos las columnas delimitadas por un ancho fijo. El código que permite dicha selección se encuentra en el anexo 1.

Tratamiento de los datos

Se almacenan en listas diferentes los datos de cada estación. Luego por medio de la función “ecef_to_geodetic” se convierten las coordenadas geocéntricas en geodésicas; también se convierte el error de las coordenadas geocéntricas, para así obtener la desviación estándar en latitud, longitud y en altura elipsoidal.

- La selección de datos se convierte al formato “GeoJSON” que permite la representación de los objetos almacenados en sistemas SIG.

4.3. Proceso 3: Obtención de resultados.

En este proceso se ha llevado a cabo la generación de gráficos a partir de los ficheros GeoJSON obtenidos en el paso anterior. A partir del id de la estación tratada se filtra cada uno de los ficheros seleccionando las coordenadas semanales de dicha estación.

El objeto es identificar posibles líneas de tendencia en la representación gráfica de las coordenadas de cada estación en un periodo de años representativo y su correlación con el cambio climático. Para ello se ha visto suficiente procesar los datos anuales obteniendo el valor promedio anual y prescindiendo de valores alejados del intervalo considerado (en algunos casos, debidos a un error de medición).

El empleo de valores promedio anuales permite eliminar las posibles oscilaciones en el año derivadas de los cambios estacionales.

Además, se ha seguido un proceso de corrección en los datos antes de presentar los resultados. Los saltos bruscos en la serie anual identificados tras un cambio de equipamiento en la estación (antena o receptor) se descuentan, a partir de ese punto, en el resto de lecturas de la serie permitiendo una representación gráfica sin estas discontinuidades; se implementa en el código desarrollando una función específica

que identifica automáticamente la fecha en la que se produce el cambio, calcula la diferencia en la altura de la antena respecto de la fecha anterior y minora con este diferencial las lecturas posteriores en esa estación.

Para una mejor visualización se ha empleado un tipo de gráfico de puntos con barra de error que permite mostrar junto a la medición el valor de su desviación estándar.

4.4. Proceso 4: Creación de una clase

Una vez implementados los distintos métodos para la descarga, análisis, tratamiento de los datos y obtención de resultados se tienen que agrupar de alguna forma ya que el objetivo de este trabajo, además del análisis del incremento de alturas, pretende crear una herramienta que facilite a cualquier usuario sin conocimientos avanzados de Python el desarrollo de estudios de esta índole.

Por esta razón se ha definido una clase que permitirá llamar a cualquier método que tenga después de inicializarla, bastando simplemente para su uso el introducir los datos que necesite el objeto, como algunos datos que necesiten sus métodos.

Para la creación de la clase después de agrupar las funciones se tiene que crear unos objetos que almacenen los datos que son necesarios para el usuario o para algún método dentro de dicha clase. Estos objetos se encuentran en el anexo 1. Después para que los métodos funcionen dentro de la clase se realizan modificaciones para emplear los objetos proporcionados por la clase, en lugar de los datos introducidos por el usuario.

Una vez que ya funciona la clase se tiene que crear un directorio para almacenar los datos. Ese espacio debe cumplir una doble condición: ser un lugar compatible tanto para sistemas basados en Windows como en Linux o Apple, y poder definirse como una ruta de entorno predeterminada para el sistema.

En nuestro caso, para sistemas Windows se ha seleccionado la ruta:

“C:\Users\your_user\AppData\Roaming\.cddis\PROJECTS”

, en la que “your_user” es el usuario en el que se esté trabajando y la carpeta PROYECTS aque en que se depositarán los proyectos creados por el usuario que se componen de ficheros comprimidos, archivos “GeoJSON” y ficheros “SINEX”.

La clase “singleton” permite guardar la configuración de la variable de entorno a la que apunta el programa “AppData\Roaming”, de modo que, a través de esa variable y evaluando si la carpeta “.cddis” no existe, se pueda crear la primera vez dicha carpeta y sus subcarpetas. Esa clase será llamada por la clase con la que se estaba trabajando anteriormente y proporcionará el nombre del proyecto en el que se está trabajando actualmente. El código de estas dos clases se encuentra en el anexo 1.

4.5. Proceso 5: Formación de una librería

Por medio de las clases que se han nombrado anteriormente se puede crear una estructura similar a la que se ve en las librerías que nos proporciona Python. Es decir, en el punto actual del desarrollo del proyecto se obtiene una librería llamada “cddis” guardada en un fichero “.py”. El usuario que desea utilizarla solo tiene que importarla como haría con cualquier librería y ejecutar las diferentes funciones que nos proporciona.

Por ejemplo, si se desea realizar una descarga de datos desde el año 1995 hasta el año 2020 y guardarlo en una carpeta llamada PROYECT_1995_2020 tan solo haría falta introducir las siguientes líneas de código:

Tabla 3: Tabla de funciones

Función	Descripción
data = Proyect("PROYECT_1995_2020", "1995-1-1", "2020-1-1")	Sirve para llamar al constructor de la clase PROYECT, nombrar la carpeta de destino y seleccionar el periodo de descarga.
data.download_cddis()	Esta función realiza la descarga de datos y descomprime los ficheros para su posterior procesamiento.
data.generate_file():	Esta función sirve para mostrar el gráfico generado.
graph = data.graphe("RESO")	Utilizando la información tratada genera un gráfico para observar la variación de la altura de una estación en particular.
graph.show()	Esta función sirve para mostrar el gráfico generado.

5. Resultados

5.1. Consideraciones previas a tener en cuenta

Antes de examinar resultados se ha de tener en cuenta el problema en las discontinuidades en la medición de las estaciones permanentes debidas a cambios en el receptor o antena del satélite que han alterado la posición de la antena al sustituir dichas partes de la estación. Para realizar una correcta interpretación de los resultados se han considerado estos cambios como se ha mencionado previamente en el apartado de metodología, identificando las modificaciones en los equipos, dentro de las series de datos, y aplicando una corrección mediante la suma de la diferencia de alturas.

5.2. Resultados generados

A partir de los datos generados con el programa se crean los siguientes gráficos que representan los incrementos medios anuales evitando así las variaciones estacionales anuales:

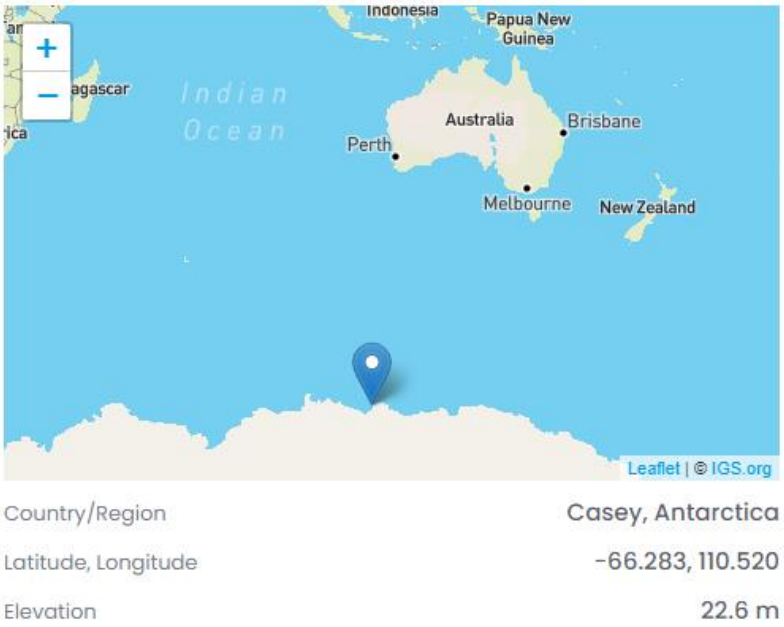


Figura 34: Estación CAS1 localización

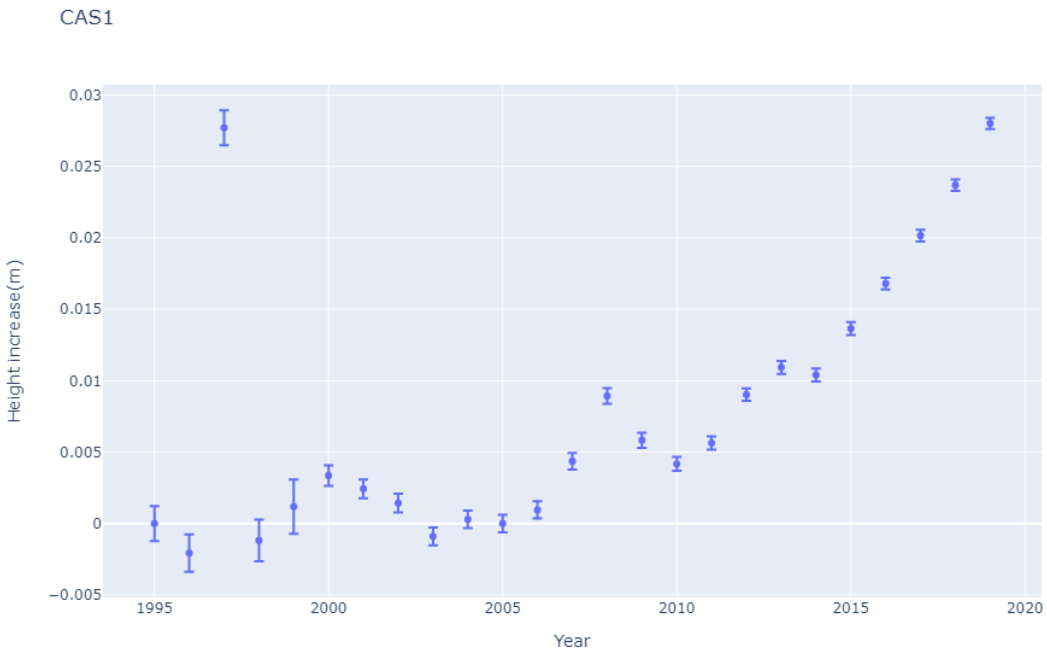


Figura 35: Gráfico con barra de error estación CAS1

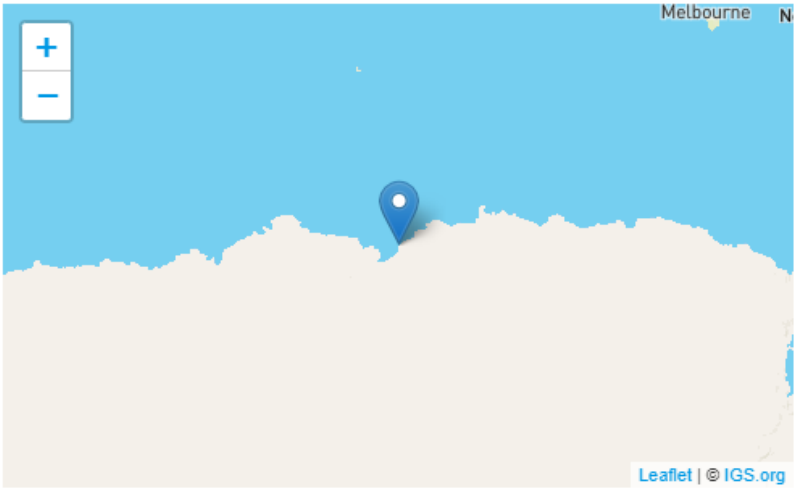


Figura 36: Estación DAV1 localización

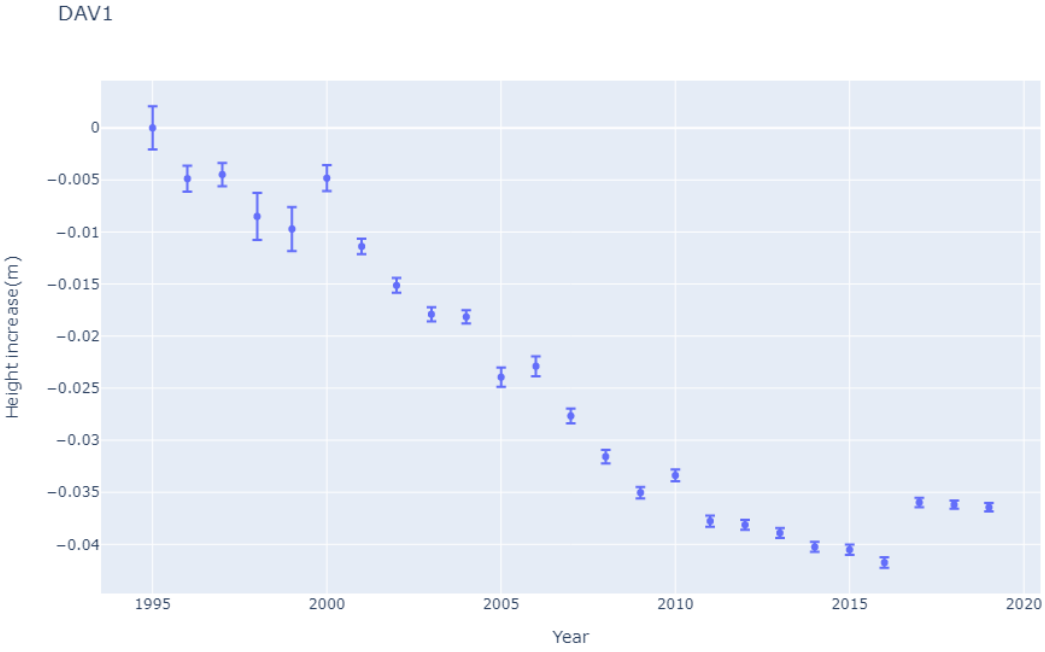
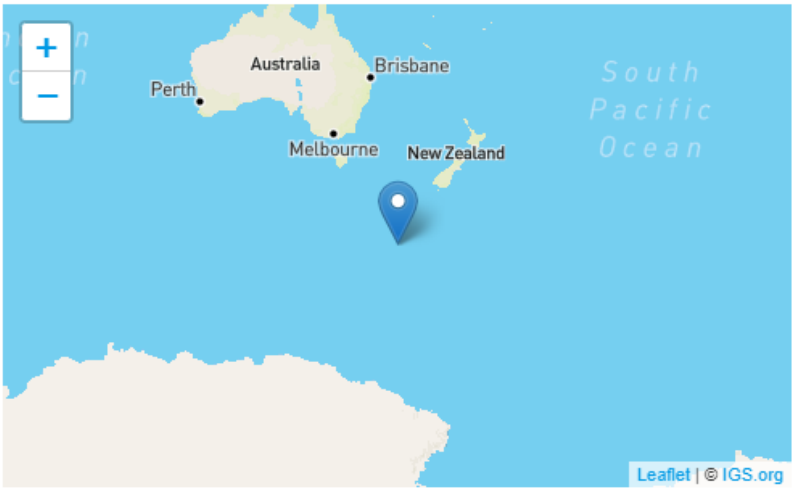


Figura 37: Gráfico con barra de error estación DAV1



Country/Region	Macquarie Island, Australia
Latitude, Longitude	-54.499, 158.936
Elevation	-6.7 m

Figura 38: Estación MAC1 localización

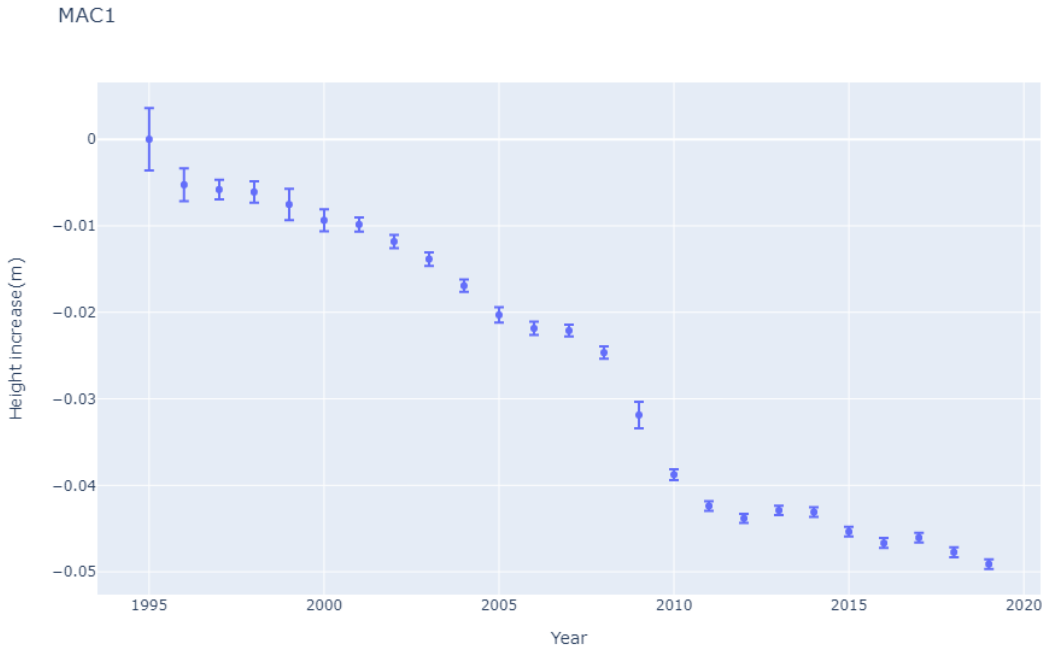


Figura 39:: Gráfico con barra de error estación MAC1

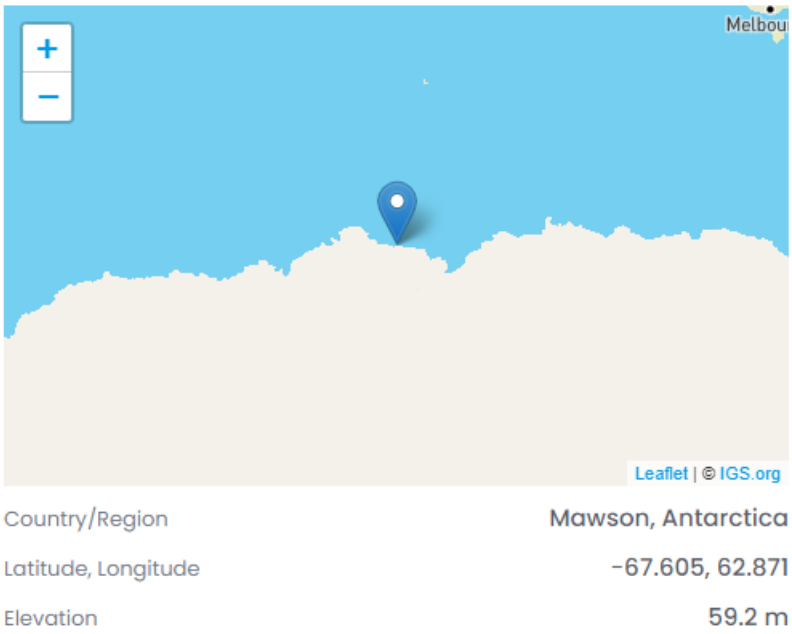


Figura 40: Estación MAW1 localización

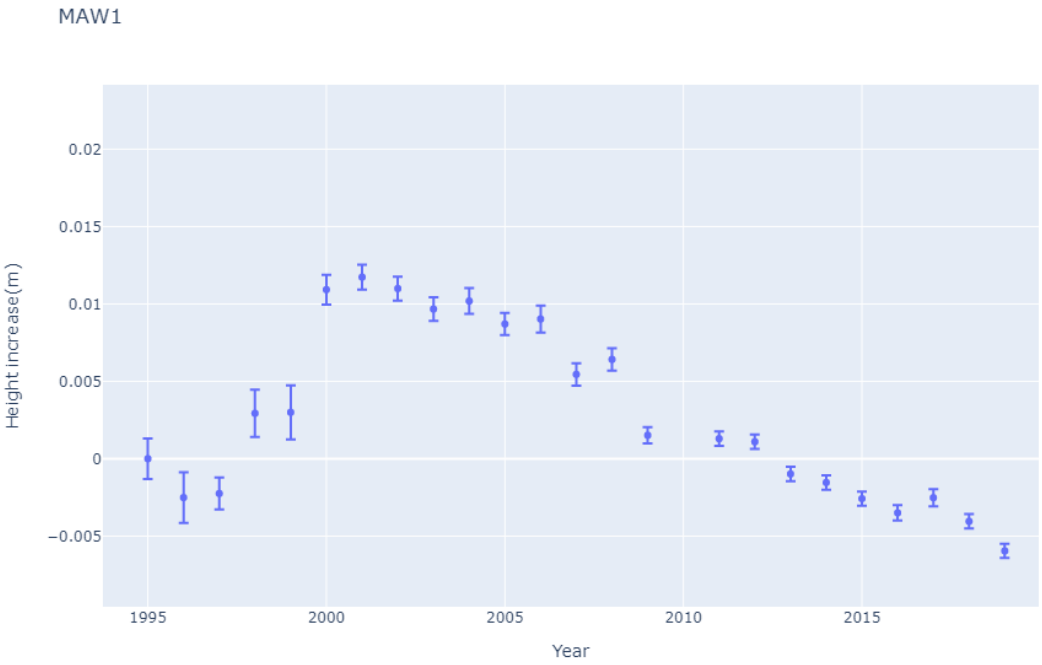
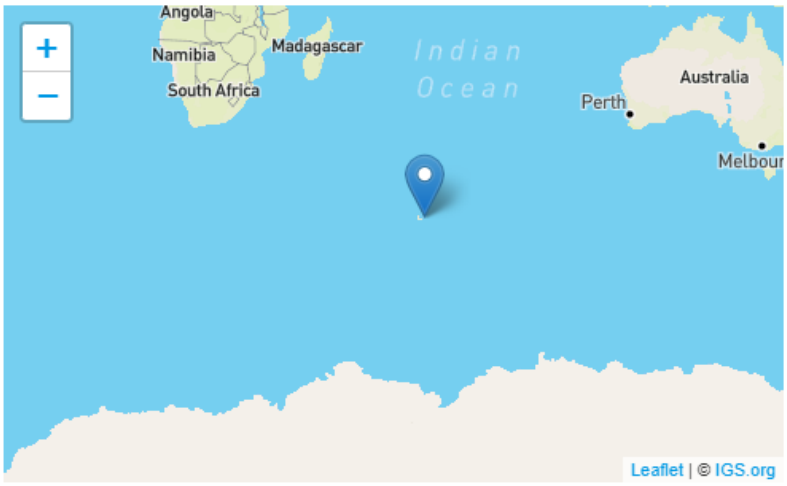


Figura 41: Gráfico con barra de error estación MAW1



Country/Region	Port aux Francais, French Southern Territories (the)
Latitude, Longitude	-49.351, 70.256
Elevation	73.0 m

Figura 42: Estación KERG localización

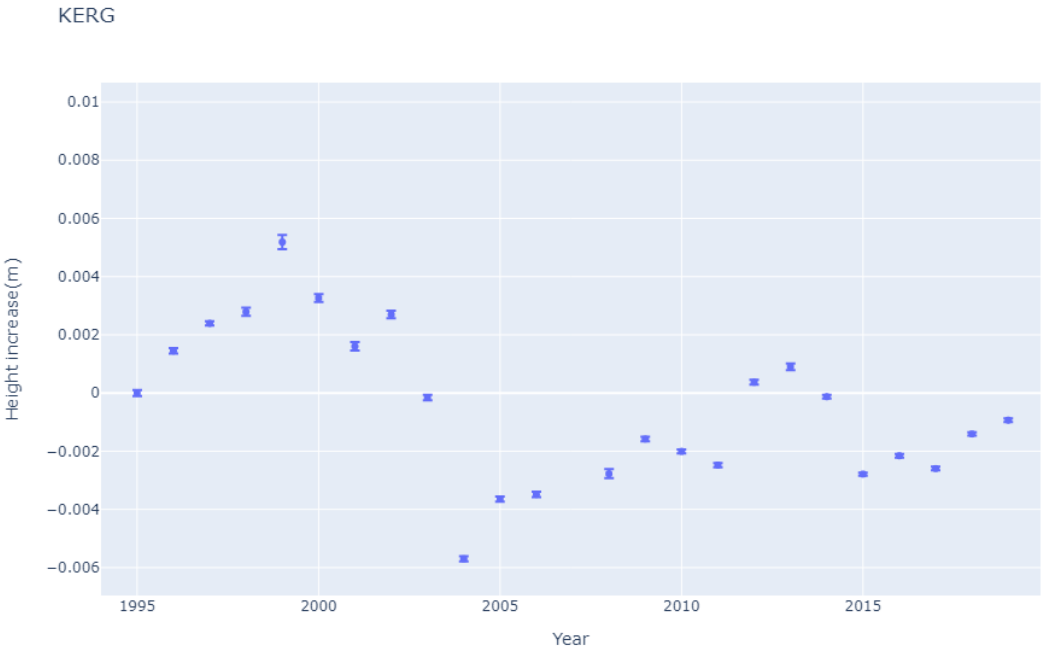
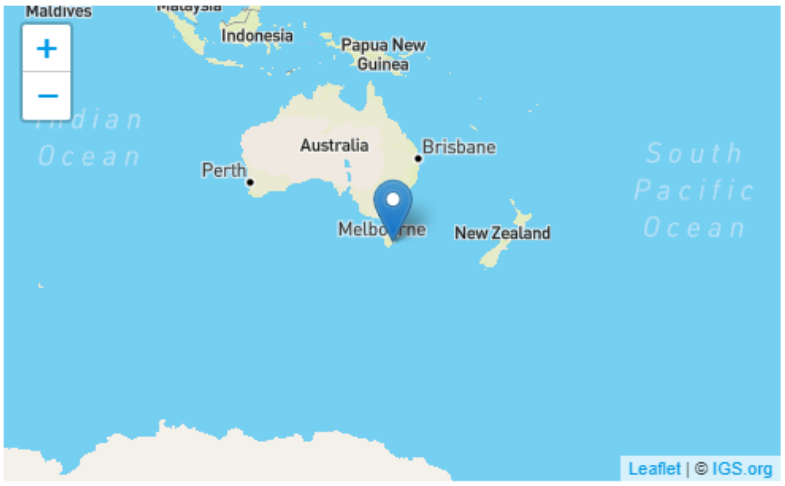


Figura 43: Gráfico con barra de error estación KERG



Country/Region	Hobart, Australia
Latitude, Longitude	-42.805, 147.439
Elevation	41.0 m

Figura 44: Estación HOB2 localización

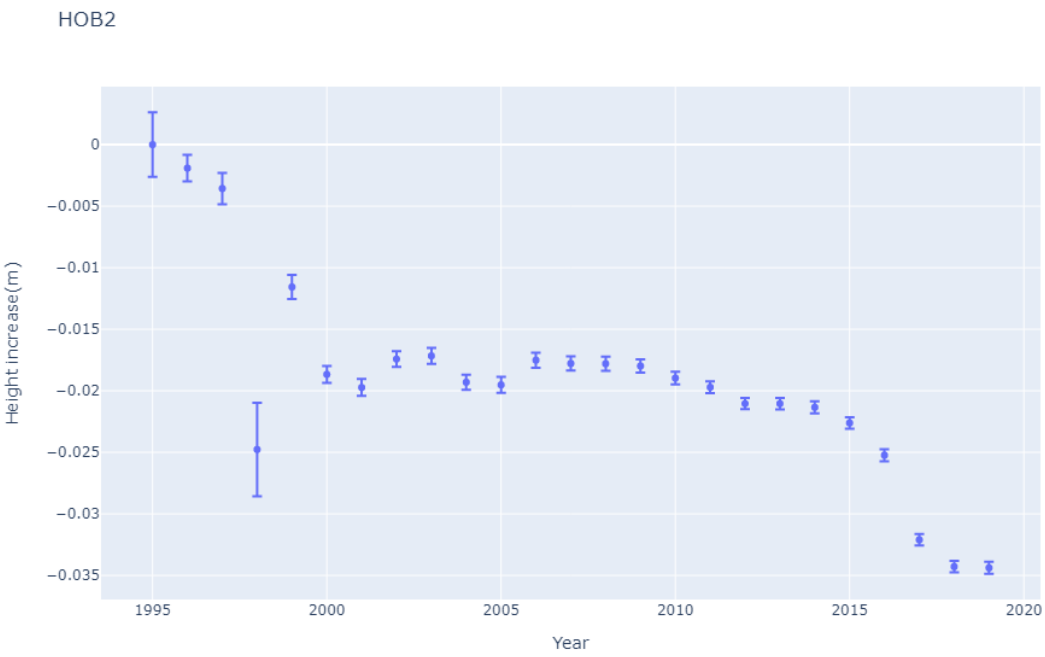


Figura 45: Gráfico con barra de error estación HOB2

Country/Region	Perth, Australia
Latitude, Longitude	-31.802, 115.885
Elevation	12.7 m

Figura 46: Estación PERT localización

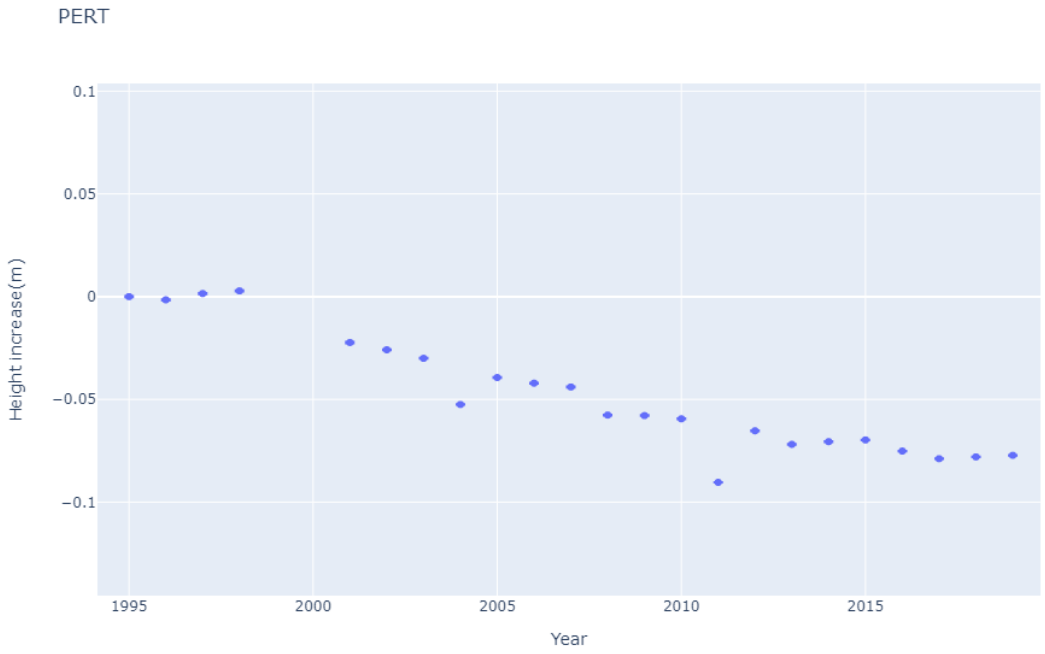
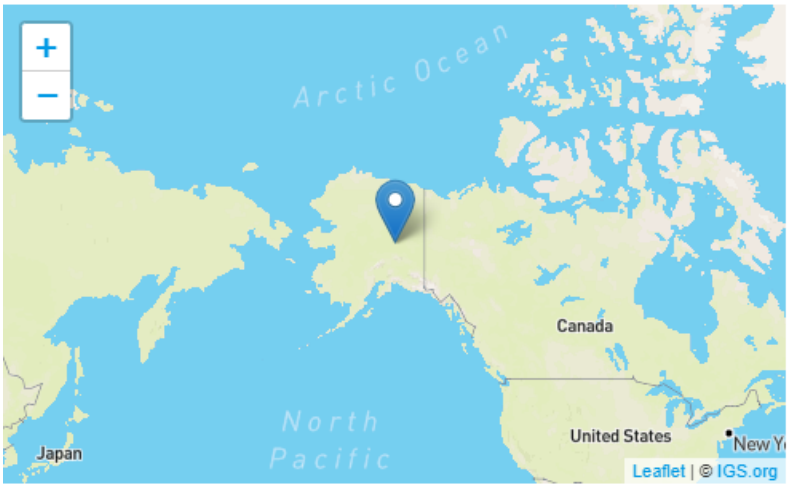


Figura 47: Gráfico con barra de error estación PERT



Country/Region	Fairbanks, United States
Latitude, Longitude	64.978, -147.499
Elevation	319.1771 m

Figura 48: Estación FAIR localización

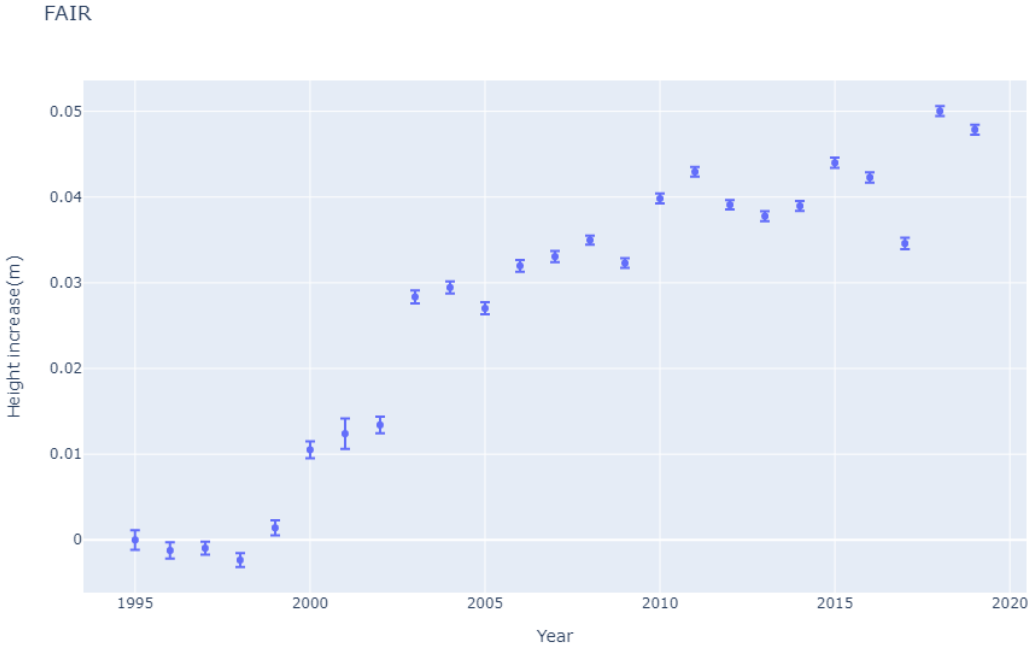


Figura 49: Gráfico con barra de error estación FAIR

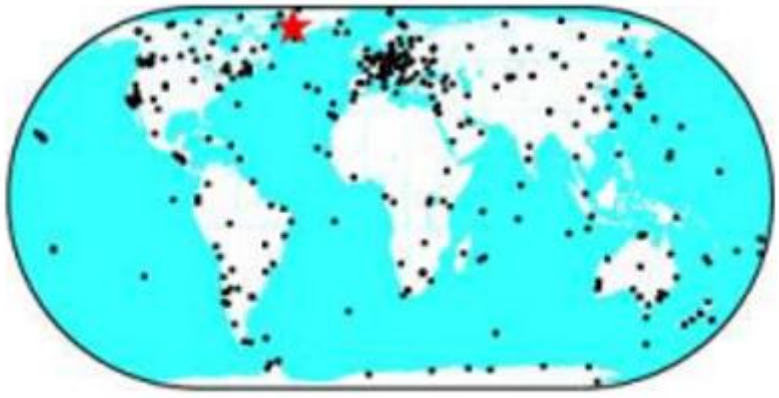


Figura 50: Estación KELY localización

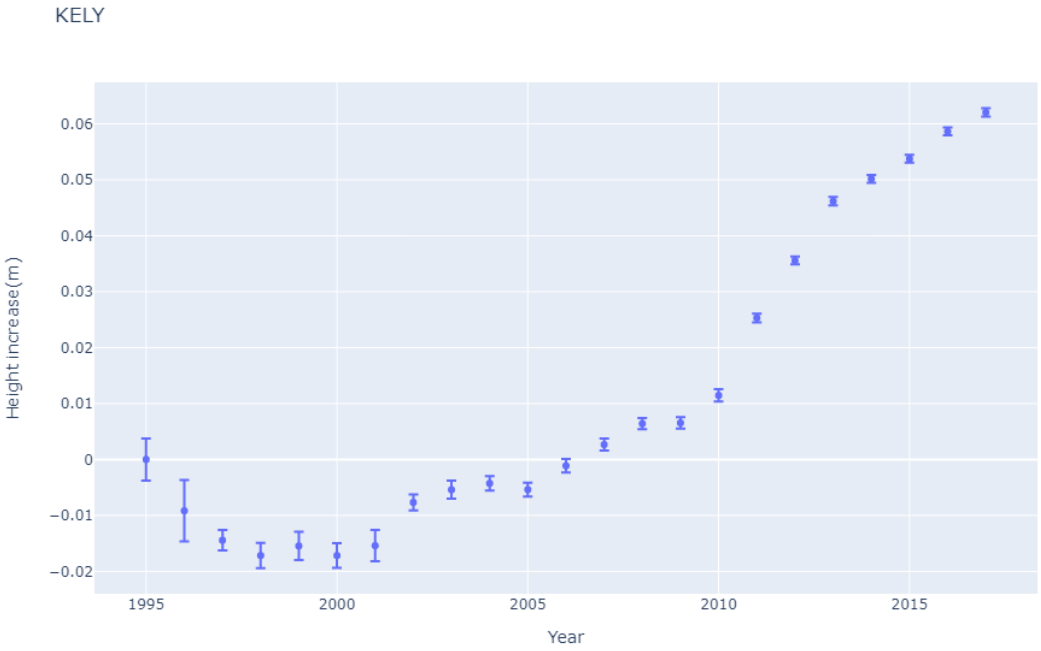
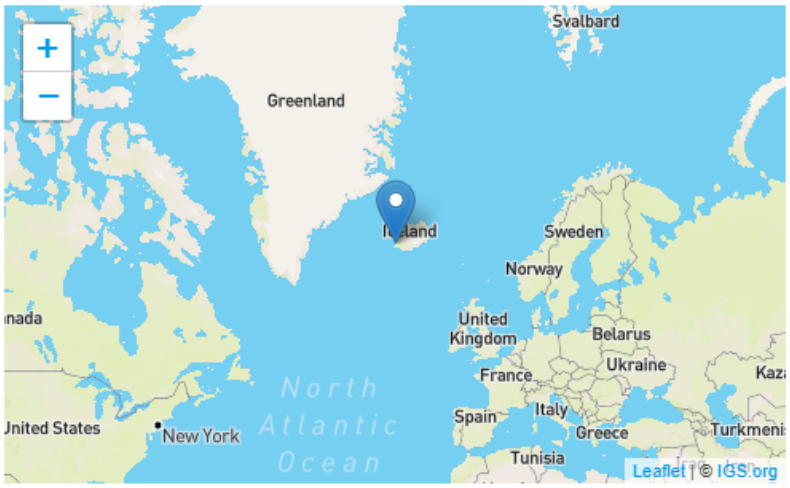


Figura 51: Gráfico con barra de error estación KELY



Country/Region	Reykjavik, Iceland
Latitude, Longitude	64.139, -21.955
Elevation	93.1 m

Figura 52: Estación REYK localización

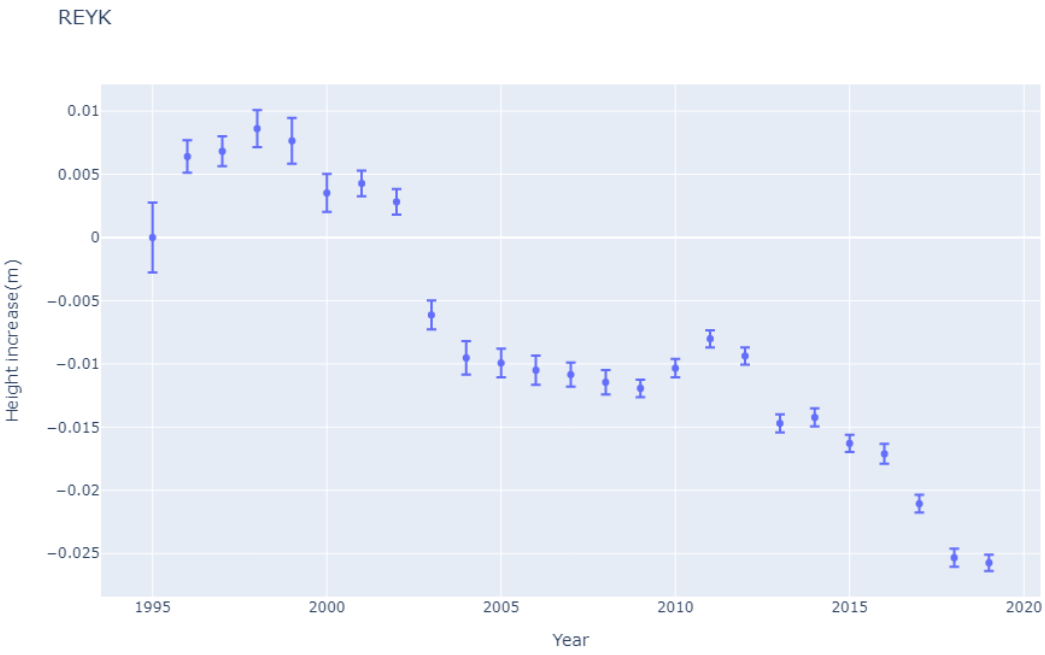


Figura 53: Gráfico con barra de error estación REYK

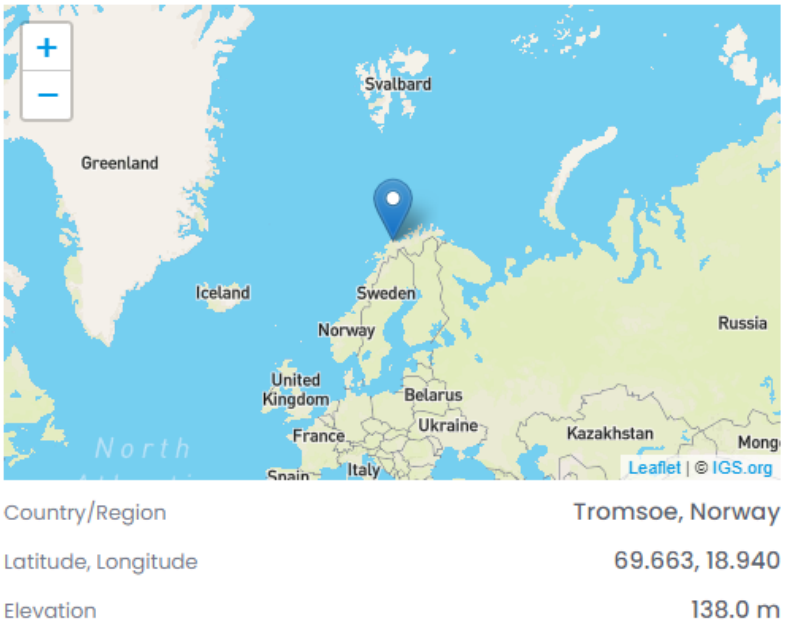


Figura 54: Estación TRO1 localización

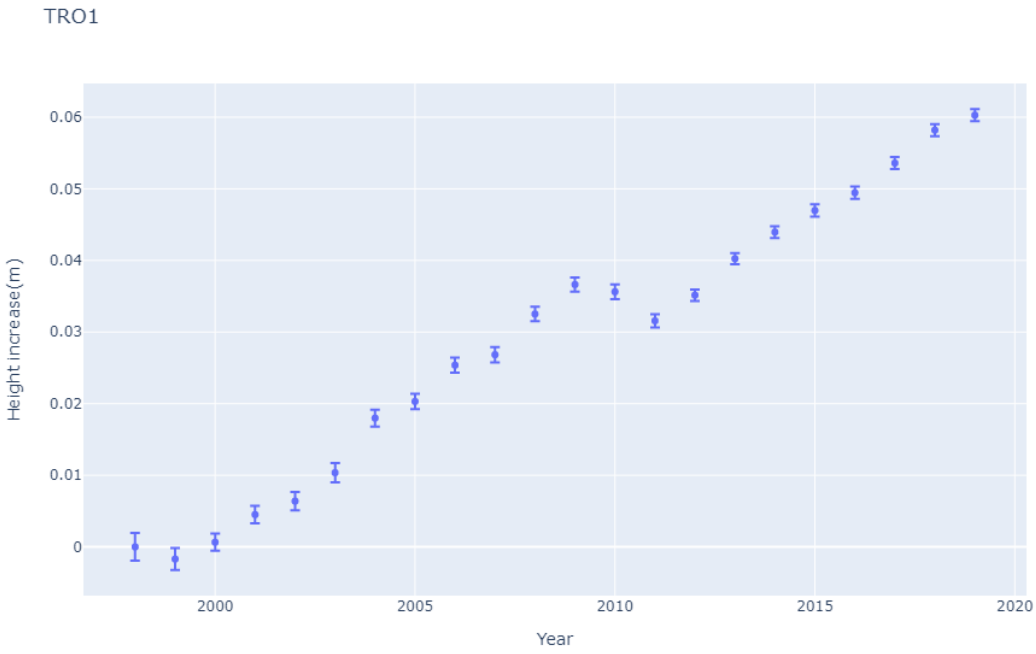


Figura 55: Gráfico con barra de error estación TRO1

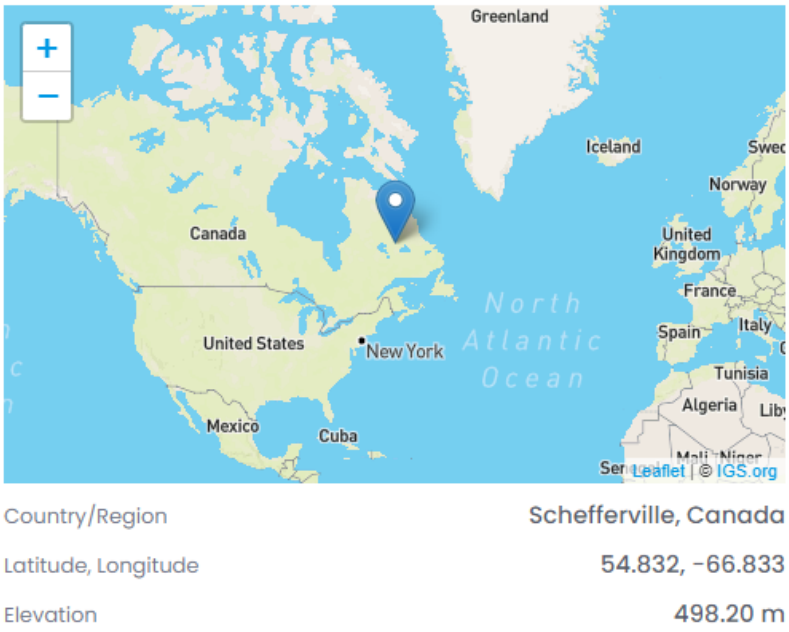


Figura 56: Estación SCH2 localización

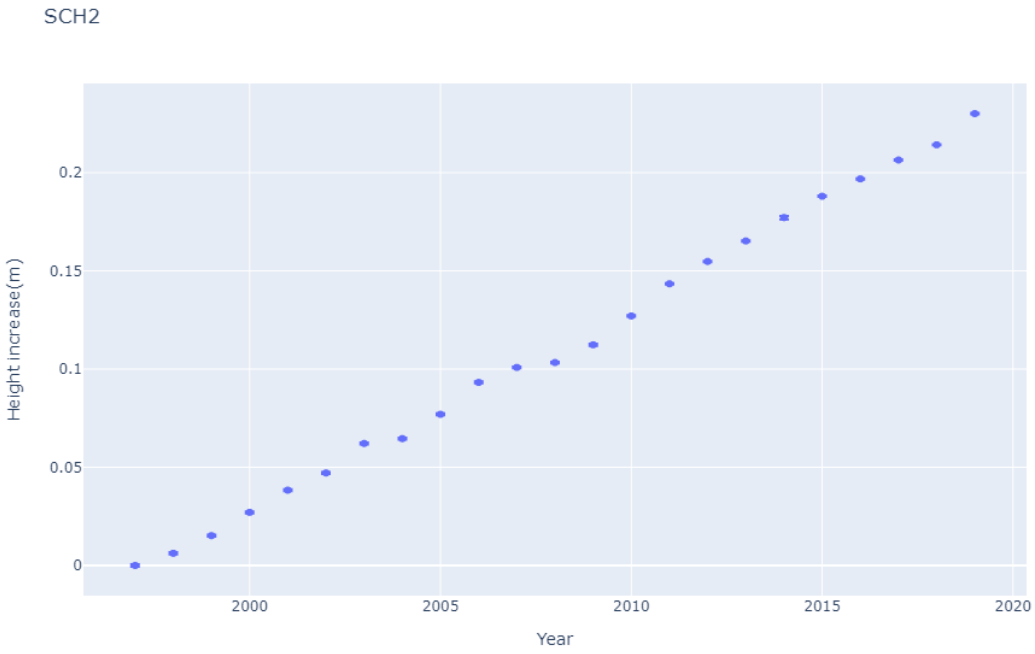


Figura 57: Gráfico con barra de error estación SCH2

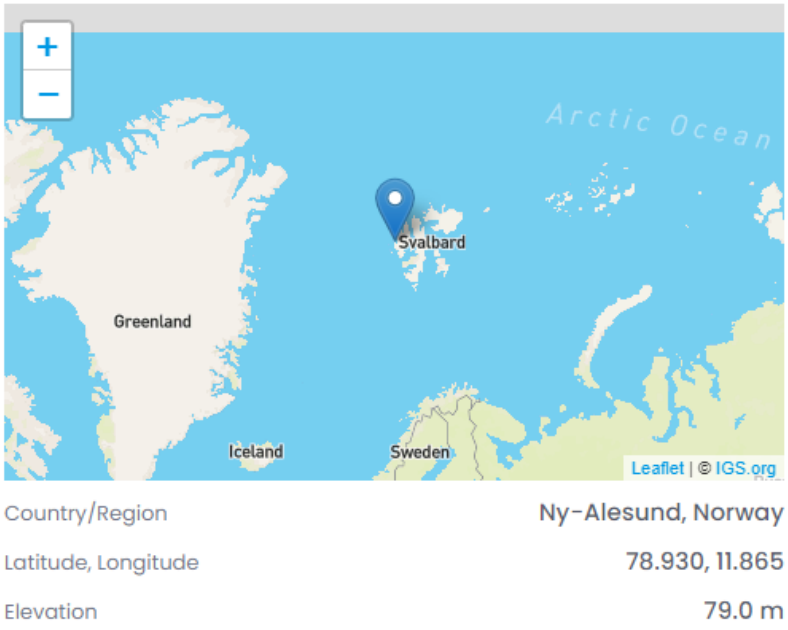


Figura 58: Estación NYAL localización

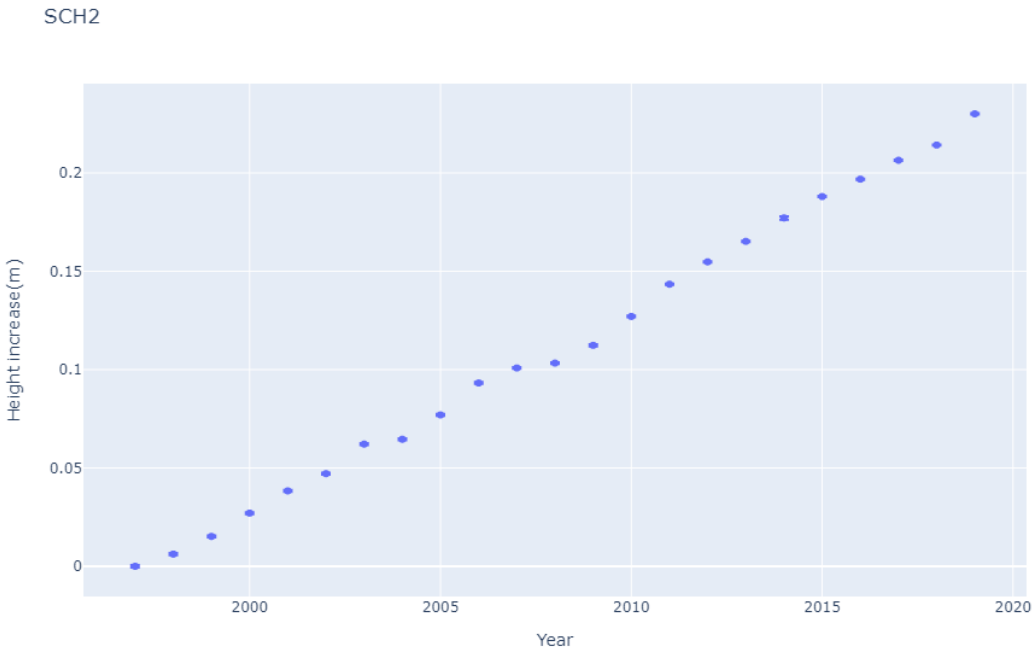


Figura 59: Gráfico con barra de error estación NYAL

Por otro lado, a partir de los archivos generados se ha realizado un intento para analizar la evolución secuencial de los incrementos de altura de la placa euroasiática a lo largo de los años 2000 a 2020. Para ello se ha generado un MDE (modelo digital de elevaciones) usando un método de interpolación por triángulos TIN para cada mes de ese periodo de tiempo. Algunas imágenes de muestra son las siguientes:

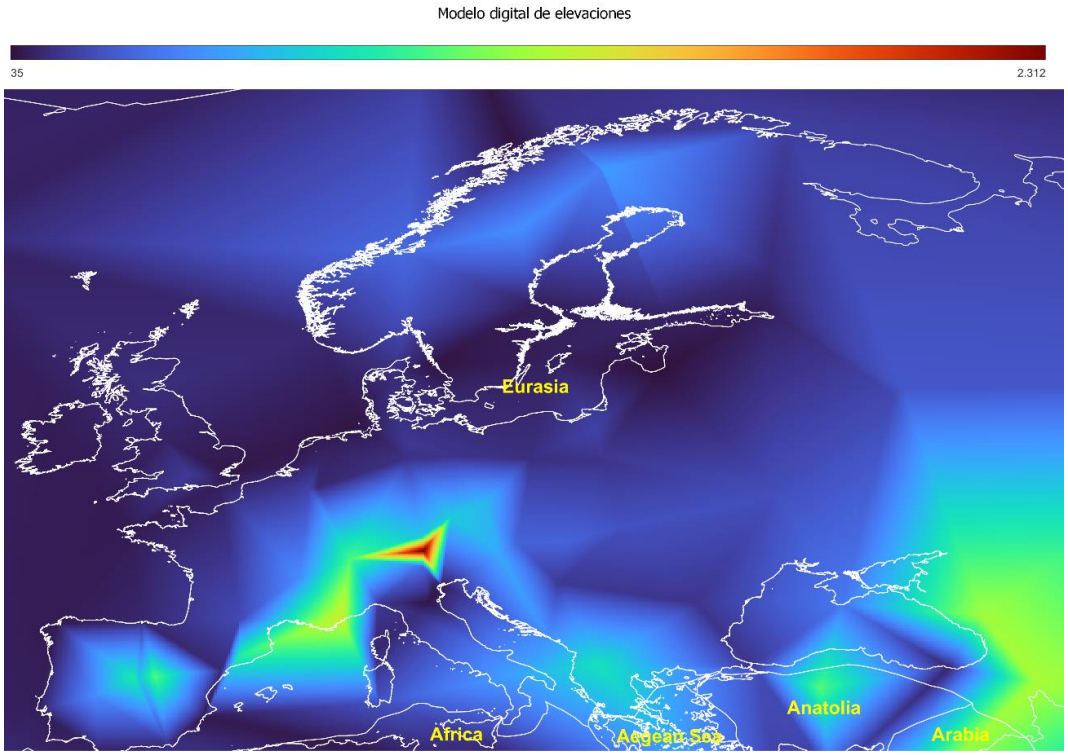


Figura 60: DEM de Europa año enero del 2001

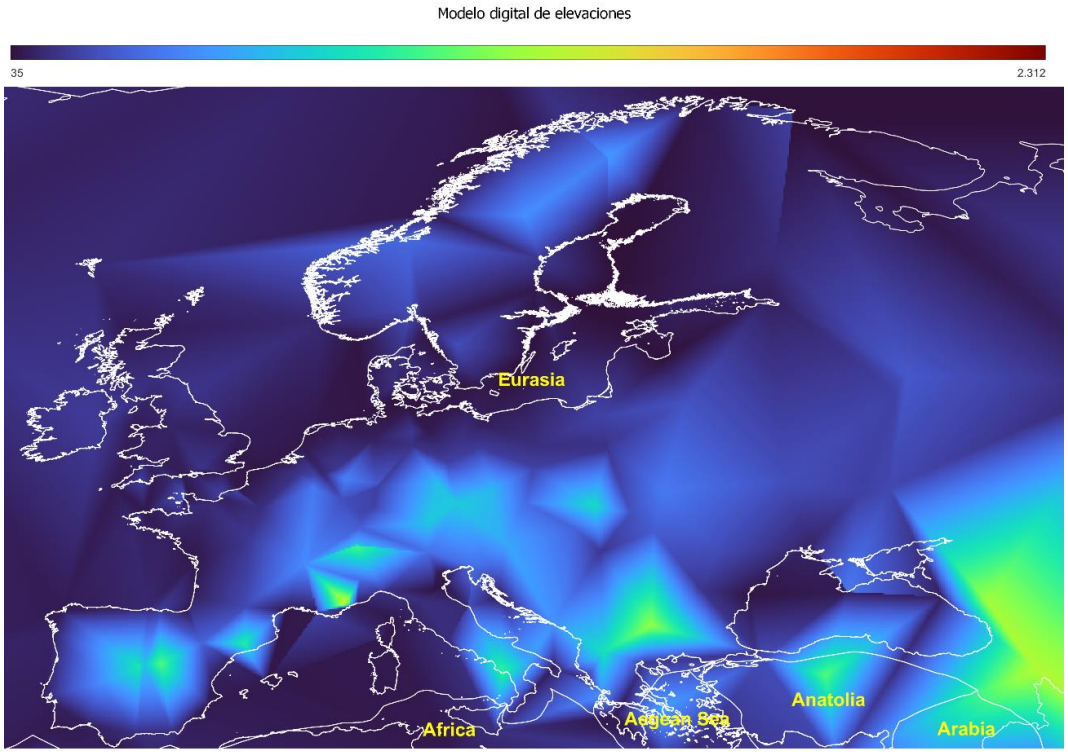


Figura 61: DEM de Europa año enero del 2015

Como se aprecia en las imágenes anteriores la densidad de puntos en los DEM generados es muy baja, esto combinado al utilizar el método TIN de interpolación dan como resultado figuras poco naturales. Además, la diferencia entre los diferentes meses es escasa y puede ser debido al efecto estacional. Con el fin de observar la evolución global se ha calculado la diferencia entre el DEM del año 2020 y el del año 2000 eliminando así el efecto estacional al estar tomadas dentro de la misma estación (enero del 2000 y diciembre de 2019). El resultado es el siguiente:

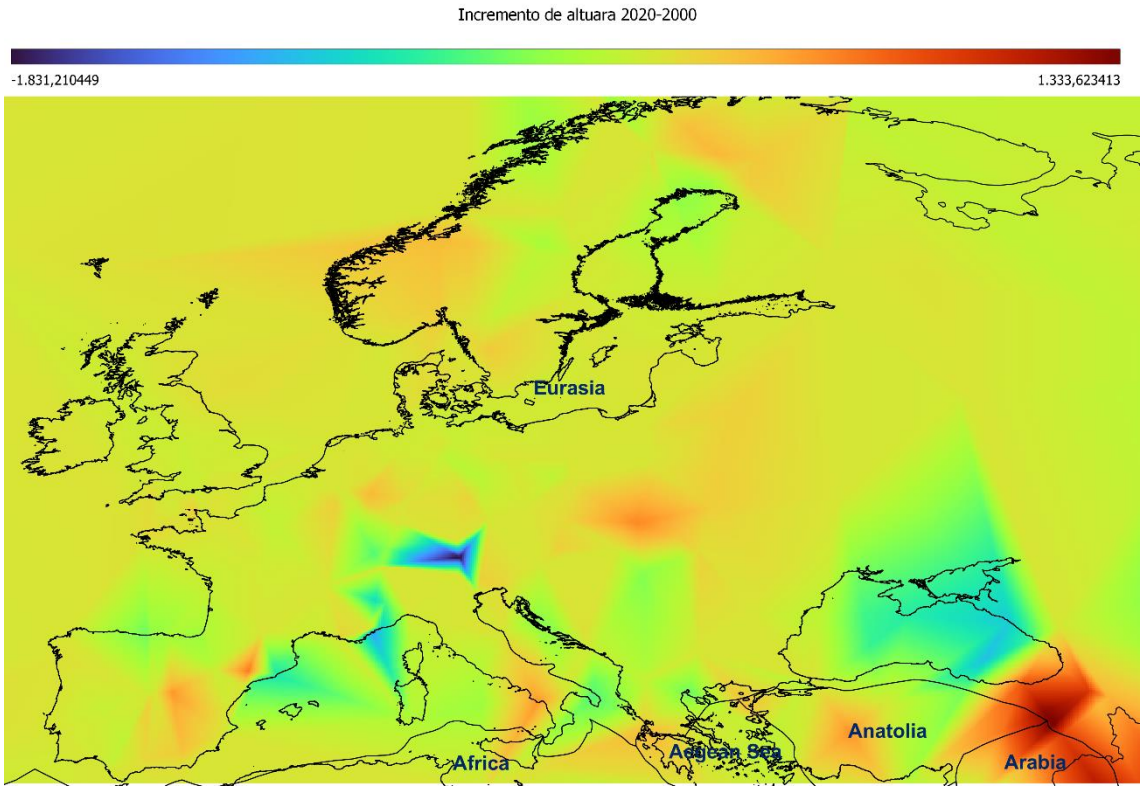


Figura 62: Diferencia DEM (2020-2000)

En el proceso de realizar el proyecto se ha creado una librería que permite a un usuario poco familiarizado con la página de la nasa obtener datos de estaciones permanentes y tratarlos de forma rápida y eficiente. El código de dicha librería se encuentra en el anexo 1.

Por último, se ha generado un mapamundi donde se muestra el número de estaciones que hay en el año 2020 y como están distribuidas. El mapa se encuentra en el anexo 1.

Analizando el resultado obtenido se concluye que se debería de investigar alternativas para la geovisualización de la variación en altura de la placa euroasiática. Estas alternativas se traducen en aplicar otros métodos de interpolación como el método “Spline”, para apreciar mejor la variación y calcular la diferencia entre los DEM creados.

Como futuras implementaciones además del video generado mencionado anteriormente se podría crear un geoportal para visualizar estos resultados dando libertad al usuario en la selección de estaciones, rango temporal de mediciones, método de procesamiento de los valores obtenidos y tipo de representación gráfica deseada.

6. Presupuesto

Se consideran los medios materiales y humanos para el desarrollo de la aplicación informática en un periodo de 7 meses (febrero-agosto) trabajando en el proyecto a tiempo parcial.

El servicio de programación ha dedicado personal equivalente de 212 días/hombre dentro del periodo.

Se consideran tanto los costes directos del proyecto como los indirectos que se denomina Presupuesto de Ejecución Material (PEM). Sumando el beneficio industrial (6%) se obtiene el Presupuesto de Ejecución por Contrata y aplicando a este último los impuestos según ley (IVA 21%), el Presupuesto Total del proyecto.

Código	Capítulo	Unidad	Descripción	Unidad Medida	Precio Unitario (€/Ud Medida)	Cantidad	Importe
1	Costes Directos	PROG_01	Programador Junior	Dias/hombre	86.16	212.00	18'265.92
		PROG_02	Equipo de programación incluyendo ordenador accesorios e impresora	Ud	852.00	1.00	852.00
		PROG_03	Material de oficina	Ud	104.00	1.00	104.00
2	Costes Indirectos	ALQ_01	Alquiler Local oficinas	Mes	450.00	7.00	3'150.00
		SERV_01	Servicios (Agua, Luz, telefonía, etc)	Mes	36.00	7.00	252.00
			PRESUPUESTO DE EJECUCION MATERIAL				**22'623.92**
			Beneficio Industrial (6%)				1'357.44
			PRESUPUESTO EJECUCION POR CONTRATA				**23'981.36**
			I.V.A. (21%)				5'036.08
			TOTAL				**29'017.44**

Tabla 4: Presupuesto

7. Conclusiones

Por una parte, observando los gráficos podemos concluir que varias estaciones cercanas a los polos presentan un incremento en la diferencia de altura respecto del primer dato del que se tiene fecha. Estos incrementos se atribuyen al deshielo de los glaciares cercanos el terreno circundante lo que se traduce en una disminución de la densidad de la corteza terrestre en la zona y la compensación en altura por el equilibrio isostático de la placa.

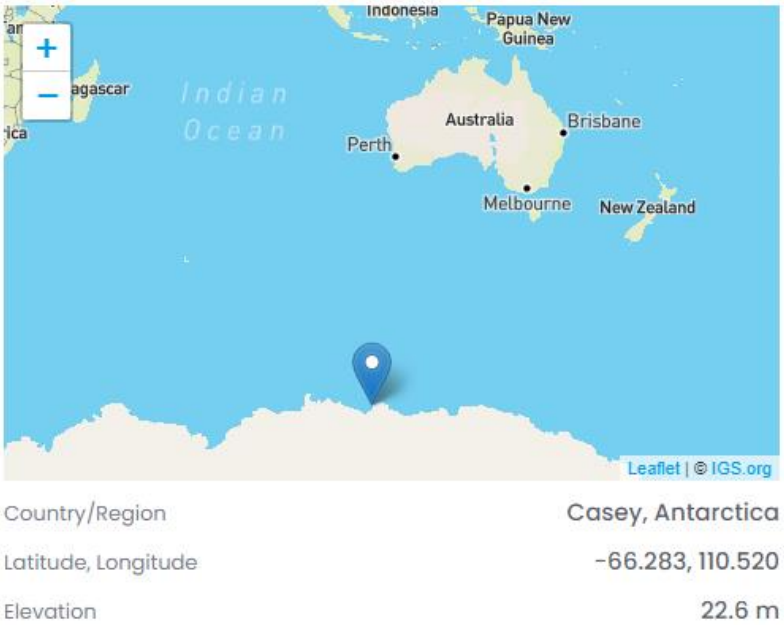


Figura 63: Estación CAS1 localización

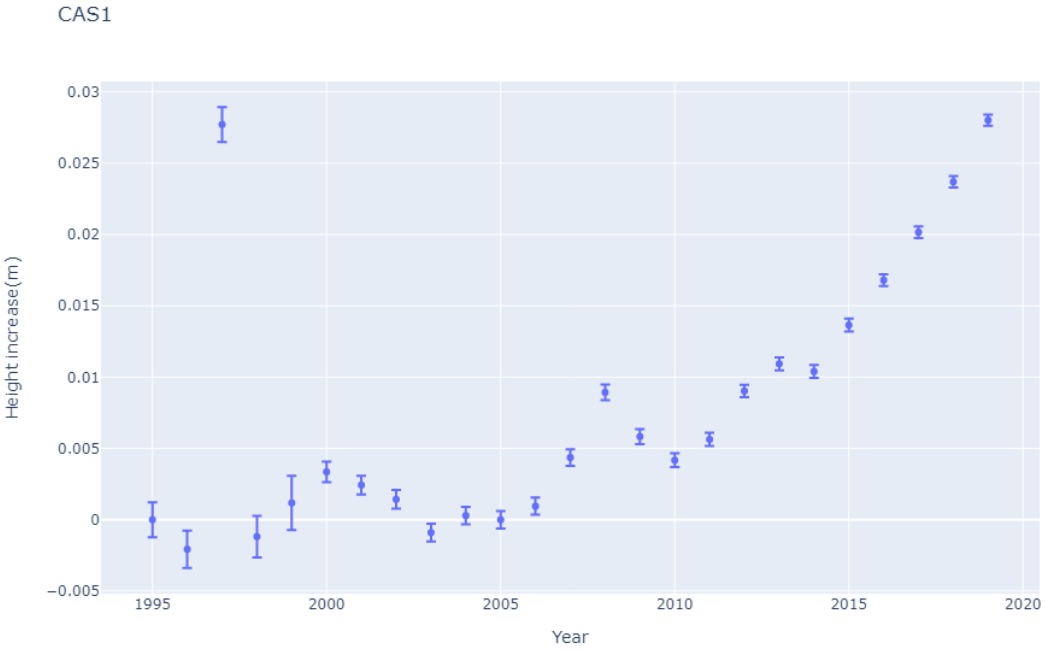


Figura 64: Gráfico con tendencia curva (CAS1)

Por ejemplo, como se puede observar en la imagen anterior de la estación CAS1 existe un importante incremento en la altura especialmente a partir del año 2014.

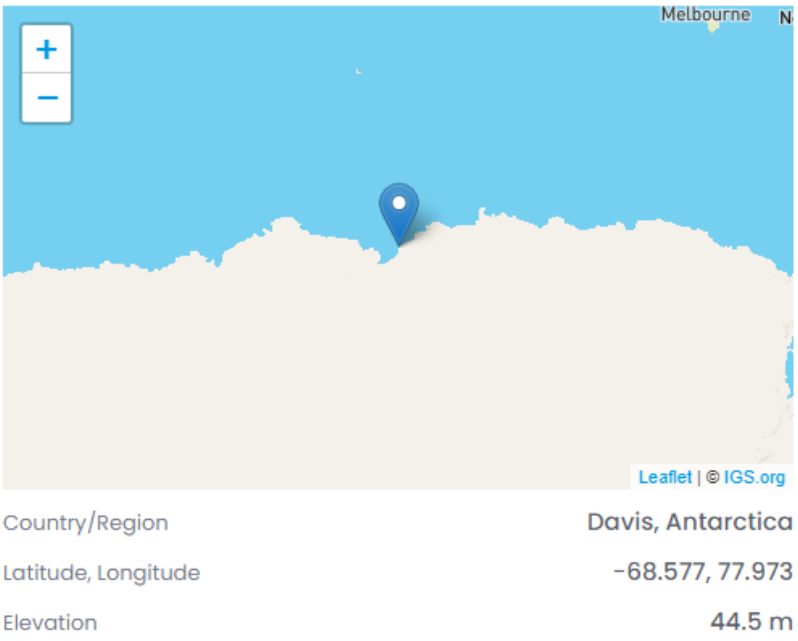


Figura 65: Estación DAV1 localización

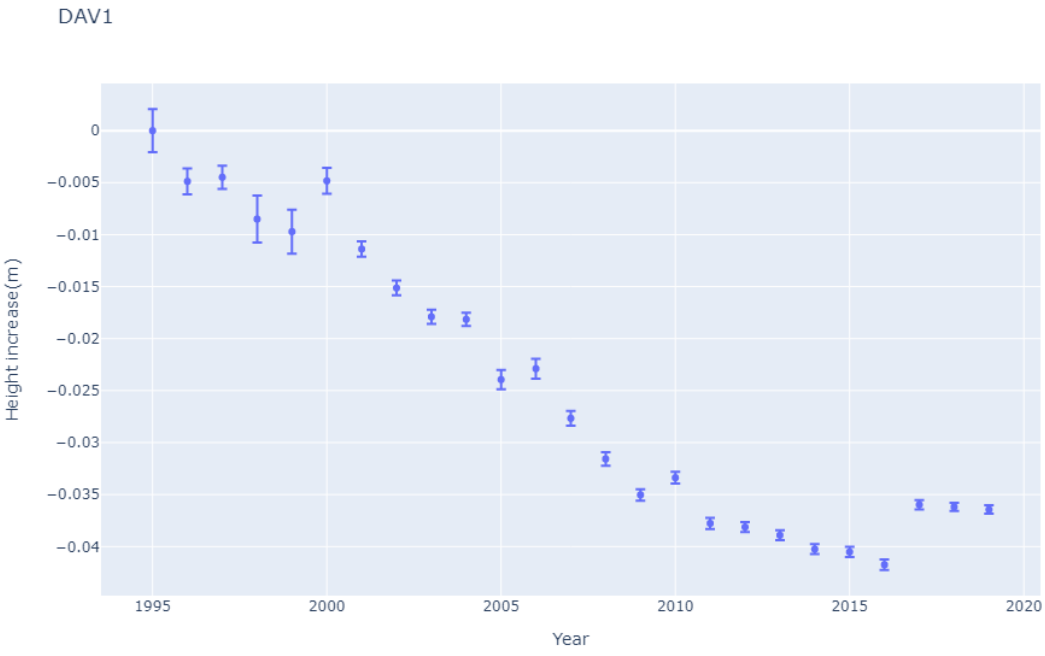


Figura 66: Estación que sigue el comportamiento de la placa tectónica (DAV1)

Por otro lado, aquí se observa una estación que pertenece a la misma placa tectónica que CAS1 (DAV1) que no sigue el mismo comportamiento. Esto refuerza más la conclusión de que ha habido un incremento en el deshielo en las zonas polares por factores humanos que no está relacionado con el comportamiento de la capa tectónica a la que pertenecen las estaciones. Debido a que en este trabajo se quiere abordar el tema analizando un amplio periodo de tiempo para llegar a mayores conclusiones sería necesario un estudio meteorológico de los años y así comprobar si la climatología de la zona tiene que ver con estas variaciones en altura y el comportamiento de esta a lo largo de las diversas estaciones del año.

Con lo que se observado anteriormente se puede llegar a la conclusión de que ha habido un incremento en el proceso de aceleración de la variación en altura a partir del año 2005. Esto coincide con las observaciones hechas en diversos artículos. Por ejemplo, en el artículo https://www.nasa.gov/goddard/2022/nasa-approves-continuation-of-icesat-2-after-3-years-of-big-results la NASA a través de un satélite lanzado en el año 2018 ICESat-2 ha observado que en los últimos años la masa de hielo que no se derretía durante el verano en el ártico ha disminuido considerablemente. También, en otro artículo de la NASA

https://www.nasa.gov/feature/esnt/2022/nasa-finds-2022-arctic-winter-sea-ice-10th-lowest-on-record se observa que debido al efecto del calentamiento global la zona del ártico se está calentado 4 veces más rápido que otras zonas del globo, provocando que el hielo que se conserva a lo largo de las estaciones del año sea menor.

Otra de las finalidades del trabajo era abordar si se ha desarrollado una aplicación que pueda ser utilizada como librería por otros usuarios facilitando la realización de sus propios estudios. Como se ha podido comprobar a través de los resultados expuestos en el presente proyecto se puede afirmar que cumple con la finalidad deseada permitiendo procesar un gran volumen de datos de forma rápida y eficiente que puedan servir de base en futuras investigaciones sobre el comportamiento de la placa euroasiática y el efecto climático en los polos. También los datos obtenidos estarían en línea con la meta 13.1 del ODS13, acción por el clima.

Finalmente conviene mencionar que este desarrollo informático estructurado dispone de margen de actualización y crecimiento siendo posible agregar otras funcionalidades que puedan surgir a demanda de los usuarios.

8. Bibliografía

Chen, W. Hu, C. Gao, S. Chen, Y. Ding, X. (2009). Error correction models and their effects on GPS Precise Point Positioning (41a ed.).

García-Asenjo Villamayor, L. (2005). Geodesia. Universidad politécnica de Valencia.

Torge, W. (2012). Geodesy (4a ed.). De Gruyter.

Heiskanen, W. & Moritz, H. (1985). Geodesia Física. Editorial IGN.

Martín Furones, Á. (2011). Sistema y marco de referencia terrestre. sistemas de coordenadas. Universidad Politécnica de Valencia.

SINEX - Solution (Software/technique) Independent Exchange Format. (2006, 15 de septiembre). https://files.igs.org/pub/data/format/sinex_v210_proposal.pdf [Consulta 25 de febrero de 2022]

Holger, S. Gitlein, O. Denker, H. Müller, J. Timmen, L. (2009). Present rate of uplift in fennoscandia fron GRACE and absolute gravimetry.

Jin, S., & Zhu, W. (2002). A revision of the parameters of the NNR-NUVEL-1A plate.

Kouba, J., & Héroux, P. (2001). Precise Point Positioning Using IGS Orbit and Clock Products.

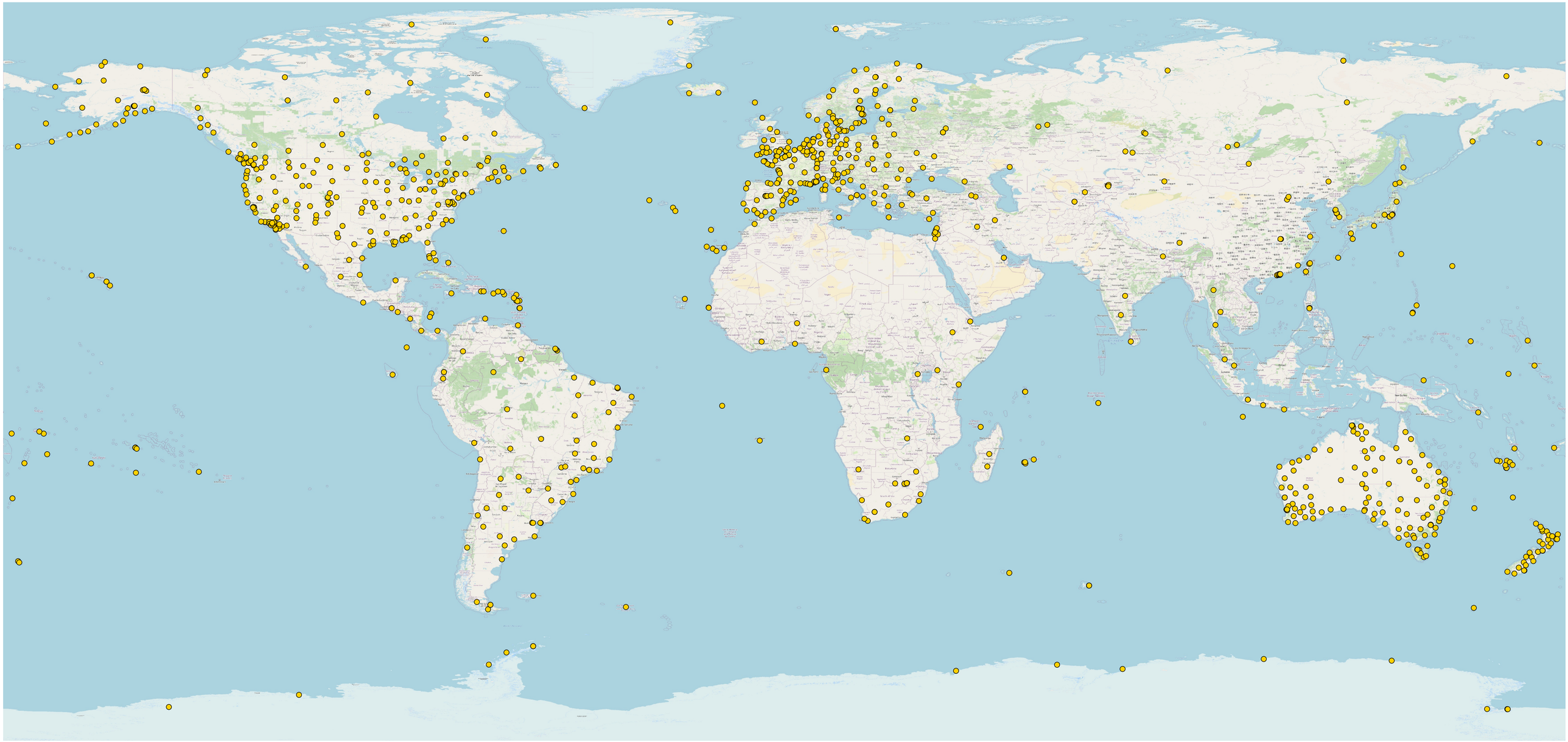
Salazar, D. Sanz-Subirana, J. Hernandez-Pajares, M. (2009). Phase-Based GNSS data processing (PPP) with the GPSTk.

9. Cartografía

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Titulo del proyecto: Evaluación de efectos del cambio climático empleando técnicas de geodesia espacial: Caso de variación en altura de estaciones permanentes GNSS por equilibrio isostático

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA GEODÉSICA CARTOGRÁFICA Y TOPOGRÁFICA



Leyenda

- Estaciones permanentes GNSS

Título del mapa:
MAPA DE SITUACIÓN ESTACIONES PERMANENTES

Flecha del norte:

Autor: Carlos Martínez Montes
Fecha: 30/08/2022

Sistema de referencia: EPSG: 3857 - WGS84/Pseudo-Mercator
ESCALA: 1:66000000

10. Anexos

10.1. Anexo 1

```
 1 import datetime
 2 import geojson
 3 import os.path
 4 import gzip
 5 import shutil
 6 from astropy.time import Time
 7 import math
 8 import unlzw3
 9 from pathlib import Path
10 import pyproj
11 import plotly.express as px
12 import json
13 import os
14 import requests
15 from collections import OrderedDict
16
17 crs = {
18     4258: '+proj=longlat +ellps=GRS80 +no_defs',
19     4346: '+proj=geocent +ellps=GRS80 +units=m +
   no_defs',
20     4326: '+proj=longlat +ellps=WGS84 +datum=WGS84 +
   no_defs',
21     4328: '+proj=geocent +datum=WGS84 +units=m +
   no_defs'
22 }
23
24
25 class Config():
26     __instance = None
27
28     @staticmethod
29     def instance():
30         if Config.__instance is None:
31             Config()
32         return Config.__instance
33
34     def __init__(self):
35         if Config.__instance is None:
36             Config.__instance = self
37
38             if os.name == 'nt':
```

```
39                 self._cddis_dir = os.path.join(
40                     os.getenv('APPDATA'), '.cddis'
41                 )
42             elif os.name == 'posix':
43                 self._cddis_dir = os.path.join(
44                     os.getenv('HOME'), '.cddis'
45                 )
46 
47             try:
48                 os.mkdir(self._cddis_dir)
49             except:
50                 pass
51 
52             try:
53                 project_dir = os.path.join(self.
   _cddis_dir, 'PROJECTS')
54                 os.mkdir(project_dir)
55             except:
56                 pass
57 
58             self._subdirs = ['SNX', 'GEOJSON']
59 
60             config_filename = os.path.join(self.
   _cddis_dir, 'config.json')
61 
62             try:
63                 with open(config_filename, 'rt') as
   config_file:
64                     self._config = json.load(
   config_file)
65             except:
66                 self._save_default_config()
67         else:
68             raise Exception('Multiple Config()
   instances are not allowed')
69 
70     def _save_default_config(self):
71         default = {
72             'url': {
73                 'cddis': 'https://cddis.nasa.gov/
   archive/gnss/products/',
```

```
 74             },
 75             'subdirs': self._subdirs
 76 
 77         }
 78 
 79         self._config = default
 80 
 81         config_filename = os.path.join(self.
    _cddis_dir, 'config.json')
 82 
 83         with open(config_filename, 'wt') as
    config_file:
 84             json.dump(self._config, config_file,
    indent=4)
 85 
 86     def _get_cddis_dir(self):
 87         return self._cddis_dir
 88 
 89     def _get_cddis_subdirs(self):
 90         return self._config['subdirs']
 91 
 92     def _get_cddis_url(self):
 93         return self._config['url']['cddis']
 94 
 95     cddis_dir = property(_get_cddis_dir)
 96     cddis_subdirs = property(_get_cddis_subdirs)
 97     cddis_cddis_url = property(_get_cddis_url)
 98 
 99 
100 class Proyect:
101     def __init__(self, name, data_first, data_last):
102         self._proyect_subdir = []
103         self._project_dir = os.path.join(Config.
    instance().cddis_dir, 'PROJECTS', name.upper())
104         self.__data_first = data_first
105         self.__data_last = data_last
106         self.__route = []
107         self.__route2 = []
108         self.__comp = None
109         self.__des = None
110         self.__doc = None
```

```
111
112         try:
113             os.mkdir(self._project_dir)
114             self._stations = []
115         except FileExistsError:
116             print(f'cddis.Project(): Project {name.
    upper()} already exists')
117
118         for subdir in Config.instance().
    cddis_subdirs:
119             try:
120                 self._proyect_subdir.append(os.path.
    join(self._project_dir, subdir))
121                 os.mkdir(self._proyect_subdir)
122             except:
123                 pass
124         self.__comp = self._proyect_subdir[0]
125         self.__des = self._proyect_subdir[1]
126         self.__doc = self._proyect_subdir[2]
127
128     def list_cddis(url):
129         """function that returns the list of files
    of the selected address within the nasa server"""
130         if url[-1] != '/':
131             url += '/'
132
133         response = requests.get(url + '*?list')
134         if response.text[0] == '<':
135             return []
136         file_list = []
137         for line in response.text.split('\n'):
138             if len(line) == 0:
139                 continue
140             if line[0] == '#':
141                 continue
142             try:
143                 file_list.append(line.split()[0])
144             except:
145                 continue
146
147         return file_list
```

```
148
149     def calc_nom_file(week):
150         """function that returns the name of the
    SINEX file for a specific week"""
151         sec = (week * 7 * 24 * 3600) + 1 * 24 * 3600
152         t_in = Time(sec, format='gps')
153         t_out = Time(t_in, format='iso', scale='utc'
    )
154         t_out2 = str(t_out).split(" ")
155         t_out3 = t_out2[0].split("-")
156         year = t_out3[0]
157         fmt = '%Y-%m-%d'
158         sdtdate = datetime.datetime.strptime(t_out2[
    0], fmt)
159         sdtdate = sdtdate.timetuple()
160         jdate = sdtdate.tm_yday
161         ext2 = year + str(f'{jdate:03d}')
162         count = 0
163         total = 11
164         for element in range(total - len(ext2)):
165             count = count + 1
166         ext3 = ext2 + "0" * count
167         return ext3, year
168
169     def download_cddis(self):
170         """Function that downloads the SINEX files
    from the NASA server"""
171         data_first, data_last = Proyect.
    period_of_time(self)
172         week = data_first
173         while week < data_last:
174             print(week)
175             url = 'https://cddis.nasa.gov/archive/
    gnss/products/'
176             url3 = url
177             a, year = Proyect.calc_nom_file(week)
178             ext = f'0R03SNX_{a}_07D_07D_CRD.SNX'
179             if url[-1] != '/':
180                 url += '/'
181             url += f'{week:04d}'
182
```

```
183             if url[-1] != '/':
184                 url += '/'
185             url += 'repro3'
186 
187             url2 = url
188             if url2[-1] != '/':
189                 url2 += '/'
190 
191             file_list = Proyect.list_cddis(url2)
192             download_list = []
193             count2 = 1
194             count = 1
195             for filename in os.listdir(self.__comp):
196                 if filename in file_list:
197                     week = week + 1
198                     print(f'the file with name: {
    filename} exists')
199                     count = 0
200             if count == 0:
201                 continue
202 
203             for filename in file_list:
204                 if ext != f'{a}_07D_07D_CRD.ALL' and
     ext not in filename:
205                     continue
206                 download_list.append(filename)
207 
208                 r = requests.get(url2 + filename,
    allow_redirects=True)
209                 print(ext)
210 
211                 open(f'{self.__comp}/{filename}', '
    wb').write(r.content)
212                 name_f, extension = os.path.splitext
    (filename)
213                 with gzip.open(f'{self.__comp}/{
    filename}', 'rb') as f_in:
214                     with open(f'{self.__des}/{name_f
    }', 'wb') as f_out:
215                         shutil.copyfileobj(f_in,
    f_out)
```

```
216                 week = week + 1
217                 self.__route.append(f'{self.__des}/{
    name_f}')
218 
219                 count2 = 0
220 
221             a, year = Proyect.calc_nom_file(week)
222             ext = f'{year[2:4]}P{week}.snx'
223 
224             url3 += f'{week:04d}'
225             file_list2 = Proyect.list_cddis(url3)
226             for filename in os.listdir(self.__comp):
227                 if filename in file_list2:
228                     week = week + 1
229                     print(f'the file with name: {
    filename} exists')
230                     count = 0
231             if count == 0:
232                 continue
233             url4 = url3
234             if url4[-1] != '/':
235                 url4 += '/'
236             if count2 == 1:
237                 for filename2 in file_list2:
238                     if ext != f'{year[2:4]}P{week}.
    all' and ext not in filename2:
239                         continue
240                     download_list.append(filename2)
241                     r = requests.get(url4 +
    filename2, allow_redirects=True)
242                     if r is None:
243                         break
244                     open(f'{self.__comp}/{filename2}
    ', 'wb').write(r.content)
245                     uncompressed_data = unlzw3.unlzw
    (Path(f'{self.__comp}/{filename2}').read_bytes())
246                     name_f, extension = os.path.
    splitext(filename2)
247                     open(f'{self.__des}/{name_f}', '
    wb').write(uncompressed_data)
248                     print(ext)
```

```
249                     week = week + 1
250                     self.__route.append(f'{self.
    __des}/{name_f}')
251 
252     def ecef_to_geodetic(ecef, in_crs=4328, out_crs=
    4326):
253         """Converts from ECEF coordinates to
    geodetic longitude-latitude
254         and elipsoidal height
255          Dependencies: This function require package
     pyproj.
256             """
257 
258         ecef_crs = pyproj.CRS.from_proj4(crs[in_crs
    ])
259         geod_crs = pyproj.CRS.from_proj4(crs[out_crs
    ])
260 
261         transformation = pyproj.Transformer.from_crs
    (
262             ecef_crs, geod_crs, always_xy=True
263         )
264 
265         geodetic = transformation.transform(*ecef)
266 
267         return geodetic
268 
269     def cutout(header, route):
270         """cutout is a function that allows you to
    extract the data you need by indicating a header"""
271 
272         with open("txt_file_work.txt", "w+") as file
    :
273             with open(route, 'r') as f:
274                 contador = 0
275                 valor1 = 1000000
276                 valor2 = 1000000
277                 for line in f:
278                     contador = contador + 1
279                     if line.find(f"+{header}") != -1
    :
```

```
                            valor1 = contador
                        if line.find(f"-{header}") != -1
    :
                            valor2 = contador
                        if valor1 < contador < valor2:
                            file.write(line)
                return "txt_file_work.txt"

    def period_of_time(self):

        """function that converts entered dates to
    GPS time"""
        """Dependencies the function needs the
    datetime and math"""

        d_last = self.__data_last.split()
        t_out2 = self.__data_first.split()
        data_pas_sec = datetime.datetime.strptime(
    d_last[0], '%Y-%m-%d').timestamp()
        data_first_sec = datetime.datetime.strptime(
    t_out2[0], '%Y-%m-%d').timestamp()
        gps = datetime.datetime(1980, 1, 6).
    timestamp()
        data_first = int(data_first_sec) - int(gps)
        week_pas = int(data_pas_sec) - int(gps)
        data_first1 = math.floor(data_first / 3600
     / 24 / 7)
        data_last1 = math.floor(week_pas / 3600 / 24
     / 7)
        return data_first1, data_last1

    def obtain_data(route):
        """function that allows to obtain the
    necessary data to later work with them"""
        dictona = []
        path = Proyect.cutout("SITE/ID", route)
        with open(path, "r") as file:
            count = 0
            for line in file:
                if count > 0:
                    a = dict(id=line[1:6].strip(),
```

```
311 description=line[21:44].strip())
312                     dictona.append(a)
313                     count = count + 1
314                 count = count + 1
315 
316         path = Proyect.cutout("SITE/ANTENNA", route)
317         antenna = []
318         ident = []
319         with open(path, "r") as file:
320             count = 0
321             for line in file:
322                 if count > 0:
323                     ident.append(line[1:6].strip())
324                     antenna.append(line[42:63].strip
    ())
325                 count = count + 1
326             count = 0
327             count2 = 0
328             for dicto in range(len(dictona)):
329                 if count >= 0:
330                     if len(dictona) > len(ident):
331                         if dictona[count2]['id'] !=
    ident[count]:
332                             if ident[count + 1] ==
    dictona[count2]['id']:
333                                 count = count + 1
334                                 dictona[count2]['
    Antenna'] = antenna[count]
335                             else:
336                                 if ident[count + 2
    ] == dictona[count2]['id']:
337                                     count = count +
    2
338                                     dictona[count2][
    'Antenna'] = antenna[count]
339                                 else:
340                                     dictona[count2][
    'Antenna'] = 'None'
341                                     count = count -
    1
342                         else:
```

```
343                                 dictona[count2]['Antenna
    '] = antenna[count]
344                         else:
345                             if dictona[count2]['id'] !=
    ident[count]:
346                                 if ident[count + 1] ==
    dictona[count2]['id']:
347                                     count = count + 1
348                                     dictona[count2]['
    Antenna'] = antenna[count]
349                                 else:
350                                     if ident[count + 2
    ] == dictona[count2]['id']:
351                                         count = count +
    2
352                                         dictona[count2][
    'Antenna'] = antenna[count]
353                                     else:
354                                         dictona[count2][
    'Antenna'] = 'None'
355                                         count = count -
    1
356                             else:
357                                 dictona[count2]['Antenna
    '] = antenna[count]
358                     count = count + 1
359                     count2 = count2 + 1
360
361             path = Proyect.cutout("SITE/RECEIVER", route
    )
362             receptor = []
363             ident = []
364             with open(path, "r") as file:
365                 count = 0
366                 for line in file:
367                     if count > 0:
368                         ident.append(line[1:6].strip())
369                         receptor.append(line[42:63].
    strip())
370                     count = count + 1
371                 count = 0
```

```
372             count2 = 0
373             for dicto in range(len(dictona)):
374                 try:
375                     if count >= 0:
376                         if len(dictona) > len(ident
    ):
377                             if dictona[count2]['id'
    ] != ident[count]:
378                                 if ident[count + 1
    ] == dictona[count2]['id']:
379                                     count = count +
    1
380                                     dictona[count2][
    'Receptor'] = receptor[count]
381                                 else:
382                                     if ident[count
     + 2] == dictona[count2]['id']:
383                                         count =
    count + 2
384                                         dictona[
    count2]['Receptor'] = receptor[count]
385                                     else:
386                                         dictona[
    count2]['Receptor'] = 'None'
387                                         count =
    count - 1
388                             else:
389                                 dictona[count2]['
    Receptor'] = receptor[count]
390                         else:
391                             if dictona[count2]['id'
    ] != ident[count]:
392                                 if ident[count + 1
    ] == dictona[count2]['id']:
393                                     count = count +
    1
394                                     dictona[count2][
    'Receptor'] = receptor[count]
395                                 else:
396                                     if ident[count
     + 2] == dictona[count2]['id']:
```

```
397                                             count =
    count + 2
398                                             dictona[
    count2]['Receptor'] = receptor[count]
399                                         else:
400                                             dictona[
    count2]['Receptor'] = 'None'
401                                             count =
    count - 1
402                                 else:
403                                     dictona[count2]['
    Receptor'] = receptor[count]
404                         count = count + 1
405                         count2 = count2 + 1
406                     except:
407                         dictona[count2]['Receptor'] = '
    None'
408
409         path = Proyect.cutout("SOLUTION/ESTIMATE",
    route)
410         coor_x = []
411         errorx = []
412         coor_y = []
413         errory = []
414         coor_z = []
415         errorz = []
416         time = []
417         with open(path, "r") as file:
418             count = 0
419             for line in file:
420                 if count > 0:
421                     if "STAX" in line:
422                         coor_x.append(float(line[47:
    69].strip()))
423                         errorx.append(float(line[69:
    81].strip()) + float(line[47:69].strip()))
424                     if "STAY" in line:
425                         coor_y.append(float(line[47:
    69].strip()))
426                         errory.append(float(line[69:
    81].strip()) + float(line[47:69].strip()))
```

```
427                     if "STAZ" in line:
428                         coor_z.append(float(line[47:
    69].strip()))
429                         errorz.append(float(line[69:
    81].strip()) + float(line[47:69].strip()))
430                         time.append(line[27:40].
    strip())
431                 count = count + 1
432             count = 0
433             for dicto in dictona:
434                 dicto['X'] = coor_x[count]
435                 dicto['ERROR_X'] = errorx[count]
436                 dicto['Y'] = coor_y[count]
437                 dicto['ERROR_Y'] = errory[count]
438                 dicto['Z'] = coor_z[count]
439                 dicto['ERROR_Z'] = errorz[count]
440                 dicto['TIME'] = time[count]
441                 count = count + 1
442         return dictona
443 
444     def snx_epoch_to_datetime(epoch):
445         """Converts SNX epoch to datetime object"""
446 
447         year, day, seconds = map(int, epoch.split(
    ':'))
448 
449         if year > 75:
450             year += 1900
451         else:
452             year += 2000
453 
454         time = datetime.datetime(year, 1, 1) +
    datetime.timedelta(days=day - 1, seconds=seconds)
455         return time
456 
457     def generate_file(self):
458         """function that together with "obtain data
    " generates a geojson file from the passed SINEX
    file"""
459 
460         for file in os.scandir(self.__des):
```

```
461             print(file)
462             dictionaries = Proyect.obtain_data(file)
463             for diction in dictionaries:
464                 value = [diction['X'], diction['Y'
    ], diction['Z']]
465                 value_error = [diction['ERROR_X'],
    diction['ERROR_Y'], diction['ERROR_Z']]
466                 coordinate = Proyect.
    ecef_to_geodetic(value, 4328, 4326)
467                 coordinate_error = Proyect.
    ecef_to_geodetic(value_error, 4328, 4326)
468                 diction['Lon'] = coordinate[0]
469                 diction['ERROR_Lon'] =
    coordinate_error[0] - coordinate[0]
470                 diction['Lat'] = coordinate[1]
471                 diction['ERROR_Lat'] =
    coordinate_error[1] - coordinate[1]
472                 diction['h'] = coordinate[2]
473                 diction['ERROR_h'] =
    coordinate_error[2] - coordinate[2]
474                 date = str(Proyect.
    snx_epoch_to_datetime(diction['TIME']))
475                 diction['TIME'] = date
476                 antenna = diction['Antenna']
477                 diction['Antenna'] = antenna
478                 receptor = diction['Receptor']
479                 diction['Receptor'] = receptor
480                 [diction.pop(key, None) for key in [
    'X', 'Y', 'Z', 'ERROR_X', 'ERROR_Y', 'ERROR_Z']]
481
482             geo = Proyect.df_to_geojson(dictionaries
    )
483             name_f, extension = os.path.splitext(
    file.name)
484             with open(f'{self.__doc}/{name_f}.
    geojson', 'w') as outfile:
485                 geojson.dump(geo, outfile)
486
487     def correction(self):
488         id = []
489         for file in os.scandir(self.__doc):
```

```
490             with open(file, 'r') as outfile:
491                 dictionaries = json.load(outfile)
492                 for stat in dictionaries['features'
    ]:
493                     id.append(stat['properties']['id
    '])
494             final_list = list(OrderedDict.fromkeys(
    id))
495 
496         for stat in final_list:
497             count = 0
498             ant = 'None'
499             rec = 'None'
500             print(stat)
501             dif = 0
502             count4 = 0
503             for file in os.scandir(self.__doc):
504                 name_f, extension = os.path.splitext
    (file.name)
505                 with open(f'{self.__doc}/{name_f}.
    geojson', 'r') as outfile:
506                     dictionaries = json.load(outfile
    )
507                     print(stat)
508                 with open(f'{self.__doc}/{name_f}.
    geojson', 'w') as outfile:
509                     count2 = 0
510                     for dicto in dictionaries['
    features']:
511                         if stat == dicto['properties
    ']['id']:
512                             if count == 0:
513                                 h0 = dicto['
    properties']['h']
514                                 ant = dicto['
    properties']['Antenna']
515                                 rec = dicto['
    properties']['Receptor']
516 
517                             if dicto['properties']['
    h'] - h0 >= 10:
```

```
518                                 continue
519                             if dicto['properties']['
    h'] - h0 <= -10:
520                                 continue
521                             count5 = 0
522                             count6 = 0
523                             if dicto['properties']['
    Antenna'].split()[0] != ant.split()[0]:
524                                 if not dicto['
    properties']['Antenna'] == 'None':
525                                     if not ant == '
    None':
526                                         dif = (dicto
    ['properties']['h'] - hm) + dif
527                                         count5 = 1
528                                         count6 = 1
529
530                             if dicto['properties']['
    Receptor'] != rec:
531                                 if count4 != 0:
532                                     if dicto['
    properties']['Receptor'] != rec2:
533                                         if not dicto
    ['properties']['Receptor'] == 'None':
534                                             if
    count5 == 0:
535                                                 dif
     = (dicto['properties']['h'] - hm) + dif
536
    count4 = 0
537
    count6 = 1
538
539                                 if not rec == 'None'
    :
540                                     rec2 = rec
541                                     count4 = 1
542                                     if not dicto['
    properties']['Receptor'] == 'None':
543                                         if count6
     == 0:
```

```
544                                                 dif = (
    dicto['properties']['h'] - hm) + dif
545                                                 count4
     = 0
546                                 ant = dicto['properties'
    ]['Antenna']
547                                 hm = dicto['properties'
    ]['h']
548                                 rec = dicto['properties'
    ]['Receptor']
549                                 print(dif)
550                                 count = count + 1
551                                 dictionaries['features'
    ][count2]['properties']['h'] = dicto['properties']['
    h'] - dif
552                             count2 = count2 + 1
553                     json.dump(dictionaries, outfile)
554
555     def graphe(self, id):
556         """function that creates a scatter plot
    based on the selected station"""
557         count = 0
558         count2 = 0
559         sum = 0
560         errorh_sum = 0
561         axisy = []
562         axis_errory = []
563         list_year = []
564         dif = 0
565         rec = 'NONE'
566         data2 = None
567         count3 = 0
568         count4 = 0
569         for file in os.scandir(self.__doc):
570             print(file)
571             with open(file, 'r') as outfile:
572                 dictionaries = json.load(outfile)
573                 for dicto in dictionaries['features'
    ]:
574                     if id == dicto['properties']['id
    ']:
```

```
575                             if count == 0:
576                                 h0 = dicto['properties'
    ]['h']
577                                 ant = dicto['properties'
    ]['Antenna']
578                                 rec = dicto['properties'
    ]['Receptor']
579 
580                             if dicto['properties']['h'
    ] - h0 >= 10:
581                                 continue
582                             if dicto['properties']['h'
    ] - h0 <= -10:
583                                 continue
584                             count5 = 0
585                             count6 = 0
586                             # if dicto['properties']['
    Antenna'].split()[0] != ant.split()[0]:
587                             #     if not dicto['
    properties']['Antenna'] == 'None':
588                             #         if not ant == '
    None':
589                             #             dif = (dicto['
    properties']['h'] - hm) + dif
590                             #             count5 = 1
591                             #             count6 = 1
592                             #
593                             # if dicto['properties']['
    Receptor'] != rec:
594                             #     if count4 != 0:
595                             #         if dicto['
    properties']['Receptor'] != rec2:
596                             #             if not dicto['
    properties']['Receptor'] == 'None':
597                             #                 if count5
     == 0:
598                             #                     dif
     = (dicto['properties']['h'] - hm) + dif
599                             #                     count4
     = 0
600                             #                     count6
```

```
600  = 1
601                             #
602                             #     if not rec == 'None':
603                             #         rec2 = rec
604                             #         count4 = 1
605                             #         if not dicto['
    properties']['Receptor'] == 'None':
606                             #             if count6 == 0
    :
607                             #                 dif = (
    dicto['properties']['h'] - hm) + dif
608                             #                 count4 = 0
609
610                             # if dicto['properties']['
    Antenna'].split()[0] != ant.split()[0]:
611                             #     if not dicto['
    properties']['Antenna'] == 'None':
612                             #         if not ant == '
    None':
613                             #             if ant != '
    JPLD/M_R' and dicto['properties']['Antenna'].split
    ()[0] != 'AOAD/M_T':
614                             #                 hm2 = hm
615                             # if hm2 != 0:
616                             #     hm3 = dicto['
    properties']['h'] - hm2 + hm3
617                             #     print(hm3)
618                             #     contad = contad + 1
619                             #     if contad > 3:
620                             #         dif = hm3/contad
621                             #
622                             #         hm2 = 0
623
624                             sum = sum + (dicto['
    properties']['h'] - dif)
625                             errorh_sum = errorh_sum +
    dicto['properties']['error_h']
626                             data = dicto['properties']['
    time'].split()[0].split('-')[0]
627                             if data != data2:
628                                 if count2 != 0:
```

```
629                                     if count3 != 0:
630                                         meanerror =
    errorh_sum / count2
631                                         meant = sum /
    count2
632                                         axisy.append(
    meant - year_dif)
633                                         axis_errory.
    append(meanerror)
634                                         errorh_sum = 0
635                                         list_year.append
    (int(data) - 1)
636                                         count2 = 0
637                                         sum = 0
638                                     else:
639                                         meanerror =
    errorh_sum / count2
640                                         meant = sum /
    count2
641                                         year_dif = meant
642                                         count3 = count3
     + 1
643                                         count2 = 0
644                                         sum = 0
645                                         errorh_sum = 0
646                                         axisy.append(
    meant - year_dif)
647                                         axis_errory.
    append(meanerror)
648                                         list_year.append
    (int(data) - 1)
649                                 else:
650                                     sum = 0
651                                     errorh_sum = 0
652                             data2 = dicto['properties'][
    'time'].split()[0].split('-')[0]
653                             # plt.scatter(list_year,
    axisy)
654                             count2 = count2 + 1
655                             count = count + 1
656                             ant = dicto['properties']['
```

```
656 Antenna']
657                         hm = dicto['properties']['h'
    ]
658                         rec = dicto['properties']['
    Receptor']
659         dicto = {'Year': list_year, 'Height increase
    (m)': axisy}
660         fig = px.scatter(dicto, x="Year", y="Height
    increase(m)", error_y=axis_errory, title=id)
661         # plt.ylabel('height increase m')
662         # plt.xlabel('TIME YEARS')
663         # plt.title(id)
664         return fig
665 
666     def df_to_geojson(dictionar):
667         """function that transforms a list of
    dictionaries into a geojson file"""
668 
669         geojsons = {'type': 'FeatureCollection', '
    features': []}
670         for row in dictionar:
671             feature = {'type': 'Feature', '
    properties': {}, 'geometry': {'type': 'Point', '
    coordinates': []}}
672             feature['geometry']['coordinates'] = [
    row['Lon'], row['Lat']]
673 
674             feature['properties'] = {'id': row['id'
    ], 'description': row['description'], ' error_Lat':
    row['ERROR_Lat'],
675                                      'error_Lon':
    row['ERROR_Lon'], 'h': row['h'], 'error_h': row['
    ERROR_h'],
676                                      'time': row['
    TIME'], 'Antenna': row['Antenna'], 'Receptor': row['
    Receptor']}
677             geojsons['features'].append(feature)
678         return geojsons
679 
680     def filter(self, latitude, hemisphere,
    latitudemax=180):
```

```
681         """function to filter stations based on
    latitude"""
682         geojsons = {'type': 'FeatureCollection', '
    features': []}
683         for file in os.scandir(self.__doc):
684             with open(file, 'r') as outfile:
685                 dictionaries = json.load(outfile)
686                 for dicto in dictionaries['features'
    ]:
687                     if hemisphere == "North":
688                         if dicto['geometry']['
    coordinates'][1] >= float(latitude):
689                             if dicto['geometry']['
    coordinates'][1] <= float(latitudemax):
690                                 geojsons['features'
    ].append(dicto)
691                                 with open(f'{self.
    __doc}/{file.name}', 'w') as outfile:
692                                     geojson.dump(
    geojsons, outfile)
693                     if hemisphere == "South":
694                         if latitude <= 0:
695                             latitude = latitude * -1
696                         if latitudemax <= 0:
697                             latitudemax =
    latitudemax * -1
698                         if dicto['geometry']['
    coordinates'][1] >= float(latitude):
699                             if dicto['geometry']['
    coordinates'][1] <= float(latitudemax):
700                                 geojsons['features'
    ].append(dicto)
701                                 with open(f'{self.
    __doc}/{file.name}', 'w') as outfile:
702                                     geojson.dump(
    geojsons, outfile)
703
704
705 persona = Proyect("Test", "2020-1-1", "2020-1-14")
706 # persona.download_cddis()
707 # persona.generate_file()
```

```
# persona.filter(60,"North",90)
# persona.correction()
plt = persona.graphe("NYAL")
# plt.ylim([-0.05,0.2])
plt.show()

```