OPTIMIZACIÓN Y PORTING DE SIMULADOR DE RECEPTORES DE SISTEMAS GLOBALES DE NAVEGACIÓN POR SATÉLITE PARA ENTORNOS DE COMPUTACIÓN DE ALTAS PRESTACIONES (HPC) Y SOBRE PLATAFORMAS NO PROPIETARIAS

JESÚS ALBERTO MUÑOZ MESA
JULIAN ANDRÉS QUIROGA GARCÍA

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2015

OPTIMIZACÍON Y PORTING DE SIMULADOR DE RECEPTORES DE SISTEMAS GLOBALES DE NAVEGACIÓN POR SATÉLITE PARA ENTORNOS DE COMPUTACIÓN DE ALTAS PRESTACIONES (HPC) Y SOBRE PLATAFORMAS NO PROPIETARIAS

JESÚS ALBERTO MUÑOZ MESA
JULIAN ANDRÉS QUIROGA GARCÍA

Trabajo de grado para optar al título de Ingeniero de Sistemas

Director:
Raúl Ramos Pollán
Ingeniero de Sistemas PhD.

UNIVERSIDAD INDUSTRIAL DE SANTANDER
FACULTAD DE INGENIERÍAS FÍSICO-MECÁNICAS
ESCUELA DE INGENIERÍA DE SISTEMAS E INFORMÁTICA
BUCARAMANGA
2015

Índice general

Introducción **14**

1 PLANTEAMIENTO DEL PROBLEMA **14**

2 JUSTIFICACIÓN **15**

3 OBJETIVOS **16**
3.1 OBJETIVO GENERAL 16
3.2 OBJETIVOS ESPECÍFICOS 16

4 ESTADO DEL ARTE **17**
4.1 INVESTIGACIÓN RELACIONADA 17
4.2 SIMULADORES GNSS 18
4.2.1 simGEN 18
4.2.2 GNSS Signal Architect 19
4.2.3 GRANADA 19
4.2.4 PILDO GSIM 19
Migración Matlab - Python 20

5 MARCO TEÓRICO **21**
5.1 SISTEMA GLOBAL DE NAVEGACIÓN POR SATÉLITE (GNSS) 21
Fuentes de imprecisión en GNSS 22
Aplicaciones en aviación civil 26
5.1.1 Sistemas de Aumentación 26
5.2 SIMULADOR GSIM 28

6 METODOLOGÍA **29**
6.1 METODOLOGÍAS ÁGILES 29
6.2 METODOLOGÍA FDD (DESARROLLO BASADO EN FUNCIONES) . . 29
6.3 SISTEMA DE GESTÍON DE VERSIONES DE CÓDIGO 30
6.4 RECURSOS SOFTWARE 32

7 VIABILIDAD **33**

8 DESARROLLO DEL PROYECTO **34**

8.1 FASE I: ANÁLISIS DEL SIMULADOR 34
8.2 Fase II: MIGRACIÓN DE LAS FUNCIONES ESENCIALES 39
Migración de las funciones de conversión 40
Migración de las funciones de cómputo 44
Migración de las funciones restantes 46
8.3 FASE III: MIGRACIÓN DE LAS FUNCIONES DE ALTO NIVEL 48
Migración de las funciones de cómputo (satélites) 49
Migración de las funciones de generación de errores 51
Migración de las funciones de escritura 52
Migración de las funciones extensas 53
Descripción Caso de uso II 54
8.4 OPTIMIZACIÓN DEL SIMULADOR 56

9 CONCLUSIONES **60**

10 RECOMENDACIONES **61**

REFERENCIAS **62**

Anexos **65**

Índice de figuras

Figura 1 Cálculo de posición del receptor 22
Figura 2 Efectos de la ionosfera sobre sisemas GNSS 24
Figura 3 Visión general de un sistema GBAS 27

Figura 4 Autenticación pildo 30
Figura 5 Revisión SVN 31
Figura 6 Revisión svn con navegador web 31

Figura 7 Banderas caso de uso I 34
Figura 8 Modelo Klobuchar 35
Figura 9 Resultado Perfilador de Matlab 36
Figura 10 Código para crear archivos prueba 39
Figura 11 Resultado prueba xyz2llh 41
Figura 12 Resultado prueba llh2xyz 41
Figura 13 Resultado prueba xyz2enu 42
Figura 14 Resultado prueba gpst2utc 43
Figura 15 Resultado prueba convertObst2D 43
Figura 16 Resultado prueba allan_plot_to_Q 44
Figura 17 Resultados prueba resize_svid_svmat 45
Figura 18 Resultado prueba compute_mask_angle 45
Figura 19 Resultado prueba computeSvPosRin 45
Figura 20 Resultado prueba clock_states_F_and_Q 46
Figura 21 Resultado prueba ionocorr 46
Figura 22 Resultado prueba createRinNavV3Msg 47
Figura 23 Resultado prueba genSatPosition 50
Figura 24 Resultado prueba computeSvTimeTrans 50
Figura 25 Resultado prueba genSvSnr 51
Figura 26 Resultado prueba update_user_clock_errors 51
Figura 27 Verificación de archivos Rinex con diff 52
Figura 28 Resultado en pantalla del main 54
Figura 29 Banderas caso de uso II 55
Figura 30 Resultados Pruebas caso II 55

Índice de tablas

Tabla 1	Especificaciones simGEN	18
Tabla 2	Especificaciones GNSS Signal Architect	19
Tabla 3	Especificaciones GRANADA	19
Tabla 4	Especificaciones PILDO GSIM	20
Tabla 5	Estimación de los errores en un sistema GNSS	25
Tabla 6	Perfilador,funciones críticas	36
Tabla 7	Clasificación de Funciones GSIM	37
Tabla 8	Características de las muestras.	38
Tabla 9	Orden de Migración fase II	40
Tabla 10	Descripción funciones de alto nivel	49
Tabla 11	Tiempos antes de optimización caso I	56
Tabla 12	Tiempos antes de optimización caso II	57
Tabla 13	Tiempos después de optimizar en python con archivo de trayectoria pequeño	58
Tabla 14	Resultado optimizado caso I con archivo de trayectoria grande	59
Tabla 15	Resultado optimizado caso II con archivo de trayectoria grande	59

Índice de Anexos

Anexo A. RESULTADOS DEL PERFILADOR DE MATLAB 69
Anexo B. REGISTRO DE REVISIONES DE SVN 74
Anexo C. PLANTILLA PARA PRUEBA DE FUNCIONES 75

RESUMEN

Título: OPTIMIZACÓN Y PORTING DE SIMULADOR DE RECEPTORES DE SISTEMAS GLOBALES DE NAVEGACIÓN POR SATÉLITE PARA ENTORNOS DE COMPUTACIÓN DE ALTAS PRESTACIONES (HPC) Y SOBRE PLATAFORMAS NO PROPIETARIAS.[*]

Autores:
Jesús Alberto Muñoz Mesa
Julian Andrés Quiroga García[**]

Palabras Clave: Python, GNSS, optimización, migración.

Resumen: Con la puesta en marcha de proyectos alternativos a los tradicionales GPS y GLONASS, la perspectiva del GNSS se ha ampliado y la necesidad de intercompatibilidad ha crecido a la par. Tomando en consideración el alto costo del hardware receptor, la dificultad para su despliegue y la incertidumbre intrínseca a su operación, surge el afán por tener simuladores software que permitan desarrollar esta actividad en un ambiente más controlado; sin embargo, son contados los simuladores estables con soporte multiconstelación que en la actualidad se encuentran disponibles. La empresa española Pildo Labs. ofrece una solución sólida, GSIM, la cual ha sido probada por sus clientes y socios colaborativos entre los que se cuentan entidades de gran renombre como la Agencia europea de GNSS (GSA), Eurocontrol, Helios y, a nivel local, Aerocivil. A pesar de las buenas prestaciones del software de Pildo, su distribución y uso afronta una barrera que no puede ser ignorada: Su ejecución está atada a un runtime específico de Matlab o al costo de una licencia del mismo cuyo precio mínimo es de 45 dólares (versión estudiantil). En este proyecto se dio solución a este inconveniente, portando el simulador a python, un lenguaje libre con licencia GNU; del mismo modo se aprovechó para hacer algunas optimizaciones iniciales sobre los tiempos de ejecución y se plantearon algunas recomendaciones para futuras versiones.

[*]Trabajo de Grado en la Modalidad de Investigación.

[**]Facultad de Ingenierías Físico Mecánicas. Escuela de Ingeniería de Sistemas e Informática. Director PhD. Raúl Ramos Pollán.

ABSTRACT

Title: Optimization and porting of global navigation satellite systems simulator for high performance computing environments (HPC) and nonproprietary platforms.[*]

Author:
Jesús Alberto Muñoz Mesa
Julian Andrés Quiroga García[**]

Keywords: Python, GNSS, optimization, porting.

Description: With the advent of alternative projects to traditional GPS and GLONASS, the perspective of GNSS has broaden and the need for intercompatibility has grown with it. Taking into account the high costs of hardware receivers, the deployment difficulty and the intrinsic uncertainty of its operation, the eagerness for software simulators allowing to develop this activity in a more controlled environment has risen; moreover, very few stable multi constellation simulators are available these days. Spanish company Pildo Labs. offers a solid solution, GSIM, which has been proven by its customers and collaborative partners among whom we can find renowned entities such as European GNSS Agency (GSA), Eurocontrol, Helios and, locally, Aerocivil. Besides good performance of Pildos software, its distribution and use face a barrier which cannot be ignored: Its execution is tied to Matlab specific runtime or the license cost whose minimum price is 45 USD (student version). In this project a solution to this problem was given, porting the simulator to Python, a free language under GNU license; in the same way, some initial optimizations on running times were carried out and some recommendations for future versions were given.

[*]Undergraduate final project, research modality.

[**]Physico-Mechanical Engineering Faculty. Systems Engineering and Computer Science School. Director PhD. Raúl Ramos Pollan.

INTRODUCCIÓN

Los orígenes de los sistemas globales de navegación por satélite (GNSS) están en el proyecto NAVSTAR GPS, en principio pensado para uso exclusivamente militar, pero puesto a disposición de la población civil por el presidente Reagan en el año 1983. Desde aquel entonces los usos se han multiplicado y la exigencia de exactitud y precisión ha aumentado con ellos; así mismo, otros sistemas GNSS han entrado a la escena mejorando la disponibilidad de satélites al precio de establecer la necesidad de permitir compatibilidad entre constelaciones.

Los sistemas GNSS funcionan a modo general mediante triangulación, cada satélite conserva con gran exactitud un registro de su trayectoria en función del tiempo (denominada efemérides) que envía periódicamente a receptores ubicados en la Tierra, quienes la reciben, procesan y comparan contra sus propios registros. Para que el proceso sea exitoso, es necesario que el receptor pueda ver por lo menos cuatro satélites de una misma constelación (cinco si desea valerse de varias de ellas).

Bajo condiciones ideales, es decir, con un mínimo de interferencias externas, el geoposicionamiento puede virtualmente satisfacer los requisitos de precisión y exactitud de cualquier aplicación; sin embargo, los usos más críticos del GNSS no suelen reducirse a llanuras a cielo abierto en regiones con nula actividad atmosférica ni escenarios similares donde las predicciones resultarían casi perfectas. En virtud de lo anterior, investigadores han desarrollado técnicas y modelos que permiten ajustar las posiciones calculadas, minimizando el error respecto a los valores esperados y, aunque han conseguido buenos resultados, es un trabajo permanente que aún hoy tiene lugar

Como parte del proceso, una vez se ha elaborado un prototipo para mejorar la señal GNSS se necesita probar su efectividad. Dos opciones se barajan: Hacer pruebas utilizando receptores físicos, cuyos costos y disponibilidad suele estar condicionada, o mediante software, más específicamente, simuladores. La empresa PILDO LABS ofrece una solución utilizando este último enfoque, cuya eficacia ha sido probada por sus usuarios, pero con el limitante de que para ser ejecutado precisa de Matlab y este último requiere a su vez de una licencia de uso. A la luz de la creciente importancia que ha ganado la investigación en este campo dentro del ámbito académico, se planteó portar el simulador a un lenguaje libre (en este caso python), e igualmente aprovechar para optimizar los tiempos de ejecución del mismo.

Se comienza profundizando en el problema, planteando los objetivos y justificando la elaboración de este proyecto; a continuación se contextualiza a través de trabajos previos relacionados y presentando algunas aplicaciones disponibles. Antes de dar paso al desarrollo como tal, se exponen brevemente las generalidades de GNSS y de los len-

guajes de programación utilizados y se describen los recursos de los que se dispuso y la metodología seguida. En la parte central del documento se relaciona el proceso seguido para portar el simulador, desde los preliminares hasta la fase de pruebas. Finalmente se concluye en base a los resultados obtenidos y se presentan recomendaciones para futuras mejoras.

1. PLANTEAMIENTO DEL PROBLEMA

Con el proyecto titulado GALILEO presentado por la Unión Europea para así poder entrar en el negocio de los sistemas de posicionamiento global por satélite (GNSS) el cual estaba siendo liderado por Estados unidos con GPS y la federación Rusa con GLONASS se abrió las puertas a diferentes entidades conocedoras del tema con nuevas ideas e intereses para generar propuestas de valor abiertas a complementar el trabajo colaborativo que prestan los continuos avances tecnológicos de la época.

GNSS va mucho más allá del simple mapeo y posee aplicaciones en las mas diversas áreas, desde una simple transacción bancaria con la sincronización de los relojes, hasta la vigilancia de cultivos y la seguridad civil en operaciones de rescate; y aquí es donde aparece PILDO LABS con todo su repertorio y conocimiento buscando transformar ciencia en tecnología.

PILDO LABS ofrece una amplia gama de servicios destinados a las telecomunicaciones y las tecnologías de la información dentro de los que se destacan validación y certificación de nuevos sistemas GNSS usados principalmente en la aviación civil, plataformas móviles para la navegación por satélite, y diseño e implementación de algoritmos para la observación terrestre.

Ya que su misión principal es garantizar que sus servicios prestados sean los mejores han tomado la decisión de no solo invertir en infraestructura y tecnología sino también apostarle al conocimiento y gracias a esto nació una de sus herramientas software GSIM desarrollada en Matlab la cual les ayudaría a simular las mediciones hechas por un receptor en tiempo real, y su vez acoplando los diferentes modelos de ajuste para los distintos errores de medición generados por los efectos atmosféricos tales como la ionosfera y la troposfera poder disminuir al mínimo el error de dicha medición que está en un rango de (1 a 3 metros) generando mayor precisión y confianza en el cálculo de su posición, pero la mayoría de las veces el tiempo de cómputo no es inmediato,lo cual causa un estancamiento al momento de realizar su trabajo de manera óptima y eficaz.

La perspectiva de este proyecto consiste en tomar el simulador GSIM portarlo al lenguaje python para así poder ser optimizado y usado sobre plataformas no propietarias, y aprovechando la libertad de licencia correrlo en GUANE y ver la mejora en sus tiempos de servicio.

2. JUSTIFICACIÓN

Las mediciones de los Sistemas Globales de Navegación por Satélite (GNSS) son susceptibles a perturbaciones de diferente naturaleza, dentro de estas cabe resaltar los efectos producidos por los campos electromagnéticos de la ionosfera.

Como alternativa de solución, se han venido desarrollando algoritmos que permiten generar modelos que representan algunos de estos problemas, los cuales precisan de una respectiva validación antes de su puesta en marcha.

Para validar estos algoritmos existen simuladores que apoyan dicho proceso omitiendo la necesidad de contar con receptores físicos en tiempo real, lo cual implica una reducción en tiempo, costos y demás complicaciones que esta tarea conlleva. Entre los pocos en el mercado, uno de los más destacados y en el que en este caso nos enfocaremos es el ofrecido por PILDO Labs el cual se encuentra desarrollado en Matlab.

Partiendo de este hecho es donde nace nuestra propuesta, en la que buscamos generar una versión paralela a GSIM basada en un lenguaje de enfoque científico y de licencia pública como lo es python, ya que su principal problema es la lentitud por como esta implementado y por el hecho de usar Matlab que a su vez genera otros problemas tales como:

- La distribución del software a terceros es pesada por el hecho de usar Matlab.
- La implementación de una GUI está limitada por el hecho de poder llamar a Matlab.
- Si se compila un ejecutable se debe también distribuir el MatLAB Runtime lo cual lo vuelve tedioso por el manejo de versiones.

Y al final teniendo el código portado y el original se miden y comparan tiempos de simulación y se plantean posibles estrategias de optimización para aprovechar la arquitectura de Guane.

3. OBJETIVOS

3.1. OBJETIVO GENERAL

Portar y optimizar el simulador GSim de receptores GNSS (Global Navigation Satellite Systems) para su utilización sobre plataformas no propietarias y aprovechamiento de sistemas de computación de altas prestaciones.

3.2. OBJETIVOS ESPECÍFICOS

- Portar el código de GSim desde Matlab a una plataforma no propietaria y que pueda ser usada y optimizada en sistemas HPC, especialmente en la infraestructura Guane de la Unidad de Supercómputo y Cálculo Científico de la UIS, con arquitectura multi-core y multi-GPU.
- Desarrollar tests unitarios para los distintos componentes de GSim y un framework para la ejecución de los mismos a través del cual se pueda verificar la integridad funcional del software original, portado o sus futuras versiones.
- Medir y comparar el rendimiento del código original y del código portado.
- Plantear posibles estrategias de optimización del código portado sobre la arquitectura de Guane.

4. ESTADO DEL ARTE

Antes de entrar en materia, es importante comprender el estado actual de la investigación en torno a la simulación de constelaciones GNSS. De manera análoga, es fundamental conocer la oferta de la que dispone el mercado en el área de interés a fin de ubicar apropiadamente al lector dentro del contexto en que yace el presente trabajo. Finalmente, para ofrecer una ilustración completa del estado del arte, se consideró necesario destacar algunos precedentes en la migración de software entre Matlab y Python, de los cuales además se extrajeron consideraciones que resultaron especialmente útiles durante el proceso de desarrollo.

4.1. INVESTIGACIÓN RELACIONADA

Los primeros vestigios de investigación sobre simulación GNSS se remontan a finales del milenio pasado, más específicamente al último lustro. Uno de los primeros trabajos en el campo fue el de Lazar et al.[1] quienes identificaron la necesidad de disponer de un medio eficiente para probar el creciente número de esquemas de modulación que se estaban desarrollando y diseñaron un simulador de señales GPS que funcionaba a partir de arreglos de compuertas programables.

Por la misma fecha Ward [2] modeló un receptor GNSS valiéndose del método de Monte Carlo, el cual incluéa las características principales que se suelen tener en cuenta en el diseño de sus contrapartes en hardware. Bajo un enfoque similar al del proyecto anterior, Winkel et al. [3] dieron vida a SNSS (Simulated Navigation Satellite System), que además de incluir la posibilidad de simular las prestaciones habituales de los receptores, permitía simular sistemas GNSS completos.

Con la llegada del nuevo milenio y, de su mano, del proyecto GALILEO, los simuladores GNSS híbridos vieron la luz. En 2004 Constantinescu, Landry e Ilie [4] describieron un simulador SDN híbrido GPS/GALILEO, cuya implementación estuvo finalizada para el año siguiente; éste se valéa de Matlab/SIMULINK para su funcionamiento e incluía soporte para administración remota mediante Matlab Web Server. Entre las características clave de dicho simulador, desarrollado en la Escuela Tecnológica Superior de Montreal, se contaban la posibilidad de simular señales y trayectorias de los satélites, receptores y las perturbaciones habituales (retardos ionosféricos y troposféricos, efecto Doppler, interferencias debido a trayectorias múltiples, entre otros) de los sistemas GPS y GALILEO (por separado y en conjunto).

Casi en simultáneo, en el marco del GAlileo Receiver Development Activities (GARDA), Deimos Space desarrolló GRANADA, otro simulador con soporte para GALILEO

y GPS que usaba Matlab/Simulink. En los años subsecuentes numerosos estudios fueron conducidos utilizando este simulador, incluyendo comparaciones entre GPS y GALILEO [5], verificación de modelos [6], [7] e incluso, evaluaciones de su efectividad [8].

En años más recientes se han desarrollado proyectos que incluyen soporte para rangos más amplios de constelaciones y sistemas de aumentación. Albart publicó en 2008 la implementación de un simulador capaz de trabajar con GPS, GALILEO y SBAS [9]; en 2010, Bisnath, Dolgasnky y Szeto desarrollaron para su estudio sobre procesamiento de datos multi-GNSS un simulador (MGOS) con soporte para las cuatro principales constelaciones GNSS: GPS, GLONAS, GALILEO y COMPASS [10].

4.2. SIMULADORES GNSS

Mientras abundan simuladores multi-constelación implementados en hardware, son pocas las soluciones softwares confiables disponibles en el mercado. A continuación se presentan algunas de ellas, cada una acompañada de una tabla con una breve descripción de sus características y funcionalidades principales.

4.2.1. simGEN

Tabla 1 Especificaciones simGEN

Desarrollador	Spirent Communications plc.
Requisitos para su ejecución	Disponible únicamente en simuladores manufacturados por Spirent.
Señales GNSS soportadas	GPS: L1, L2, L5. GLONASS: E1, E5ab, E6. GALILEO: L1, L2. BeiDou-2: B1, B2. SBAS: L1, L5. QZSS.
Formatos de entrada/salida soportados	NMEA, RTCM, YUMA, RINEX, SEM, A-GNSS.

Una descripción detallada de sus características puede consultarse en la página de Spirent o en su manual de usuario [11].

Tabla 2 Especificaciones GNSS Signal Architect

Desarrollador	Navsys corporation.
Requisitos para su ejecución	Compatible con Linux y Windows, precisa de un SDR con capacidades de radio GNU.
Señales GNSS soportadas	GPS: L1, L2. GLONASS: E1, E5ab. GALILEO: En desarrollo. BeiDou-2: En desarrollo. SBAS: En desarrollo. QZSS: En desarrollo.
Formatos de entrada/salida soportados	NMEA, YUMA, .dsf (Nativo de Navsys), binarios, .agl (definidos por el usuario).

4.2.2. GNSS Signal Architect

Una caracterización más profunda se encuentra disponible en la página de Navsys, incluidas las especificaciones del sistema [12]

4.2.3. GRANADA

Tabla 3 Especificaciones GRANADA

Desarrollador	Deimos Space.
Requisitos para su ejecución	Requiere de Matlab/Simulink, funciona únicamente en Windows.
Señales GNSS soportadas	GPS: L1, L2, L5. GLONASS: E1, E5ab, E6
Formatos de entrada/salida soportados	ASCII, entradas numéricas, RINEX.

A pesar de tener varios años encima es un proyecto aún vigente, sus especificaciones a un mayor nivel de detalle están disponibles en la página de Deimos Space [13].

4.2.4. PILDO GSIM

Más detalles acerca del simulador y la compañía se pueden consultar en la página oficial de la misma [14].

Existen además un sinnúmero de simuladores monoconstelación, que en su mayoría

Tabla 4 Especificaciones PILDO GSIM

Desarrollador	Pildo Labs.
Requisitos para su ejecución	Requiere de Matlab, trabaja tanto en Windows como en plataformas basadas en Unix.
Señales GNSS soportadas	GPS: L1, L2, L5. GLONASS: E1, E5ab
Formatos de entrada/salida soportados	ASCII, IONEX, YUMA, RINEX.

ofrecen soporte para GPS y funcionan bajo Matlab. Algunas alternativas completamente libres han comenzado a desarrollarse, como el caso de GPS GYAN [15], sin llegar a concluirse (el código está disponible en sourceForge).

Migración Matlab - Python La discusión presentada en el trabajo de Jurica y Van Leeuwen [16] fue bastante esclarecedora en lo que respecta a los principales retos que portar código de Matlab a python supone. El caso de ClawPack es un antecedente interesante que demuestra cómo soluciones implementadas en Python pueden ofrecer la misma funcionalidad que otras en Matlab e incluso hacerlo de mejor forma [17].

En el trabajo de Airey. Sullivan y Velázquez [18] se incluye la traducción de un modelo del patrón de radiación de AMISR donde, de forma similar a los simuladores GNSS, la precisión es un factor crítico; parte de la sección 3 de dicho artículo se ocupa de lo concerniente a la precisión en ambos lenguajes. Otros trabajos menores se encuentran documentados en la red, de estos se han considerado fundamentalmente las recomendaciones y advertencias que los mismos autores han publicado.

5. MARCO TEÓRICO

5.1. SISTEMA GLOBAL DE NAVEGACIÓN POR SATÉLITE (GNSS)

Un sistema global de navegación por satélite (GNSS) consiste en una constelación de satélites orbitando la tierra a una altura promedio de 20 000 km, la cual emite señales que permiten a pequeños receptores electrónicos determinar su ubicación (longitud, latitud y altitud). El principio de posicionamiento se basa en resolver un problema elemental de geometría, comenzando con las distancias a un conjunto mínimo de cuatro satélites (Distancias que son medidas por el receptor utilizando las señales emitidas por el satélite) y coordenadas conocidas [19].

La computación en paralelo es el uso simultáneo de varios procesadores para resolver un problema computacional. El problema se divide en varias partes que serán resueltas concurrentemente, cada una de estas partes se subdivide en series de instrucciones.

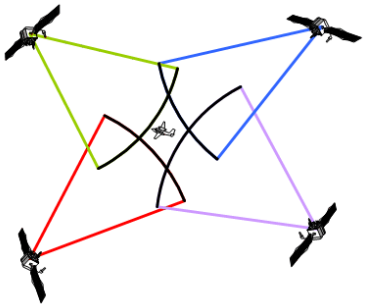
Los principios en un sistema GNSS son:

- Las coordenadas de los satélites GNSS son calculadas a partir de las efemérides transmitidas.
- Las distancias entre el receptor y los satélites son calculadas usando la propagación en el tiempo de una onda electromagnética desde el satélite al receptor. Este tiempo, multiplicado por la velocidad de la luz, nos da una medida de la distancia (pseudorango) entre ellos [28].
  Dada la distancia entre cuatro satélites y el receptor, es posible determinar la posición del receptor, calculando el punto de intersección entre los pseudorangos.

Además de aplicaciones militares, los sistemas GNSS están siendo usados para aplicaciones comerciales tales como agrimensura, mapeo, agricultura, transporte, control de máquinas, navegación marítima, navegación vehicular, comunicaciones móviles y muchos otros sectores. En particular, el uso de los sistemas GNSS en aplicaciones civiles está generalizado y cada sistema GNSS en operación o construcción posiciona a las aplicaciones civiles como uno de los usos más importantes de los sistemas GNSS.

La comunidad de aviación civil ha puesto el mayor esfuerzo en la racionalización y estandarización de los parámetros de desempeño y requerimientos del posicionamiento (navegación), especificando así el llamado Required Navigation Performance (RNP) que un sistema de navegación aérea debe cumplir:

Figura 1 Cálculo de posición del receptor



RNP: Una declaración de la precisión del desempeño, integridad, continuidad y disponibilidad de la navegación necesarias para operaciones sobre un espacio aéreo definido.

Los cuatro parámetros utilizados para caracterizar el desempeño de GNSS (RNP) son:

- Precisión: Es una medida de la desviación de la salida de la navegación contra el valor esperado. Más precisamente, la precisión es una cantidad estadística asociada con la distribución probabilística del error de navegación.
- Integridad: Es la medida de la confianza que puede ponerse en la exactitud de la información suministrada por el sistema de navegación. La integridad incluye la habilidad del sistema de proveer advertencias temporizadas a los usuarios cuando el sistema no debería ser usado para navegación.
- Continuidad: Es la habilidad del sistema como un todo para llevar a cabo su función sin interrupción durante la operación prevista. Más específicamente, la continuidad es la probabilidad de que el desempeño del sistema especificado se mantendrá mientras dure la fase de operación, presumiendo que el sistema estará disponible al principio de dicha fase.
- Disponibilidad: Es la fracción del tiempo de navegación que el sistema puede utilizarse (como es determinado por la satisfacción de los requisitos de precisión, integridad y continuidad) antes de que la aproximación inicie.

5.1.1. Fuentes de imprecisión en GNSS. Una situación ideal incluye la perfecta sincronización de los satélites, órbitas perfectas, velocidad constante de propagación de la señal y ausencia de interferencias. Cualquier desviación de este escenario ideal puede contribuir a errores en la medida de las distancias. A continuación se presentan las principales fuentes de imprecisión en GNSS.

Reloj satelital: La sincronización es un factor crítico porque un nanosegundo de imprecisión en el reloj satelital resulta en cerca de 30 cm de error en la medida de la distancia a ese satélite. A pesar de que están diseñados con relojes atómicos muy precisos, hay desviaciones y ruidos que son modelados e incluidos como parte del mensaje de difusión [20].

Error de la órbita satelital: Las órbitas de los satélites son monitorizadas continuamente desde varias estaciones alrededor de la tierra y su ubicación predicha es transmitida a los satélites, que a su vez los transmiten a los receptores. La historia de GNSS ha mostrado, hasta ahora, que la precisión de la predicción orbital es del orden de unos cuantos metros. Esto inducirá a un par de metros de error en la computación de la posición.

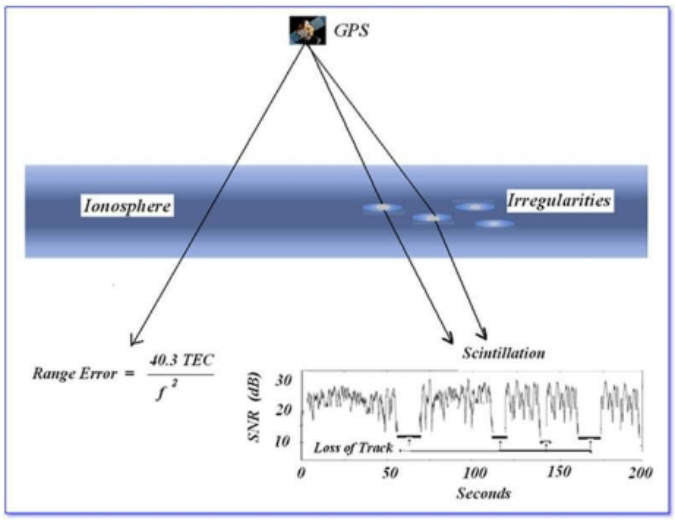
Errores atmosféricos: Ionosfera: Las señales GNSS pasan a través de un espacio cuasi-vacío, para luego pasar a través de las diferentes capas de la atmósfera hacia la tierra. Las ondas electromagnéticas siguen un camino estrictamente lineal solo en el vacío o en medios perfectamente homogéneos, cualquier substancia o imperfección en el medio de propagación produce una desviación por el camino más corto, incrementando el tiempo de viaje y afectando los cálculos de la distancia. Los viajes a través de la atmósfera introducen este tipo de errores en el tiempo de propagación [21].

Para aplicaciones GNSS, las regiones atmosféricas de interés son principalmente dos: La troposfera y la ionosfera. La troposfera va desde la superficie terrestre hasta entre 12 y 20 km de altitud, mientras la ionosfera comprende el rango de 75 a 500 km.

La ionosfera es un escudo de electrones y átomos y moléculas eléctricamente cargados que rodean la Tierra. Los electrones libres en la ionosfera afectan la propagación de las ondas de radio, afectando las señales GNSS de dos formas:

- Rango de error: El rango de error resultante introducido por la ionosfera terrestre puede variar desde menos de 1 m hasta 100 m y cambia con la hora del día, la temporada, la ubicación del receptor en la superficie terrestre, la dirección de avistamiento, la actividad solar y el estado del campo magnético de la Terra.
- Centelleo: Son fluctuaciones rápidas de fase y amplitud de señales satelitales de radio y pueden tener un efecto considerable en el desempeño de la geodesia satelital, la navegación y los sistemas de comunicación. En GNSS, el centelleo puede reducir la precisión de las medidas. A veces, la amplitud del centelleo puede ser tan intenso que el receptor pierde el enfoque de la señal. Si un número de satélites son afectados simultáneamente, el posicionamiento puede volverse imposible.
- Este fenómeno afecta especialmente las señales GNSS en regiones de latitud baja (alrededor de la línea del Ecuador) [22].

Figura 2 Efectos de la ionosfera sobre sisemas GNSS



Como la ionosfera es un medio dispersivo, la refracción de las señales GNSS depende de sus frecuencias (como el inverso cuadrado). Esta dependencia en la frecuencia de la señal permite reducir el efecto de la misma usando medidas bifrecuenciales [5]. Pero se estima que cerca del 75 % de los receptores son monofrecuenciales, por lo tanto es necesario aplicar un modelo de predicción ionosférico para remover (todo lo posible) este efecto cuyo alcance puede abarcar varias decenas de metros.

Por ejemplo, los sistemas GPS difunden los parámetros del modelo ionosférico de Klobuchar para usuarios de monofrecuencia [29]. Este modelo de difusión se basa en una aproximación empírica, e incluye un desplazamiento vertical dependiente de un valor constante durante la noche y un coseno medio en el día. Se estima que reduce cerca del 50 % del rango de error ionosférico RMS en algunas regiones. Otros sistemas, dígase los satélites Galileo, utilizan el modelo ionosférico de NeQuick, el cual es otro modelo empírico basado en datos recogidos. Estos modelos se sustentan en observaciones y medidas almacenadas, pero fallan en la región ecuatorial. Es necesario mejorar el uso de esta información para incrementar la precisión en esta región [23].

Errores atmosféricos: Troposfera: Comparado con el efecto ionosférico, el efecto troposférico es casi un orden de magnitud menor. Esta región tiene un índice de refracción que varéa con la altitud. Dado que el índice de refracción troposférico es mayor que uno, atravesar esta región causa un corrimiento de grupo en las señales de GNSS [6]. La característica principal de la troposfera es que ésta es un medio no dispersivo con respecto a ondas electromagnéticas de hasta 15 GHz, es decir, los efectos troposféricos no son dependientes en frecuencia para las señales GNSS. Una consecuencia inmediata de ser un desplazamiento no dependiente de la frecuencia es que la refracción troposférica no puede ser removida por combinación de medidas bifrecuenciales (como es hecho con

la ionosfera). Así, la única forma de mitigar el efecto troposférico es utilizar modelos o estimarlo a partir de los datos de las observaciones. Afortunadamente, la mayor parte de la refracción troposférica (casi el 90 %) proviene de componentes hidrostáticos predecibles, y puede ser fácilmente modelada.

Multicamino: Adicionalmente, el efecto multicamino puede ser crítico, particularmente para la navegación aérea. El fenómeno de propagación multicamino resulta en llegadas a la antena receptora por dos o más caminos, debido a las múltiples reflexiones sobre la ruta satélite receptor. El efecto de varios caminos causa la presencia de múltiples réplicas desfasadas y atenuadas de la señal principal en el lado del receptor. Así, estas señales pueden causar interferencia constructiva y destructiva en la antena receptora, afectando de este modo, la calidad de la señal recibida. No obstante, la señal GNSS está diseñada para resistir la interferencia de señales multicamino con diferencias de desplazamiento mutuo superiores a 1 s. De forma complementaria, una ganancia aún mayor puede obtenerse utilizando software específico de procesamiento de señales o dispositivos en las antenas receptoras. En particular, el uso de planos de tierra puede proteger la antena de señales no deseadas antes de que lleguen al receptor, mejorando así el nivel de la señal recibida [20].

Dilución de la precisión: La distribución geométrica de los satélites, tal como se presentan al receptor, afecta la precisión de la posición y de la solución en el tiempo. Los receptores usarén las señales de los satélites para minimizar la dilución de la precisión [21].

Error de estimación: Dado que en todos los casos mencionados, se requiere la compensación del error basada ya sea en modelamiento, operaciones entre medidas o técnicas de diferenciación. Cada factor contribuye en diferente medida. La siguiente tabla muestra la magnitud de cada uno.

Tabla 5 Estimación de los errores en un sistema GNSS

Fuente	Rango de error
Relojes satelitales	$\tilde{2}$ m
Errores de órbita	$\tilde{2}$,5 m
Desplazamientos ionosféricos	$\tilde{10}$ m
Desplazamientos troposféricos	$\tilde{0}$,5 m
Ruido en el receptor	$\tilde{0}$,3 m
Multicamino	$\tilde{1}$ m

Esta información muestra que la ionosfera es la fuente más grande de imprecisión. Además, se ha establecido que los efectos de la ionosfera son mayores en latitudes bajas

[22], y cualquier modelo o técnica funciona bien en esta región, lo cual es de interés particular si se tiene en cuenta la posición geográfica de nuestro país. Por estas razones, el enfoque central de este trabajo será disponibilizar una herramienta no privativa que permita probar modelos de la ionosfera que permitan lidiar con los errores de los sistemas GNSS.

5.1.2. Aplicaciones en aviación civil. Los sistemas de navegación actuales (GPS, GLONASS) no fueron diseñados con la capacidad de monitorización en tiempo real de la integridad requerida para satisfacer las necesidades de seguridad de la navegación en la aviación civil. Errores como los retardos ionosféricos deben ser corregidos en tiempo real por una aproximación de precisión donde hay poca o ninguna visibilidad. Para sobreponerse a esta limitación, diferentes sistemas de aumentación fueron desarrollados para complementar GNSS.

La aumentación de un sistemas global de navegación por satélite (GNSS) es un método que permite mejorar los atributos del sistema de navegación, tales como precisión, confiabilidad y disponibilidad, mediante la integración de información externa en el proceso de cálculo de la posición.

Existen diferentes sistemas en marcha, generalmente nombrados o descritos basados en como el sensor GNSS recibe la información externa. Los sistemas de aumentación más comunes son:

- Sistema de aumentación basado en tierra (GBAS).
- Sistema de aumentación basado en satélites (SBAS).
- Sistema de aumentación basado en aeronaves (ABAS).

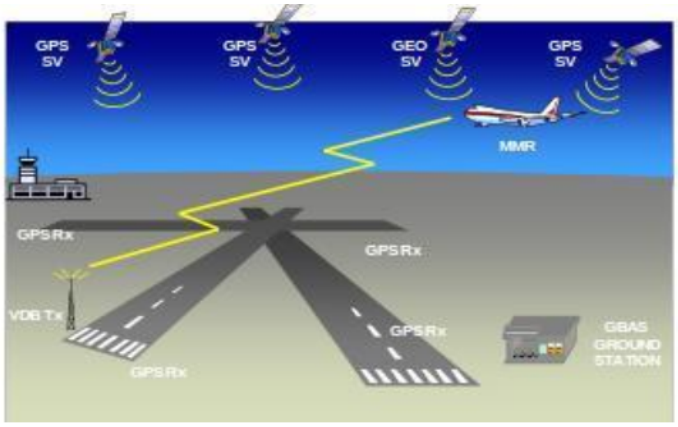
5.1.1. Sistemas de Aumentación

GBAS: Ground Based Augmentation System (Sistema de Aumentación Basado en Tierra) es un sistema crítico en la seguridad en la aviación civil que apoya la aumentación local de las constelaciones GNSS primarias proveyendo niveles mejorados de servicio que soporta todas las fases de aproximación, despegue, partida y operaciones superficiales.

GBAS consiste en tres subsistemas primarios: El subsistema satélite (incluye las constelaciones GNSS, la transmisión de la señal telemétrica y los mensajes de navegación), el subsistema de tierra (incluye todo el equipo requerido instalado en el aeropuerto, básicamente la estación GBAS en tierra que provee a las aeronaves de aproximaciones al camino de los datos y correcciones para cada satélite visible, y los receptores

de referencias) y el subsistema de aeronaves (incluye el equipo instalado a bordo de las aeronaves, que recibe las señales GNSS y los mensajes GBAS transmitidos por la estación en tierra y es capaz de aplicar correcciones GBAS sobre la posición GNSS para determinar su posición relativa al camino de aproximación con más precisión).

Figura 3 Visión general de un sistema GBAS



La infraestructura en tierra para GBAS incluye dos o más (hasta cuatro) receptores GNSS que recolectan pseudorangos (distancia entre un satélite y un receptor de navegación satelital) para cada uno de los satélites GNSS primarios a la vista y computa y difunde correcciones diferenciales e información relacionada con la integridad a ellos basado en su propia posición sondeada. Estas correcciones diferenciales son transmitidas desde el sistema en tierra a las aeronaves a través de difusión de datos de muy alta frecuencia. La difusión de información incluye correcciones a pseudorangos, parémetros de integridad y varios datos localmente relevantes tales como el segmento de aproximación final (FAS). Las aeronaves en el área de cobertura de la estación en tierra pueden usar las correcciones difundidas para computar sus propias medidas. Entonces, esta posición corregida es utilizada para generar señales guía de navegación.

SBAS: Satellite Based Augmentation System (Sistema de Aumentación Basado en Satélites) es un sistema crítico en la seguridad en la aviación civil que apoya la aumentación en áreas amplias o regional a través del uso de mensajes de difusión satelitales adicionales. Tales sistemas comúnmente están compuestos por varias estaciones en tierra, ubicada en puntos determinados con precisión. Las estaciones en tierra toman medidas de uno o más satélites GNSS, de señales satelitales u otros factores ambientales que podrían impactar en la señal recibida por los usuarios. Utilizando estas medidas, se crean mensajes informativos que son enviados a uno o más satélites para ser difundidos a los usuarios finales.

Los sistemas SBAS difunden muchos mensajes, proveyendo información acerca de correcciones, integridad o información sobre el estado del sistema.

ABAS: El Aircraft Based Augmentation System (Sistema de Aumentación Basado en Aeronaves) puede proveer información GNSS conforme sea necesaria por medios suplementarios de navegación. Un ABAS es básicamente un sistema que aumenta o integra la información obtenida de otros elementos GNSS con información disponible a bordo en las aeronaves.

5.2. SIMULADOR GSIM

El simulador de constelaciones GSIM de PILDO genera las posiciones de los satélites y las medidas tomadas por receptores GNSS en una ubicación específica, bajo un escenario (datos de entrada) definido por el instante de tiempo (día, mes, año y época del año), los parámetros de las órbitas de los satélites y los errores incluídos. Las características de GSIM son:

- Múltiples constelaciones: Simula los satélites GPS y GALILEO.
- Multifrecuencia: Simula diferentes frecuencias de emisión de GPS y GALILEO.
- Tasa de generación de mediciones configurable.
- Formatos de Entrada/Salida acorde a los estándares más comunes de la industria.

Los efectos simulados más significativos son:

- Errores de ruido troposférico, ionosférico, multicamino, relativista y térmico.
- Modelos realistas de las órbitas y los errores de reloj y tasas de actualización de información satelital.
- Efectos de bloqueo del terreno: Definidos por el usuario o de archivos de terrenos reales.
- Modelo de sincronización de reloj del usuario.

6. METODOLOGÍA

6.1. METODOLOGÍAS ÁGILES

En el presente proyecto se usaron las metodologías ágiles, que surgen a principios de los 90 ante ciertos problemas de tiempo, costo y calidad en el desarrollo y creación de software.

El desarrollo de software basado en metodologías ágiles es un marco de trabajo conceptual de la ingeniería del software que promueve iteraciones en el desarrollo a lo largo de todo el ciclo de vida del proyecto; es una metodología de tipo creativa que adopta un carácter flexible como respuesta a posibles cambios que se presenten en las diferentes etapas del proyecto.

El software desarrollado en una unidad de tiempo es llamado una iteración, la cual debe durar de una a cuatro semanas. Cada iteración del ciclo de vida incluye: planificación, análisis de requerimientos, diseño, codificación, revisión y documentación. Una iteración no tendrá demasiada funcionalidad para justificar el lanzamiento del producto al mercado, pero la meta es tener un demo (sin errores) al final de cada una de estas [25].

Dentro de las diferentes metodologías ágiles, para la presente propuesta se optó por utilizar la FDD (desarrollo basado en funciones) ya que se ajusta perfectamente a la naturaleza y enfoque del proyecto ya bosquejados anteriormente en los objetivos en donde lo que se busca principalmente es el porting de funciones ya definidas.

6.2. METODOLOGÍA FDD (DESARROLLO BASADO EN FUNCIONES)

Es un proceso ágil diseñado por Peter Coad, Erick Lefebvre y Jeff DeLuca. Se basa en un proceso iterativo con iteraciones cortas que producen un software funcional que se puede ejecutar y monitorizar. Estas iteraciones se deciden en base a funcionalidades, que son pequeñas partes del software con un significado propio de acuerdo al análisis hecho previamente [26]; entre sus ventajas se destacan:

- Cada componente del producto final ha sido probado y satisface los requerimientos.
- Rápida respuesta a cambios de requisitos a lo largo del desarrollo.

- Entrega continua y en plazos cortos de software funcional.
- Minimiza los costos frente a cambios.
- Importancia de la simplicidad, al eliminar el trabajo innecesario.
- Atención continua a la excelencia técnica y al buen diseño [27].

6.3. SISTEMA DE GESTÍON DE VERSIONES DE CÓDIGO

Antes de iniciar con la migración de las funciones se pensó en buscar la manera más adecuada de controlar y ordenar el desarrollo continuo de las versiones del simulador que facilitara al mismo tiempo el trabajo colaborativo y que cumpliera con el perfil planteado por la metodología a seguir. En consecuencia con lo anterior y por sugerencia de PILDO Labs se optó por utilizar subversion, una herramienta de código abierto que cumplía a totalidad con las preocupaciones y exigencias.

Subversion proporciona un repositorio o almacén de datos y versiones controlado en el que se crea una estructura jerárquica similar a la del simulador original y desde allí monitorear el desarrollo continuo de presente proyecto.

Para garantizar la seguridad de transferencia de datos desde y hacia svn, el ingreso al mismo se hacía a través de una VPN (Red Virtual Privada) logrando una conexión segura a la intranet de Pildo Labs en Barcelona, quienes generaron certificados y credenciales especiales para el libre desarrollo de este proyecto.

Lo primero que se hizo fue obtener la primera revisión del repositorio o lo que es lo mismo hacer una copia local, de allí se obtuvo el simulador a portar el cual ya se encontraba en el repositorio gracias al director del presente proyecto.

Figura 4 Autenticación pildo

Luego de obtener la revisión respectiva del simulador y habiendo iniciado con la Fase I y II referentes a portar las primeras funciones fue hora de guardar el progreso y para ello se hizo el respectivo commit, el cual inicia añadiendo los elementos a guardar escribiendo la siguiente línea en la consola svn add ubicación _del_ archivo/nombre_archivo

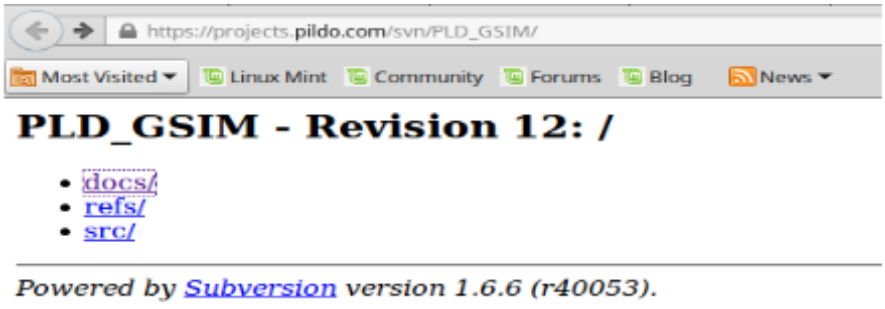
Figura 5 Revisión SVN

```
A    PLD_GSIM/src/python/core/support/writeSimErrorsFile.py
A    PLD_GSIM/src/python/core/support/createRinNavV3Msg.py
A    PLD_GSIM/src/python/core/support/writeRinexObsFile.py
A    PLD_GSIM/src/python/core/support/xyz2enu.py
A    PLD_GSIM/src/python/core/support/writeRinexNavFile.py
A    PLD_GSIM/src/python/core/support/readYuma.py
A    PLD_GSIM/docs
A    PLD_GSIM/docs/TODO.txt
Checked out revision 12.
```

seguida de svn commit -m çomentarioz así se genera la versión con su respectivo indicador y fecha de creación.

Las revisiones del estado del repositorio se hicieron con la ayuda de un navegador web.

Figura 6 Revisión svn con navegador web



6.4. RECURSOS SOFTWARE

NOMBRE	VERSIÓN	DESCRIPCIÓN
	8.1.0.604 (R2013a)	Entorno de computación iterativo para cálculo numérico y tratamiento de datos, posee paquetes de herramientas que ayudan a mejorar su rendimiento. Funciona con Licencia Privada Cabe resaltar el uso del profile, y de su generador de archivos para los bancos de prueba.
	2.9 Anaconda 2.1.0	Lenguaje de alto nivel interpretado y multipropósito. Es acreedor a un conjunto de módulos muy útiles al momento de realizar cálculos científicos como lo son numpy y scipy al igual que facilidad de optimización y paralelización de segmentos de código, se caracteriza por ser de licencia libre.

7. VIABILIDAD

La programación científica consiste, en su mayor parte, en cálculos numéricos intensos, un gasto de CPU en estado puro. Python es un lenguaje interpretado al igual que Matlab y posee también una gran variedad de herramientas que complementan su uso normal, pero cuando de dinero se habla python posee una licencia de software libre lo que lo hace aún más atractivo.

El paquete de herramientas NumPy y el modulo Math de python son muy similares a Matlab (tratamiento de matrices, descubrimiento automático de tipos de variables, entre otras cosas), lo cual es de gran ayuda al momento de la migración; además, posee paquetes de librerías para optimización muy completos, donde puede destacarse NumbaPro, ofreciendo la posibilidad a los desarrolladores de crear rápidamente código optimizado que se integra bien con NumPy.

Python ofrece mayor facilidad de incluir el código en un servidor o en la arquitectura software de un usuario cualquiera, ya que a diferencia de Matlab no está ligado a la generación de un runtime que provoque dependencia de versiones.

El trabajo de investigación actual que lidera PILDO Labs relacionado con el modelamiento de la ionosfera a partir primero de datos simulados y luego con datos reales, se está realizando en python, por lo que la migración de GSIM a python genera un valor agregado muy grande a dicho proyecto, debido a que con los datos obtenidos por parte del simulador se generan conjuntos de datos que contienen información de la geometría de la disposición del satélite y el receptor con los cuales se calcula, por ejemplo, el punto de perforación ionosférica para medir la distorsión que la ionosfera produce en la señal y por tanto modelar el error de posicionamiento.

8. DESARROLLO DEL PROYECTO

8.1. FASE I: ANÁLISIS DEL SIMULADOR

Se comenzó recorriendo el sistema de carpetas del simulador con el objetivo de adquirir cierta familiaridad con la estructura en la que se encuentra dispuesto; se dedicaron un par de días a ojear las funciones y leer su documentación, buscando entender la forma en que el sistema está cohesionado. Sin embargo, era claro que una exploración pasiva de GSIM no brindaría una visión concreta de su comportamiento; a la luz de este propósito, fue evidente la necesidad de ejecutar el programa con datos reales

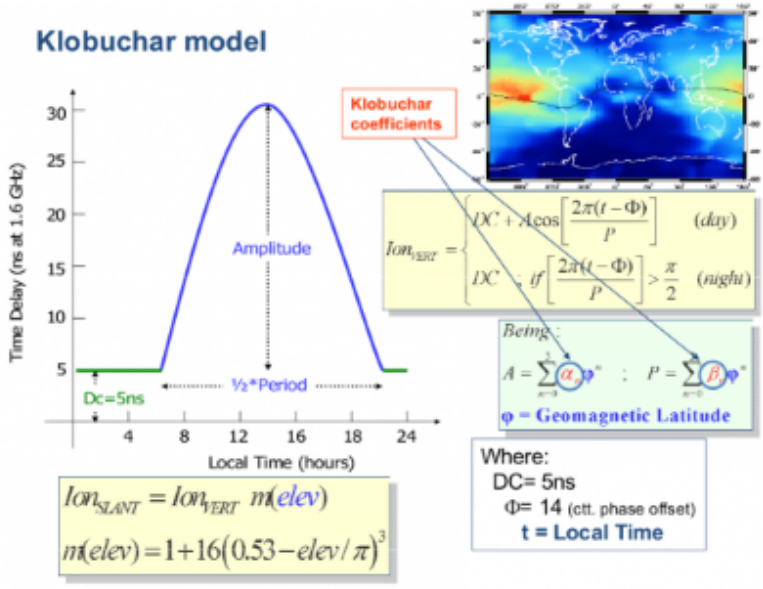
Se solicitó un caso de uso a la investigadora y estudiante de doctorado Susana Sánchez, quien se apoya de este simulador para validar los modelos que ha desarrollado para su disertación; ella puso a disposición un archivo de configuración básico con una característica particular importante: Los resultados generados son determinísticos y es sencillo contrastarlos. Por la forma en que los parámetros han sido definidos, el simulador debe leer los archivos en formato YUMA y retornar tres ficheros RINEX v3; las banderas más relevantes, encargadas del control de los errores y las constelaciones, se muestran en la figura [7]. El programa funciona además en modo dinámico, lo que se traduce en el requerimiento de un archivo de trayectorias.

Figura 7 Banderas caso de uso I

```
% GPS simulation flag
gnssParam.gSimFlag = [4 3 3 3];




% GALILEO simulation flag
gnssParam.eSimFlag = [3 3 3 3];%[0 0 0 0];%



% GNSS Errors Scaling Factor flags
gnssParam.esf = [0 0 0 0 3 0 0 0 0];
%  gnssParam.esf = [0 0 0 0 0 0 0 0 0]
```

Otro parámetro significativo es la bandera del error de ajuste, establecida para generar error ionosférico. Se utiliza el modelo klobuchar de refracción ionosférica el cual da como resultado una aproximación de la latitud, longitud, ángulo de elevación y acimut del satélite observado en forma de coeficientes los cuales son enviados en el mensaje de navegación del mismo [20].

Figura 8 Modelo Klobuchar



Siendo el objetivo principal de esta fase extraer la estructura del programa, se utilizó el perfilador que viene incorporado en Matlab; aunque éste únicamente guarda registro de las funciones que son llamadas durante la ejecución (es decir, la relación de los scripts no utilizados no aparece en el reporte de resultados), para la primera parte del proceso de porte fue más que suficiente.

El perfilador arroja como resultado un archivo interactivo el cual se encuentra organizado por tres características diferentes: Cantidad de llamadas, tiempo individual y tiempo total; en la tabla [8.1] se listan las funciones cuyos valores sobresalen en cada una de estas categorías, para complementar la información suministrada puede remitirse al [Anexo A].

Tabla 6 Perfilador,funciones críticas

Función	Tiempo Por Llamada [s]	Tiempo Total	Número de Llamadas
xyz2llh	24,375	24,375	150097
xyz2enu	25,956	46,399	136728
computeSvPosRin	31,925	31,925	121248
genSatPosition	26,803	96,981	3789
genRangeErr	3,744	26,928	184
genSvSnr	0,642	1,159	138

Como complemento también se puede obtener de cada función una descripción más detallada donde deja en evidencia las líneas de código en las cuales gasta más tiempo de cómputo, si posee funciones padre por las cuales es llamada y de la misma manera sus funciones hijas, esto se puede ver en la figura [9].

Figura 9 Resultado Perfilador de Matlab

Function Name	Function Type	Calls
genSatPosition	function	121248
main	function	528
genSvSnr	function	1584
mopstropgen	function	6684
ionogen	function	6684

Lines where the most time was spent

Line Number	Code	Calls	Total Time	% Time	Time Plot
68	orgllh = xyz2llh(orgxyz);	136728	24.093 s	51.9%	
65	if size(tmpxyz) ~= size(tmporg...	136728	5.028 s	10.8%	
75	R = [-sinlam coslam ...	136728	3.838 s	8.3%	
78	enu = R*difxyz;	136728	3.337 s	7.2%	
66	difxyz = tmpxyz - tmporg;	136728	1.426 s	3.1%	
All other lines			8.678 s	18.7%	
Totals			46.399 s	100%	

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
xyz2llh	function	136728	20.803 s	44.8%	
Self time (built-ins, overhead, etc.)			25.596 s	55.2%	
Totals			46.399 s	100%	

A partir de la interpretación de los resultados se elaboró un esquema con la estructura de dependencias básica del programa , las funciones se clasificaron en esenciales y de alto nivel como se muestra en la tabla [8.2].

Tabla 7 Clasificación de Funciones GSIM

Funciones	Descripción	Categoría
allan_plot_to_Q clock_states_F_and_Q compute_mask_angle computeSvPosRin convertObst2D createRinNavV3Msg gpst2utc ionocorr llh2xyz loadradpat readyuma resize_svid_svmat xyz2enu xyz2llh	Funciones hoja en el árbol de dependencias y funciones en los niveles 2 y 3 del mismo.	Esenciales
computeSvTimeTrans configfile genRangeErr genSatPosition genSvSnr main update_user_clock_errors writeRinexNavFile writeRinexObsFile writeSimErrorsFile	Funciones en los niveles 0 y 1 de la jerarquía que durante su ejecución llaman otras funciones (dependientes).	Alto nivel
Otras	Funciones no contempladas en el primer caso de uso.	Alto nivel

La migración del simulador se planeó para llevarse a cabo en dos etapas, una primera en la que se portarían las funciones clasificadas como esenciales y una segunda donde se trabajaría con las que se categorizaron como de alto nivel. Para verificar el correcto funcionamiento de cada uno de los scripts que se generaran, se almacenaron parejas de entradas - salidas reales provenientes de la ejecución del primer caso de uso. Incialmente se consideró escribir las variables a un archivo CSV;sin embargo, resultó ser una alternativa excesivamente lenta y que le agregaba cierta complejidad al manejo de las variables (necesidad de definir algún identificador para diferenciarlas entre sí, por

ejemplo). Luego de explorar otras alternativas, ninguna menos convincente que el ya descartado formato, se optó simplemente por almacenar las variables en ficheros .mat.

Tabla 8 Características de las muestras.

Función	Número de muestras	Tamaño en disco
xyz2llh	59446	235 M
llh2xyz	1	992 K
xyz2enu	76138	302 M
gpst2utc	62	260 K
convertObst2D	1	1,2 M
allan_plot_to_Q	25	1,1 M
resize_svid_svmat	185	1,7 M
compute_mask_angle	528	3,1 M
computeSvPosRin	74616	297 M
clock_states_F_and_Q	20	100 K
ionocorr	2217	9,8 M
createRinNav3Msg	57	1,2 M
loadratpat	2	992 K

Función	Número de muestras	Tamaño en disco
genSatPosition	3721	16 M
computeSvTimeTrans	139	1,6 M
genSvSnr	139	1,6 M
update_user_clock_errors	23	1.1 M
genRangeErr	185	2.9 M
writeRinexNavFile	1	984 K
writeRinexObsFile	1	1.2 M
writeSimErrorsFile	1	1.2 M
configfile	1	992 K

Considerando que la mayoría de funciones son llamadas más de una vez, y para evitar confusiones al momento de las pruebas, era necesario guardar cada pareja en un archivo distinto. Así, con cada llamada a un script debía generarse un nuevo fichero, lo cual requería encontrar una forma de variar el nombre en cada ocasión. En un principio se intentó hacerlo mediante una numeración ordenada, pero el tiempo adicional que esto implicaba decantó la situación en favor de generar nombres aleatorios. Para facilitar el uso posterior de los archivos, se decidió que todos compartieran en sus nombres una primera parte genérica. Un ejemplo del código puede verse en la figura [10].

Figura 10 Código para crear archivos prueba

```
name='xyz2llh';
name=strcat(name,num2str(rand),'.mat');
save(name,'xyz');
save(name,'llh','-append');
```

Para las pruebas se diseñó un plantilla adaptable según las características de cada función. Básicamente, se cargan uno a uno los archivos .mat de la carpeta actual, se invoca la función que se quiere probar y luego se comparan las salidas de ésta con las almacenadas en el archivo (los valores numéricos con tolerancias, para los demás se verifica la igualdad). Si desea consultar la plantilla, remítase al [anexo C].Es importante aclarar que algunas funciones generan por salidas archivos de texto, para la comprobación de estas se utilizó un enfoque completamente distinto que consistió en verificar la estructura y contenido de los archivos mediante el comando diff desde la terminal.

8.2. Fase II: MIGRACIÓN DE LAS FUNCIONES ESENCIALES

De acuerdo a lo planeado se comenzó el proceso de migración con las funciones esenciales, las cuales a pesar de su menor complejidad con respecto a las funciones consideradas de alto nivel conforman una parte fundamental en el funcionamiento del simulador. Aunque los entregables de esta fase por sí solos no tienen demasiada aplicación práctica, su buen desempeño se constituyó en una base sólida para dar inicio a la siguiente fase.

Siendo la precisión uno de los factores clave en este proyecto, se tomó la decisión de comenzar portando las funciones más críticas en lo que a esto respecta. De los datos obtenidos en la fase anterior se concluyó que los scripts encargados de la conversión de unidades son los que manejan mayor cantidad de cifras decimales; además, los resultados que estos arrojan son luego procesados por otras funciones, haciendo que cualquier error de precisión se amplifique varias veces. Considerando que dentro de esta categoría caen aún varias partes del programa, se utilizó como segundo criterio de prioridad el menor número de dependencias y como tercero, la complejidad de las instrucciones.

Como dato curioso acerca de las primeras funciones portadas, todas ellas se encuentran en la carpeta support dentro de la estructura del simulador. Una segunda subcategoría se definió tras tomar la decisión de continuar con la migración de las funciones que toman las salidas de las previamente portadas y a partir de ellas realizan cálculos, en este caso el criterio de desempate fue una combinación entre la percepción de complejidad por parte de los desarrolladores y un cierto grado de aleatoriedad. Por último se

tradujeron las funciones restantes, en su mayoría encargadas de la lectura de archivos. El orden general en que se portaron puede verse en la tabla [8.4].

Tabla 9 Orden de Migración fase II

Orden	Función	Categoría	Líneas de Código
1	xyz2llh	Conversión	105
2	llh2xyz	Conversión	51
3	xyz2enu	Conversión	87
4	gpst2utc	Conversión	197
5	convertObst2D	Conversión	37
6	allan_plot_to_Q	Cómputo	16
7	resize_svid_svmat	Cómputo	17
8	compute_mask_angle	Cómputo	10
9	computeSvPosRin	Cómputo	93
10	clock_states_F_and_Q	Cómputo	14
11	ionocorr	Cómputo	68
12	createRinNav3Msg	Escritura	194
13	loadradpat	Lectura	51
14	readyuma	Lectura	220

Migración de las funciones de conversión. El principal problema encontrado en esta parte de la segunda fase fue, como ya se dijo, lidiar con lo concerniente a la precisión. Entre los inconvenientes de menor relevancia se cuentan algunas consideraciones de sintaxis y cuestiones en la estructura de los datos, los cuales se explicarán más en detalle posteriormente. A continuación se presentan en orden cronológico de migración, y con cierto detalle, el funcionamiento de cada una de las funciones de conversión de unidades, describiendo el proceso y las consideraciones tenidas en cuenta durante su traducción.

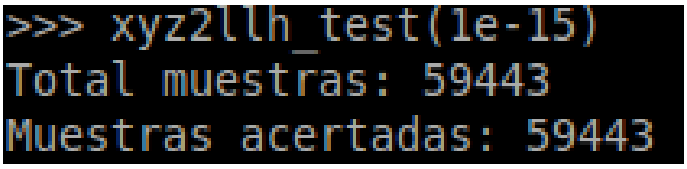
xyz2llh: Del nombre de esta función resulta bastante intuitivo inferir su papel dentro del simulador, aún para quien no esté familiarizado con el tema; se encarga de la conversión entre dos sistemas de coordenadas GNSS, puntualmente, de coordenadas cartesianas ECEF (xyz) a coordenadas latitud - longitud - altura (llh). Naturalmente, tanto la entrada como la salida son tripletas de coordenadas, aunque por el momento la función únicamente soporta una a la vez (dicho de otro modo, solo procesa vectores de tres elementos y no matrices n x 3).

Las operaciones internas se reducen a matemáticas básicas entre variables escalares. Durante la traducción de esta función fue importante prestar atención a los resultados

intermedios, pues en varias ocasiones se efectuaban cocientes entre enteros y los resultados no eran los esperados.

Con la intención de asegurar que la salida fuera la correcta, los valores generados a partir de las entradas que se tenían almacenadas (77 063 muestras) se contrastaron con las correspondientes salidas. Se puede garantizar que la totalidad de estos resultados coinciden hasta una tolerancia de 1e-15.

Figura 11 Resultado prueba xyz2llh

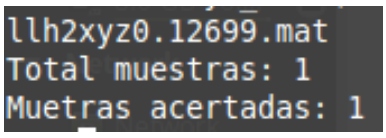


llh2xyz: Esta función desempeña una labor complementaria a la anterior, convirtiendo coordenadas llh a xyz. Es utilizada por el simulador únicamente cuando se especifican coordenadas llh como parámetro de posición estática del usuario en el archivo de configuración, para generar las correspondientes coordenadas cartesianas ECEF.

Una vez más las variables son netamente escalares y los cálculos, operaciones matemáticas sobre ellas con predominio de funciones trigonométricas. Dado el menor número de cocientes y que estos involucran siempre variables flotantes, el truncamiento por operaciones entre enteros no fue un problema en esta ocasión.

Partiendo de que solo se tiene una muestra para esta función, sería arrebatado sacar conclusiones respecto a su precisión a partir de la prueba; sin embargo, del uso dado por PILDO Labs para medir la distorsión que produce la ionosfera sobre la señal y así modelar el error de posición del receptor, se encontró que los resultados son tan precisos como en el caso anterior.

Figura 12 Resultado prueba llh2xyz



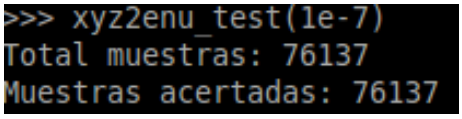
xyz2enu: Se encarga de convertir coordenadas cartesianas ECEF (xyz) a rectangulares tangentes a nivel local (ENU). Además de recibir la terna a transformar, precisa de un vector auxiliar de tres elementos que designa la posición de referencia local. Depende

de xyz2llh para la conversión de formato de este último.

Dentro de las operaciones que ocurren en este script, además de aritmética de escalares, es posible encontrar un producto matricial. Salvo para la creación de matrices donde se utilizó la función array de numpy, todas las operaciones se hicieron con python nativo (incluido el uso de algunos módulos de la librería math).

Se comprobó precisión en las salidas hasta una tolerancia de 1e-7, ésta fue menor que en los casos previos a causa de la propagación del error presente en xyz2llh.

Figura 13 Resultado prueba xyz2enu



gpst2utc: A diferencia de las anteriores, ésta no realiza conversión de coordenadas sino entre formatos de tiempo. La entrada esperada incluye un par de vectores n x 1 indicando valores válidos de semanas (1 - 3640) y segundos (0 - 604799) de tiempo GPS y otra variable con la cantidad de segundos intercalares. El tiempo GPS indicado es convertido a tiempo universal coordinado (UTC) y devuelto en términos de años, meses, días, horas, minutos, segundos y días del año, como vectores de la misma dimensión (n x 1).

Durante el proceso de traducción fue importante prestar atención a varios detalles. Primero, asegurar que las matrices estuvieran propiamente alineadas a fin de evitar la difusión de elementos y en consecuencia, resultados erróneos. Segundo, estudiar las instrucciones find presentes y encontrar la forma adecuada de sustituirlas, lo que finalmente se hizo con la función find de numpy, utilizando el índice cero. Por último, asignar a cada variable el tipo de dato apropiado antes de devolverla (algunas requerían ser enteras, pero eran generadas como flotantes).

Contrastando las salidas generadas con las esperadas hay coincidencia absoluta, pues en la fase de cálculo todas las respuestas son truncadas y habitualmente se trabaja con divisiones enteras. En caso de entradas no válidas, al igual que el simulador original, se producen inf en las correspondientes posiciones.

convertObst2D: La última de las funciones en esta subcategoría, reduce una dimensión a una matriz de observables (3D) y devuelve como resultado la matriz de nombres extendidos equivalente (2D). Además de la matriz a transformar, recibe un par de

Figura 14 Resultado prueba gpst2utc

```
>>> gpst2utc_test(1e-900)
Total muestras: 62
Muestras acertadas: 62
```

parámetros auxiliares con etiquetas e índices.

Aunque el número de líneas era reducido, esta función introdujo dos desafíos importantes: La búsqueda de una alternativa para los structs de Matlab y el uso de la función eval. Para resolver el primero de los inconvenientes se procuraron algunas opciones y se leyó la documentación de cada una de ellas; los candidatos principales eran diccionarios, arreglos estructurados y named tuples. Los diccionarios, aunque intuitivos y de sintaxis muy similar a los structs, fueron descartados de forma casi inmediata por su enorme consumo de memoria en comparación a las otras dos estructuras. Entre las restantes alternativas se optó por los arreglos estructurados dada la mayor versatilidad que ofrecen tanto en su manejo como en su definición, además de permitir compatibilidad directa (sin requerir procesamiento adicional) con archivos .mat.

Para crear el arreglo estructurado de manera precisa se hizo necesario agregar unos cuantos arreglos temporales, lo que en términos prácticos se tradujo en la duplicación de las líneas de código del script. La razón para lo anterior es que python no permite la definición dinámica de arreglos como sí lo hace Matlab y no había forma de declarar todo el arreglo en una línea única pues las dimensiones de cada campo no eran conocidas con anticipación.

En cuanto a eval, el inconveniente no fue encontrar con qué reemplazarla, sino adecuar correctamente la sintaxis. El eval ejecutaba una asignación; después de observar detenidamente la forma en que se comportaba se decidió extraer la variable asignada, es decir, mover la instrucción (eval) a la derecha del igual.

Consecuencia de la disponibilidad de un único caso de uso, y por ende de una única matriz de observables, la comparación se hizo con una sola muestra, obteniendo coincidencia perfecta. No se consideró necesario realizar pruebas de precisión adicionales por el hecho de que esta conversión es de naturaleza estructural y no implica cálculos.

Figura 15 Resultado prueba convertObst2D

```
>>> convertObst2D_test(1e-900)
Total muestras: 1
Muestras acertadas: 1
```

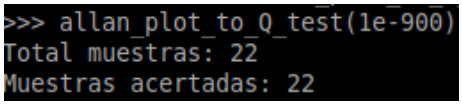
Migración de las funciones de cómputo. La precisión siguió siendo un problema, aunque no tan marcado como en las anteriores funciones. El principal reto afrontado en esta parte del proceso fue la correcta creación y manipulación de los arreglos y demás estructuras en los que se almacenan los datos y con los que se efectúan los cálculos. La cronología de la migración se presenta a continuación; una vez más se acompaña de una breve descripción del funcionamiento de cada script, los problemas que supuso su traducción y la solución que a estos se dio.

allan_plot_to_Q: Esta función genera la matriz de reloj Q partiendo de los parámetros h definidos en el archivo de configuración y el tiempo entre épocas consecutivas. La salida es una matriz cuadrada de cuatro elementos, con la particularidad de que los valores en su diagonal secundaria son iguales.

Como ya había ocurrido con xyz2llh el truncamiento que hace python a las divisiones entre enteros fue un inconveniente, al igual que la necesidad de devolver la matriz de reloj Q como un arreglo de flotantes. La solución a ambos problemas fue forzar la conversión de los resultados parciales al tipo requerido.

Aunque había algunas operaciones matemáticas de por medio, las salidas fueron idénticas a las teóricas; lo anterior se explica porque tanto los operandos como los resultados tenían pocas cifras decimales.

Figura 16 Resultado prueba allan_plot_to_Q

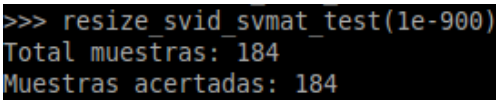


resize_svid_svmat: Este fragmento del simulador cambia el tamaño de los vectores de identificación y posición del satélite. Para ello recibe adicionalmente como entrada una lista de índices, los cuales se verifican para luego utilizarse en el filtrado de los elementos de los susodichos vectores. Las salidas son, por supuesto, los vectores filtrados.

En este script aparece nuevamente find, el cual se trató del mismo modo que en gpst2utc. Al momento de aplicar los índices fue necesario restar uno por las diferencias en indexación que Matlab y python tienen. No involucrando aritmética alguna, la precisión no fue un inconveniente aquí, siendo la verificación de la prueba netamente orientada a la estructura.

compute_mask_angle: El código en este script computa el ángulo de la máscara de un satélite respecto al ángulo de la máscara de visibilidad y el acimut en la posición del usuario. Además de recibir los dos parámetros anteriores en una matriz, precisa de un

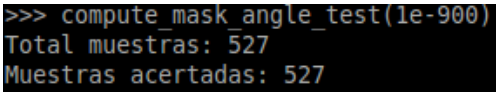
Figura 17 Resultados prueba resize_svid_svmat



vector 1 x 3 con las coordenadas en formato enu de la posición del satélite (utilizando como origen al usuario).

Todos los datos se trabajan en grados y se procesan mediante una función de interpolación en una dimensión; para portar esta sección se utilizó el módulo interp de numpy, el resto del código se tradujo a python nativo. Los resultados tienen una precisión alta.

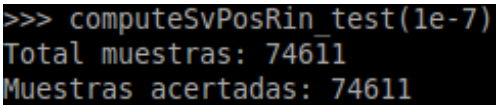
Figura 18 Resultado prueba compute_mask_angle



computeSvPosRin: El papel de esta función es calcular la posición del vehículo espacial (SV) utilizando los parámetros de navegación típicos. Toma 19 constantes como parámetros de entrada y devuelve un vector de coordenadas de tamaño 1 x 3. Luego de verificar la consistencia de los argumentos, resuelve iterativamente la ecuación de Kepler y aplica correcciones a algunos parámetros. Finalmente, calcula la matriz de rotación, y con ella, las coordenadas del SV.

Dada la gran cantidad de cálculos y el uso de funciones trigonométricas, la precisión desempeñó un papel fundamental en la migración de esta función. Teniendo en cuenta todas las consideraciones tenidas con funciones previas se logró garantizar una precisión mínima de 1e-7 (similar a la de xyz2enu).

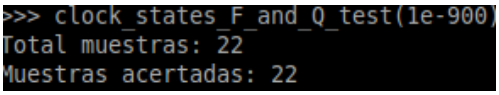
Figura 19 Resultado prueba computeSvPosRin



clock_states_F_and_Q: Sus resultados se utilizan para computar el error de reloj de usuario; específicamente, genera las matrices F y Q. Sus argumentos son los mismos que la función allan_plot_to_Q (a la cual llama), aunque a diferencia de ésta los parámetros h son pasados como una sola variable y no por separado.

La definición de la matriz F no da lugar a error pues se reduce a la creación de una matriz identidad y la sustitución de algunos de sus valores por uno de los argumentos de entrada. Si se considera además que la matriz Q es creada invocando a la función allan_plot_to_Q, cuyo error absoluto es nulo, era de esperarse que clock_states_F_and_Q tampoco tenga problemas de precisión.

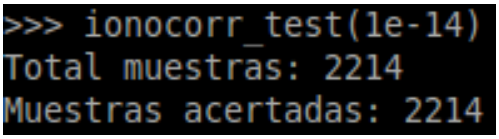
Figura 20 Resultado prueba clock_states_F_and_Q



ionocorr: La última de las funciones de cómputo, determina la corrección del retardo ionosférico en metros dados un tiempo en el cual computarlo, las posiciones del satélite y del usuario en coordenadas ECEF (como vectores 1 x 3) y los parámetros alfa y beta del modelo de difusión ionosférico (como un struct en Matlab y un arreglo estructurado en python).

La traducción del código fue bastante directa, salvo algunas diferencias de sintaxis entre los lenguajes. Puesto que llama tanto a xyz2enu como a xyz2llh, la precisión está condicionada hasta cierto punto por estas funciones. Se logró conseguir una precisión de 1e-14 de acuerdo a la prueba hecha con 2214 muestras.

Figura 21 Resultado prueba ionocorr



Migración de las funciones restantes Finalmente, antes de dar por concluida la fase II, se tradujeron algunas funciones de lectura de archivos y una función auxiliar de escritura, curiosamente todas dentro de la carpeta support. Aquí no fue importante hablar de precisión sino verificar que las variables se cargarán correctamente o que las constantes se definieran de forma adecuada. Se tuvo la necesidad de explorar algunas técnicas de lectura y escritura de archivos y decidir cuál era más conveniente para las necesidades del proyecto. El uso de cada script se presenta a continuación, junto a algunos pormenores del proceso de porte.

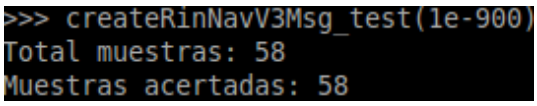
createRinNavV3Msg: Esta función sirve como auxiliar a la función writeRinexNavFile (portada en la siguiente fase) generando los mensajes RINEX v3 que ésta imprime. La salida es un vector de 1 x n, donde cada elemento es una cadena de caracteres; como entradas se requieren una estructura con las constantes necesarias y tres variables

auxiliares que sirven como índices.

La cuestión más importante al momento de portar esta función fue el formateo adecuado del texto, teniendo que buscar los equivalente apropiados para cada código de formato. También hubo la necesidad de convertir algunas variables de entero o flotante a cadena de caracteres para así evitar errores al momento de aplicar ciertos formatos.

La comparación en esta ocasión se hizo con 58 muestras, verificando que coincidieran línea por línea. Terminada la prueba se corroboró que el formato había sido aplicado correctamente.

Figura 22 Resultado prueba createRinNavV3Msg



loadradpat: Siguiendo la ruta que se le pasa como argumento, intenta cargar el archivo de radiación o devuelve un error de acceso. Si encuentra el fichero, lo itera por líneas y almacena y retorna los factores de ruido vertical y horizontal en decibeles.

Se consideró la aproximación más apropiada abrir el fichero con open para luego iterar el objeto línea por línea, la principal ventaja de este método es que una vez se alcanza la última línea el ciclo se termina evitando errores de lectura. Para guardar la salida se utilizó un arreglo estructurado de dos campos, emulando el struct que devuelve Matlab.

Siendo las variables tomadas directamente de un archivo, no es necesario verificar precisión sino que hayan sido cargadas de forma correcta. Recorriendo las 2 muestras tomadas, se verificó que las salidas de Matlab y python son idénticas.

readYuma: Funciona de forma similar a la anterior, aunque requiere un parámetro adicional (el número de satélites). Empleando la ruta que se le indica trata de abrir el almanaque del satélite (en formato YUMA) para luego recorrerlo e ir almacenando las variables. Al final todos los resultados temporales son puestos en una estructura (struct de Matlab), la cual es retornada al usuario.

La iteración sobre el fichero se hizo del mismo modo que en loadradpat; sin embargo, surgió el inconveniente adicional de tener que lidiar con los espacios en blanco al final de cada línea. Una forma práctica que se encontró para deshacerse de ellos fue utilizar la función replace para eliminar los vacíos de las cadenas de caracteres que los estaban almacenando.

Una vez más la comparación se hizo elemento a elemento, entre datos teóricos (Matlab)

y experimentales (python), confirmando que las variables guardadas concuerdan con lo esperado.

Tras la conclusión de esta fase del proyecto se procedió a hacer entrega de las funciones portadas con su respectivo conjunto de pruebas; además de servir como constancia del trabajo realizado, tres de ellas comenzaron a utilizarse en la generación de conjuntos de datos con coordenadas de posición para la validación de modelos ionosféricos, con resultados satisfactorios.

8.3. FASE III: MIGRACIÓN DE LAS FUNCIONES DE ALTO NIVEL

Habiendo portado las funciones esenciales, se abrió el camino a la migración de las funciones restantes necesarias para la ejecución del simulador. Inicialmente se tradujeron las funciones que se precisaban para soportar el primer caso de uso; sin embargo, para no limitar la funcionalidad se solicitaron casos de uso adicionales sobre los cuales trabajar. En respuesta a la solicitud se recibió una descripción de las funcionalidades esenciales en base a las cuales se elaboró un nuevo caso de uso.

De manera análoga a como se procedió en la fase anterior, se definieron subcategorías que orientaron el orden de migración. Las primeras cuatro se establecieron con la información del primer caso de uso y una quinta se incluyó con los resultados del análisis de los casos adicionales. Se decidió portar primero las funciones que realizan cálculos relacionados a los satélites, cualquiera fuera su naturaleza (posición, elevación, tiempos, etc.), salvo los referentes a errores que se agruparon en una segunda categoría. Un tercer grupo se definió con las funciones de escritura de archivos y un cuarto para incluir a los dos scripts más extensos y a partir de los cuales el simulador corre: el archivo de configuraciones y el script principal.

A continuación se detalla el proceso de migración de las primeras cuatro categorías, la secuencia de migración dentro de cada una de ellas se decidió en base a las dependencias, a la percepción de complejidad por parte de los desarrolladores y en caso de condiciones similares, por azar; el orden general se resume en la tabla [8.5]. Posteriormente se reproduce el caso de uso adicional, se analiza brevemente y se describen las funciones que se agregaron al proceso consecuencia de él.

Migración de las funciones de cómputo (satélites). La principal característica de esta parte del simulador es la presencia de ciclos donde se calculan variables de forma iterativa; así mismo, reaparecen algunas de las cuestiones de sintaxis y varias de las funciones nativas de Matlab con las que se lidió en la fase anterior. Las tres

Tabla 10 Descripción funciones de alto nivel

Orden	Función	Categoría	Líneas de código
1	genSatPosition	Cómputo	155
2	computeSvTimeTrans	Cómputo	99
3	genSvSnr	Cómputo	179
4	update_user_clock_errors	Generación de errores	26
5	genRangeErr	Generación de errores	712
6	writeRinexNavFile	Escritura	282
7	writeRinexObsFile	Escritura	630
8	writeSimErrorsFile	Escritura	477
9	configfile	Extensas	420
10	main	Extensas	1077

funciones que se portaron en esta parte de la tercera fase se presentan a continuación junto a algunas generalidades de cada una.

genSatPosition: Esta función cómputa las posiciones de los n satélites visibles y las devuelve en una matriz n x 3, retorna además tres vectores: Los dos primeros, vectores fila de n y m elementos, respectivamente, conteniendo los identificadores de los satélites visibles y los índices de las efemérides utilizadas; el tercero, un vector columna de tamaño m con las posiciones de los satélites de la constelación. Dentro de las entradas se incluyen un vector renglón con la posición del usuario en coordenadas ECEF, el tiempo de la semana en segundos, un vector m x 1 con los ángulos de las máscaras y un par de estructuras (structs en Matlab) con las constantes necesarias. Acepta además dos parámetros opcionales: Un vector con los índices de las efemérides a usar y una terna de coordenadas de referencia local.

El primer cambio en la sintaxis que se tuvo que hacer fue la forma de manejar los parámetros opcionales, reemplazando las verificaciones internas en el código original por valores por defecto en python. Una segunda cuestión que se cruzó en el camino fue el manejo de los vectores de índices, siendo necesario analizar en qué casos debían ajustarse y en cuáles no. Por último, los problemas de precisión reaparecieron y fue necesario aplicar las medidas habituales para controlarla.

Con 3719 muestras, la prueba fue 100 % exitosa hasta una tolerancia de 1e-7; puede inferirse así que su precisión está condicionada por la llamada que internamente hace a xyz2enu.

computeSvTimeTrans: Las instrucciones en esta función permiten determinar el tiempo en que la señal es transmitida, la posición de los satélites visibles y la eleva-

Figura 23 Resultado prueba genSatPosition

```
>>> genSatPosition_test(1e-7)
Total muestras: 3719
Muestras acertadas: 3719
```

ción de los satélites de la constelación; para su correcta ejecución precisa de la posición del usuario en coordenadas ECEF (vector 1 x 3), una matriz con las posiciones de los n satélites visibles también en ECEF (n x 3), el tiempo de recepción de la señal, un vector con los identificadores de los satélites (1 x n), las efemérides y los ángulos de las máscaras de los m satélites de la constelación (1 x m) y una estructura (struct en Matlab) con las constantes requeridas. Las dimensiones de los vectores de salida son 1 x n, n x 3 y m x 1, respectivamente.

Los retos planteados al momento de su migración no fueron muy diferentes a los de genSatPosition; sin embargo, un manejo más cuidadoso de las estructuras se requirió como consecuencia de su dependencia a esta última. El hecho de que el código internamente minimice el error de las salidas permitió acertar todas las muestras (138) hasta una tolerancia de 1e-9, a pesar de que algunas de las variables internas eran precisas solo a 1e-7.

Figura 24 Resultado prueba computeSvTimeTrans

```
>>> computeSvTimeTrans_test(1e-7)
Total muestras: 138
Muestras acertadas: 138
```

genSvSnr: De la ejecución de este script se obtienen dos salidas, ambas relacionadas a la proporción señal a ruido (un vector con los valores para los n satélites visibles y un escalar con el valor en el cenit). De manera análoga a las funciones anteriores, dos de los parámetros son la posición del usuario en ECEF y las posiciones de los satélites visibles (con las mismas dimensiones que en los casos previos); se requieren además ciertas constantes que son pasadas dentro de estructuras provenientes del fichero de configuración o, en algunos casos, de manera aislada (por ejemplo, la frecuencia de la señal).

Los cálculos y la estructura general son similares a las dos funciones anteriores, aunque a diferencia de ellas genSvSnr tiene muchas más sentencias condicionales; durante el porte fue importante prestar suma atención a la indentación. La precisión también jugó un papel importante, siendo necesario depurar cuidadosamente los resultados parciales.

Aunque también depende de xyzenu, a la cual llama directamente, la precisión fue mayor que en las demás funciones de esta subcategoría por el hecho de que la variable

que de ésta obtiene no interviene directamente en el cálculo de las salidas. La prueba tuvo éxito hasta una tolerancia de 1e-13.

Figura 25 Resultado prueba genSvSnr

```
>>> genSvSnr_test(1e-13)
Total muestras: 138
Muestras acertadas: 138
```

Migración de las funciones de generación de errores. Las funciones en esta categoría comparten únicamente su propósito de generación de errores, las especificidades de cada una son radicalmente distintas y cada una de ellas planteó retos distintos. A continuación se dan más detalles acerca del proceso de migración en esta etapa.

update_user_clock_errors: Esta función actualiza los errores del reloj de usuario en tiempo y frecuencia. Teniendo en cuenta que llama a clock_states_F_and_Q, recibe entre sus argumentos los necesarios para la invocación de esta última; entre sus entradas se cuentan también un par de vectores con los errores de reloj de usuario, los cuales son devueltos como salida con un elemento adicional agregado al final de ellos.

Un factor importante en este script es el uso de la función randn de Matlab, considerando la naturaleza normalizada de los números aleatorios que dicha rutina genera se decidió reemplazarla por normalvariate, parte de módulo random de python. Una diferencia sintáctica importante aquí presente fue el uso de la palabra reservada end, la cual se sustituyó sin mayor inconveniente por el índice -1 de python.

Aunque se diseñó una prueba para esta función, la precisión hasta una tolerancia de 1e-8 no es un indicador confiable en la medida en que no se tiene control sobre la semilla de las rutinas de números aleatorios (es decir, no se pueden reproducir en python de forma exacta los valores generados en Matlab). Por el contrario, sí es una señal de que la estructura de los vectores de salida se creó correctamente.

Figura 26 Resultado prueba update_user_clock_errors

```
>>> update_user_clock_errors_test(1e-8)
Total muestras: 22
Muestras acertadas: 22
```

genRangeErr: Este script desempeña el papel de centro de generación de errores, pues desde él según las banderas que se hayan indicado en el archivo de configuración, se verifica qué errores son requeridos y se calculan los necesarios. Es entre todas las funciones de GSIM la que más entradas tiene entre obligatorias y opcionales, con 20,

las cuales incluyen desde estructuras con constantes y parámetros hasta frecuencias y banderas booleanas. La salida son cuatro vectores con pseudorangos, rangos delta acumulados, efecto doppler y los errores en los rangos delta.

La complejidad que suponía adaptar la verificación interna de argumentos a la sintaxis de valores por defecto llevó a seguir una aproximación similar a la de Matlab haciendo uso de la función inspect de python. El alto nivel de ramificación y la dependencia de funciones no portadas a este punto hizo que la prueba se centrara únicamente en el fragmento de código utilizado por el primer caso de uso; sin embargo, esto no resulta ser un problema pues la sintaxis en las demás ramas es muy similar, además de que en su mayoría dependen de resultados calculados a partir de aleatorios lo que hace poco práctico discutir acerca de su precisión.

Para la porción de código en que la precisión es relevante, es decir, el caso de uso actual, la prueba arrojó coincidencia absoluta hasta una tolerancia de 1e-8.

Migración de las funciones de escritura. Las tres funciones aquí incluídas tienen bastante en común por lo cual no se dará una descripción individual de cada una de ellas sino una visión general del conjunto. WriteRinexNavFile, writeRinexObsFile y writeSimErrorsFile se encargan de generar archivos rinex de navegación, de observaciones y de errores de simulación, respectivamente. Todas ellas reciben por entradas el nombre del archivo que se generará, la versión de rinex a usar y los parámetros a escribir.

La sintaxis es bastante similar a la de createRinNavV3Msg, portada en la fase pasada, por lo cual el problema más relevante con el que se tuvo que lidiar fue el formateo adecuado de las cadenas de texto; habiendo ya afrontado esta cuestión antes, se aplicó la misma solución. Un inconveniente adicional fue crear y escribir archivos de texto, optándose por abrir el archivo con open indicando la bandera de escritura e iterarlo para ir escribiendo el contenido línea por línea.

La verificación de los archivos se hizo con el comando diff desde la línea de comandos, los resultados se muestran en la figura [27]. Note que la letra al final de cada nombre de archivo indica a cuál corresponde: n para navegación, o para observaciones y e para errores. Las diferencias encontradas se deben a la fecha en que se generó cada archivo.

Figura 27 Verificación de archivos Rinex con diff

```
[root@localhost main]# diff pld_2650.13n /home/gnss/pld_2650.13n
2c2
< WRITERINEXN        PILDO               20150101 183945 LCL PGM / RUN BY / DATE
---
> WRITERINEXN        PILDO               20141022 102250 LCL PGM / RUN BY / DATE
```

```
[root@localhost main]# diff pld_2650.13o /home/gnss/pld_2650.13o
2c2
< WRITERINEXOB       PILDO                20150101 183945 LCL PGM / RUN BY / DATE
---
> WRITERINEXOB       PILDO                20141022 102251 LCL PGM / RUN BY / DATE
```

```
[root@localhost main]# diff pld_2650.13e /home/gnss/pld_2650.13e
2c2
< WRITERRORS          CTAE                 20150101 183946 LCL PGM / RUN BY / DATE
---
> WRITERRORS          CTAE                 20141022 102256 LCL PGM / RUN BY / DATE
```

Migración de las funciones extensas. Imprescindibles en todo caso para correr el simulador, los dos grandes inconvenientes encontrados fueron la gran extensión de estos ficheros (que hizo demorada y tediosa la tarea de su depuración) y la gran cantidad de structs que se maneja en cada uno. Describir en detalle estos archivos, además de precisar una gran cantidad de palabras iría en contra del acuerdo de confidencialidad suscrito con PILDO labs., por este motivo, la información que se suministra sobre estos es aún más general que con las demás funciones.

configfile: Incluye todas las configuraciones necesarias para ejecutar el simulador, de las cuales las más importantes ya han sido mencionadas al comienzo de esta sección. Hacer coincidir la estructura de los arreglos estructurados (utilizados en lugar de los structs de Matlab) con la requerida por las funciones portadas hasta este punto fue una labor que requirió tiempo y esfuerzo adicional. Toda una semana de trabajo se dedicó a este ajuste, en especial en lo concerniente a las dimensiones de los arreglos.

Del mismo modo, tuvo que diseñarse una prueba especial que descendiera a través de la estructura del arreglo y comparara uno a uno los parámetros allí almacenados. Como precaución extra se verificaron las salidas visualmente, imprimiéndolas a la terminal y revisándolas minuciosamente.

main: La más extensa de todas las funciones con más de mil líneas de código. Puede suministrársele un archivo de configuración, aunque de no hacerse éste lo cargará del configfile. Mientras se ejecuta se imprime en pantalla el estado actual de la simulación, cuando termina se generan tres archivos en formato rinex (utilizando las funciones de escritura previamente portadas). Los mensajes mostrados en pantalla durante una ejecución típica pueden verse en la figura [28].

Aunque reemplazar los structs fue en general complejo, uno de ellos fue especialmente difícil pues crecía dentro de un ciclo para iteración tras iteración. Finalmente se recurrió a darle una estructura más flexible, mezclando las estructuras y tipos de datos de los que python y numpy disponen. Por su parte los componentes gráficos se trataron de formas distintas, la barra de progreso se reemplazó por una versión en línea de comandos similar a la que se ve cuando se instala un programa desde ella; un explorador

Figura 28 Resultado en pantalla del main

```
>>> from main import *
m>>> main()
Dynamic mode enabled!
Trajectory file succesfully loaded!
Radiation pattern succesfully loaded!
Week number: 1759
Validity time interval for the navigation ephemeris:
First epoch: 0 s
Last epoch: 22 s
Processing GPS observation measurements...
[************************] 100.00%
Processing GALILEO observation measurements...
[************************] 100.00%

RINEX navigation file saved
RINEX observation file saved
RINEX formatted error file saved
**********************************************************************
******************** SIMULATION COMPLETED *********************
The following files have been created:
pld_2650.13o    --> RINEX v3 Observation File

pld_2650.13n    --> RINEX v3 Navigation File

pld_2650.13e    --> Simulation Errors File

**********************************************************************
**********************************************************************
```

de archivos incluído no se portó pues GSIM actualmente se trabajar en entornos no gráficos e igualmente no es indispensable para el funcionamiento del mismo.

Descripción Caso de uso II: El segundo caso de uso se caracteriza principalmente por su enfoque en el manejo de las banderas del factor de escalado de los errores parte fundamental de la estructura de los parámetros de simulación que contiene GNSS. Los parámetros más significativos son:

- Factor de escalado del ruido térmico: Se encuentra presente dentro del parámetro referente al error de ruido del reloj gaussiano y que interviene en el resultado de las funciones svClkErrGen y genRangeErr.
- Factor de escalado del error troposférico: Es el encargado de generar el retraso troposférico en metros, correspondiente a las posiciones de los usuarios y del satélite en coordenadas cartesianas (ECEF); la función que realiza este cálculo es mopstropgen.
- Factor de escalado del error del reloj satelital: Su cálculo depende de varios factores tales como el número de satélites, el número de épocas para las cuales se

Figura 29 Banderas caso de uso II

```
% GPS simulation flag
gnssParam.gSimFlag = [4 3 3 3];




% GALILEO simulation flag
gnssParam.eSimFlag = [3 3 3 3];%[0 0 0 0];%



% GNSS Errors Scaling Factor flags
gnssParam.esf = [1 1.5 0 0 1 1 0 1 0];
%  gnssParam.esf = [0 0 0 0 0 0 0 0 0]
```

debe generar el error del reloj y el tiempo de aplicabilidad de la señal efemérides, entre otros; las funciones encargadas de realizar este proceso son svClkErrGen y computeSvClkSamples.

- Efecto relativista: Es un componente periódico debido a la excentricidad orbital una característica propia de la trayectoria de cada satélite en este caso. La funciones involucradas son compute_anomaly_eccentricity e ionogen.

La migración de estas funciones no introdujo nuevos desafíos, los resultados de las pruebas y las precisiones alcanzadas pueden verse en las figuras siguientes.

Figura 30 Resultados Pruebas caso II

```
>>> compute_anomaly_eccentricity_test(1e-900)
Total muestras: 1278
Muestras acertadas: 1278
```

```
>>> computeSvClkSamples_test(1e-1)
Total muestras: 1
Muestras acertadas: 1
```

```
>>> ionogen_test(1e-13)
Total muestras: 6267
Muestras acertadas: 6267
```

```
>>> mopstropgen_test(1e-13)
Total muestras: 6480
Muestras acertadas: 6480
```

```
>>> svClkErrGen_test(1e-1)
Total muestras: 2
Muestras acertadas: 2
```

8.4. OPTIMIZACIÓN DEL SIMULADOR

Una vez terminado el proceso de migración se midió el tiempo de ejecución total de GSIM para los dos casos de uso. Para tomar los tiempos en Matlab se utilizaron las funciones tic y toc; en tanto en python las mediciones se realizaron con el comando repeat del módulo timeit como se indica a continuación:

```
1 from main import *
2 import timeit
3 t = timeit.Timer(main)
4 t.repeat(10,1)
```

Los resultados pueden verse en las tablas [8.6] y [8.7]. Siguiendo la sugerencia de la documentación de timeit [30], se escogieron como tiempos de referencia los menores de cada caso, resaltados en negrilla. El porcentaje de optimización se calculó por medio

Tabla 11 Tiempos antes de optimización caso I

Toma	Matlab	Python
1	30,013117	24,547748
2	26,468131	24,524453
3	26,465613	24,548029
4	26,429471	**24,307907**
5	26,438206	24,593839
6	26,392086	24,684631
7	**26,341569**	24,56531
8	26,359482	24,367200
9	26,442041	24,556623
10	26,375660	24,702987

de una regla de tres simple, con la fórmula siguiente:

$$\%optimización = \left(1 - \frac{t_{matlab}}{t_{python}}\right) \times 100$$

Tabla 12 Tiempos antes de optimización caso II

Toma	Matlab	Python
1	**28,854778**	26,977947
2	28,912755	27,115818
3	28,957351	27,115152
4	30,420943	27,127548
5	31,444267	27,125760
6	33,133946	27,100454
7	32,970865	**26,927785**
8	32,877227	27,087094
9	29,024145	27,096858
10	28,991543	27,096245

Obteniendo resultados de 7,72 % y 6,68 %, respectivamente para cada uno de los casos. Siendo ambos porcentajes más bajos de lo esperado, se resolvió aplicar algunas técnicas de optimización al código. De los resultados del perfilador se decidió centrar los esfuerzos de optimización en las 6 funciones que consumían más tiempo, las cuales representaban cerca del 80 % del tiempo total. Todas las alternativas consideradas se probaron antes de forma aislada utilizando algunas de las muestras capturadas. Dentro de las optimizaciones aplicadas se incluyeron las siguientes:

- Se utilizó un par de funcionalidades del módulo de optimización NumbaPro, una versión mejorada de numba, en las que se pretendía mejorar el tiempo el que el intérprete de python tarda en descifrar el tipo de variable sobre el cual debe operar; las librerías usadas fueron @jit y @autojit pero el resultado no fue el esperado debido a que la mayoría de los parámetros de entrada de las funciones que se intentanron optimizar eran arreglos estructurados y su tipo variaba de acuerdo al campo del vector en donde se estaba operando.

- Se reemplazaron los condicionales si más sencillos por comprensiones de listas, esto solo fue posible en la función computeSvTimeTrans pues era la única que tenía las mencionadas cláusulas.

- Se precalcularon algunos valores que eran utilizados varias veces dentro de la función computeSvPosRin (computados una y otra vez) y se almacenaron en variables temporales.

- Se almacenaron en variables llamadas a funciones que se efectuaban repetidamente utilizando la sintaxis punto (f.i. variable.append). Esta modificación se hizo en genSatPosition y en computeSvTimeTrans.

- Se hicieron algunos cambios de sintaxis que restaron un par de segundos al tiempo de ejecución (utilizar concatenate en lugar de hstack o vstack, range(n) por

range(0,n), entre otras). Esto se aplicó a la mayoría de funciones en el simulador, incluido el main y el archivo de configuración.

- Se reemplazaron ciertos cálculos matemáticos por versiones alternativas menos costosas computacionalmente; por ejemplo, el cuadrado de las variables se cambió por el producto por sí misma. Esta optimización se aplicó a xyz2llh, xyz2enu y al main.
- Se reajustó el orden de algunas ramificaciones condicionales en genSatPosition
- Se sustituyó el formateado con format por %, este cambio no fue demasiado significativo en cuanto a tiempos pero tuvo la ventaja adicional de dar a la sintaxis más parecido a la versión original en Matlab. Esto se hizo en las funciones createRinNavV3Msg, writeRinexNavFile, writeRinexObsFile y writeSimErrorsFile. Se efectuaron otros ajustes menores a ciertas funciones particulares.

Hecho los cambios anteriores se tomaron tiempos nuevamente y se obtuvieron los resultados presentados a continuación (no se volvieron a medir tiempos para Matlab pues este no fue modificado).

Tabla 13 Tiempos después de optimizar en python con archivo de trayectoria pequeño

Toma	Caso 1	Caso 2
1	19,433538	22,524615
2	19,376225	**22,492692**
3	**19,363775**	22,533903
4	19,390301	22,532426
5	19,402101	22,531562
6	19,423878	22,535051
7	19,419510	22,592223
8	19,392324	22,592555
9	19,412090	22,551464
10	19,436726	22,611875

Los porcentajes de optimización fueron ahora de 26,49 % y 22,05 %. Con el fin de corroborar estos porcentajes y facilitar la medición, se solicitó un archivo de trayectorias más grande, el cual fue proporcionado por pildo labs. Este nuevo archivo tenía 8000 líneas y 10 s entre épocas. La nueva medición llevada a cabo puede verse en la tabla [8.9], en este caso el efecto de las optimizaciones fue mayor, arrojando porcentajes de optimización de 44,07 % y 39,65 %.

Habiendo alcanzado un porcentaje bueno de optimización, se procedió a entregar la versión 2 del simulador para su utilización por parte de los usuarios. De este modo, se

Tabla 14 Resultado optimizado caso I con archivo de trayectoria grande

Toma	Matlab	Python
1	**180,22**	**100,80**
2	181,15	101,92
3	180,78	102,04
4	-	102,20
5	-	102,12

Tabla 15 Resultado optimizado caso II con archivo de trayectoria grande

Toma	Matlab	Python
1	196,60	**116,05**
2	192,75	117,08
3	**192,30**	117,97
4	-	117,99
5	-	118,06

espera que archivos que antes tardaban hasta 24 horas en ser generados, puedan tenerse listos en menos de 16 horas.

9. CONCLUSIONES

GSIM es un software de la propiedad de PILDO Labs cuya sede principal se encuentra en Barcelona. Gracias al profesor Raúl Ramos Pollán adscrito a la escuela de Ingeniería de Sistemas y sus relaciones con la empresa nombrada anteriormente se logró el desarrollo de este tipo de proyecto, generando un primer acercamiento y creando un lazo de confianza para investigaciones a futuro.

Se concluye que según lo expuesto en el análisis presentado en la viabilidad del proyecto con lo relacionado a la migración, python fue una elección conveniente, no sólo porque se logró una notable mejora en los tiempos, sino también, porque se logo cumplir con las fechas establecidas para cada entrega lo cual fue posible debido al trabajo continuo de los autores y por la similitud de desarrollo que tanto Matlab como python presentan.

La metodología seleccionada fue muy exacta al momento del desarrollo y generación de versiones ya que con la utilización de SVN se logró un ambiente de desarrollo colaborativo y organizado, facilitando el monitoreo continuo del proyecto.

Python con su módulo numbapro permitió generar pruebas de optimización con sus decoradores @autojit y @jit de las cuales se concluye que para lograr una mejora en los tiempos de cómputo utilizando este módulo se deben indagar otras opciones más avanzadas como lo son: la vectorización, o la programación de numba para CUDA y GPU.

Luego de realizar las debidas pruebas de rendimiento del software y con ayuda de los perfiladores se concluye que la escritura de archivos en python en este caso fue un 96 % más rápida que su contraparte en Matlab, lo cual junto a la optimización de los fragmentos críticos del simulador GSIM, lograda con el uso de las funciones de numpy y el replanteamiento del código se alcanzó una mejora final promedio del 30 % en el tiempo de cómputo.

Un archivo promedio de trayectorias en la versión antigua se demoraba alrededor de 48 horas computando. Para ponerlo en perspectiva en un mes de trabajo se podían correr al menos 11 pruebas ahora con la nueva versión portada de GSIM se podrán realizar casi 16 pruebas en el mismo rango de tiempo.

10. RECOMENDACIONES

El proceso de migración todavía no concluye, a la fecha se ha portado el 75 % del simulador faltando por traducir algunas de las funciones generadoras de los errores menos comunes. Como paso natural se recomienda que una vez portada la totalidad de GSIM se construya y agregue una interfaz gráfica de usuario, de forma tal que el uso del simulador sea más amigable para usuarios no habituados a la línea de comandos.

En lo que respecta a optimización, existen tres alternativas que tienen el potencial de generar resultados interesantes: Paralelizar el código, utilizar paquetes de optimización automática y traducir las partes críticas del código a lenguaje C. La primera se sugiere como la opción más viable de todas ellas considerando que el programa tiene varios ciclos, muchos de los cuales realizan cálculos que son independientes de las iteraciones previas.

Para la segunda, en el ámbito académico se puede sugerir una exploración más profunda de las opciones que ofrece NumbaPro [22]; sin embargo, hay que tener en cuenta que para fines comerciales se obliga a pagar por el derecho de uso, y por tanto sería necesario hacer una evaluación de la relación costo-beneficio entre el precio de la licencia y la aceleración conseguida.

La última alternativa sólo se aconseja si no se han conseguido los resultados esperados con lo citado anteriormente, porque además de compleja, quitaría legibilidad al código traduciéndose en una mayor dificultad para su mantenimiento.

REFERENCIAS

1. A GPS Tutorial: Basics of High-Precision Global Positioning Systems. Javad Positioning Systems. 1998

2. ABART, C., Simulating GNSS constellations - GPS, Galileo and SBAS. ELMAR, 2008. 50th International Symposium, septiembre de 2008, vol.2, no., pp.569,572.

3. AIREY, L. Backscatter Gain and Array Modeling for a Large-Aperture High-Power Radar. Disertación, Worcester Polytechnic Institute, 2014.

4. BERGER, M. J. et al., The GeoClaw software for depth-averaged flows with adaptive refinement, Advances in Water Resources 34, 2011, pp. 1195-1206.

5. BISNATH, S. et al., Development of a Software-based Multi-GNSS Observable Simulator. En: Proceedings of the 23rd International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS 2010), Septiembre de 2010, pp. 892-899.

6. CANÓS, J.H., et al., Metodologas Ágiles en el Desarrollo Software. Disponible en: http://noqualityinside.com.ar/nqi/nqifiles/XP_Agil.pdf

7. CONSTANTINESCU, A. et al., Performance Evaluation and Analysis of a Hybrid Version of a Software Defined GPS/Galileo GNSS Receiver for Dynamic Scenarios. En: Proceedings of the 17th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2004), Septiembre de 2004, pp. 2697-2708

8. CONTINUUM ANALYTICS, NUMBAPRO. Disponible en: http://docs.continuum.io/numbapro/index.html

9. DEIMOS SPACE, GRANADA: Galileo Receiver ANalysis And Design Application. Disponible en: `http://ww1.deimos-space.com/granada/`

10. DIEZ, J. et al., GRANADA: a low-cost commercial simulator for GNSS receivers design and evaluation. En: Proceedings of NAVITEC 2006, ESA/ESTEC, 2006, pp. 11-13.

11. FERNÁNDEZ, A. et al., Navigation Algorithm Optimisation for Combined Galileo/GPS Receivers with the GRANADA Environment and Navigation Simulator. En: Proceedings of the 18th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2005), Septiembre de 2005, pp. 1939-1944.

12. GROH et al., Complexity Reduced Multipath Mitigation in GNSS with the GRANADA Bit-True Software Receiver. En: Proceedings of IEEE/ION PLANS 2008, Mayo de 2008, pp. 418-423.

13. HERNNDEZ, M.J. y ZOENIZA, J.M. GPS data processing: code and phase Algorithms, Techniques and Recipes. CPET. 1 Ed. 2005.

14. JEFFREY, C., An Introduction to GNSS. GPS, GLONASS, Galileo and other Global Navigation Satellite Systems. 1 Ed., Novatel Inc., 2010.

15. JURICA, P., y VAN LEEWEN, C., OMPC: An Open-Source Matlab-to-Python Compiler. En: Frontiers in Neuroinformatics 3 (2009), febrero de 2009, pp. 3 - 5.

16. KITNER, P. y HINKS, J., GNSS and Ionospheric Scintillation. En: Inside GNSS, 2009.

17. KUMAR, P., GPS-Gyan: An Open-Source GPS Software Simulator Using Object-Oriented System Design And Modeling Framework. Tesis, North Carolina State University, Julio 2010, pp. 11 - 13.

18. LAZAR, S. et al., A GPS Modernization Simulator. En: Proceedings of the 11th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 1998), Nashville, TN, Septiembre de 1998, pp. 1887-1893.

19. Metodologías Ágiles. Disponible en : http://es.slideshare.net/mikyWatt/metodologias-agiles-15338080#

20. NAVIPEDIA, Klobuchar Ionospheric Model. Disponible en: http://www.navipedia.net/

21. NAVIPEDIA. The reference for Global Navigation Satellite Systems. Disponible en: www.navipedia.net.

22. NAVSYS, GNSS Signal Architect: GNSS simulation and testing software. Disponible en: `www.navsys.com/Products/signal_architect_simulator_software_new.htm`

23. PILDO, Pildo Labs. Disponible en:`http://www.pildo.com/`

24. PRASAD, R. y RUGGIERI, M., Applied Satellite Navigation Using GPS, Galileo and Augmentation Systems. Artech House, 2005.

25. PYTHON, Python Documentation. Disponible en: https://docs.python.org/

26. SPIRENT, simGen: A fully flexible Multi-GNSS simulator software suite. Disponible en: `http://www.spirent.com/Products/SimGEN`

27. SILVA, J.S. et al., An Integrated and Cost-Effective Simulation Tool for GNSS Space Receiver Algorithms Development. En: Proceedings of the 26th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS+ 2013), Septiembre de 2013, pp. 1951-1961.

28. WARD, P.W., Using a GPS Receiver Monte Carlo Simulator to Predict RF Interference Performance. En: Proceedings of the 10th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 1997), Septiembre de 1997, pp. 1473-1482.

29. WINKEL, J.. et al., SNSS - Simulated Navigation Satellite System; Simulating a Generic GNSS Receiver in Virtual Environments. En: Proceedings of the 11th

Anexos

RESULTADOS DEL PERFILADOR DE MATLAB

Estos son los resultados del perfilador de Matlab, se incluyen los resultados generales para el main y específicos para las genSatPosition y allan_plot_to_Q. En ellos se puede ver la dependencia de funciones, en forma de hijas y padres, acompañado del tiempo consumido en cada una de ellas. De manera análoga, se muestran el tiempo propio de la función (invertido ejecutando su propio código) y el tiempo acumulado (tiempo total transcurrido).

Profile Summary

Generated 01-Feb-2015 22:58:27 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
allan_plot_to_Q	22	0 s	0.000 s	
blanks	359	0 s	0.000 s	
boolean	2	0 s	0.000 s	
clock_states_F_and_Q	22	0 s	0.000 s	
close	3	0.374 s	0.025 s	
...ddenHandles',oldUDDShowHiddenHandles)	2	0 s	0.000 s	
close>checkfigs	5	0.025 s	0.025 s	
...ddenHandles',oldUDDShowHiddenHandles)	2	0 s	0.000 s	
close>getEmptyHandleList	5	0.025 s	0.025 s	
close>handleFromNumber	2	0 s	0.000 s	
close>request_close	3	0.324 s	0.075 s	
close>request_close_helper	4	0.199 s	0.075 s	
close>safegetchildren	1	0 s	0.000 s	
closereq	2	0.050 s	0.050 s	
colormap	2	0.025 s	0.025 s	
...s.mlwidgets.workspace.WhosInformation (Java method)	1	0 s	0.000 s	
computeEphUpdateTime	2	0 s	0.000 s	
computeSvClkSamples	1	0.224 s	0.050 s	

computeSvPosRin	121248	48.223 s	48.223 s	
computeSvTimeTrans	138	121.132 s	1.322 s	
compute_anomaly_eccentricity	2228	1.571 s	1.571 s	
compute_mask_angle	528	1.521 s	0.199 s	
configfile	1	0.224 s	0.175 s	
convertObst2D	1	0.150 s	0.050 s	
createRinNavV3Msg	58	2.070 s	0.673 s	
fgetl	1437	0.374 s	0.374 s	
gcbf	4	0 s	0.000 s	
genRangeErr	184	50.068 s	6.982 s	
genSatPosition	3789	146.714 s	41.267 s	
genSvSnr	138	1.696 s	0.873 s	
gpst2utc	62	0.424 s	0.424 s	
int2str	1279	0.524 s	0.524 s	
interp1	528	1.322 s	1.022 s	
interp1>parseinputs	528	0.299 s	0.249 s	
interp1>sanitycheckmethod	528	0.050 s	0.050 s	
ionogen	6684	16.906 s	4.089 s	
iscellstr	46	0 s	0.000 s	
ismember	186	0.274 s	0.100 s	
ismember>ismemberR2012a	186	0.175 s	0.175 s	

loadradpat	1	0.374 s	0.125 s	
main	1	273.332 s	14.811 s	
mean	6743	2.867 s	2.867 s	
mopstropgen	6684	24.610 s	14.038 s	
num2str	15463	27.827 s	15.285 s	
num2str>cellPrintf	14184	6.658 s	6.658 s	
num2str>strvrcat	14184	5.361 s	1.720 s	
onCleanup>onCleanup.delete	3	0 s	0.000 s	
onCleanup>onCleanup.onCleanup	3	0 s	0.000 s	
readYuma	2	0.873 s	0.424 s	
resize_svid_svmat	184	0.848 s	0.125 s	
setdiff	184	0.723 s	0.100 s	
setdiff>setdiffR2012a	184	0.623 s	0.175 s	
sqrtm	22	0.050 s	0.050 s	
str2double	1370	0.274 s	0.274 s	
strcat	359	0.723 s	0.723 s	
strjust	14184	3.640 s	3.640 s	
svClkErrGen	2	0.175 s	0.175 s	
uitools/private/uiwaitbar	48	0.224 s	0.224 s	
uitools/private/warnfiguredialog	48	0.050 s	0.025 s	

waitbar>updateWaitbar	46	0.199 s	0.000 s	
workspacefunc	4	0.150 s	0.025 s	
...pace.MatlabWorkspaceListener.swl(swl)	1	0 s	0.000 s	
...pace.MatlabWorkspaceListener.swl(swl)	1	0 s	0.000 s	
workspacefunc>getAbstractValueSummary|	2	0.025 s	0.025 s	
workspacefunc>getCleanupHandler	1	0 s	0.000 s	
workspacefunc>getShortValueObject|	1	0.025 s	0.025 s	
workspacefunc>getShortValueObjects|	1	0.025 s	0.000 s	
workspacefunc>getStatObject|	2	0.050 s	0.025 s	
workspacefunc>getStatObjectM	2	0 s	0.000 s	
workspacefunc>getStatObjects|	2	0.075 s	0.025 s	
workspacefunc>getWhosInformation	1	0.025 s	0.025 s	
workspacefunc>getclass	2	0 s	0.000 s	
workspacefunc>num2complex	2	0.025 s	0.000 s	
writeRinexNavFile	1	2.219 s	0.025 s	
writeRinexObsFile	1	8.577 s	8.303 s	
writeSimErrorsFile	1	39.845 s	13.041 s	
xyz2enu	136728	73.906 s	40.269 s	
xyz2llh	150097	38.623 s	38.623 s	

Self time is the time spent in a function excluding the time spent in its child functions. Self time also includes overhead resulting from the pr

Profile detallado para la función allan_plot_to_Q. Profile detallado para la función

Parents (calling functions)

Function Name	Function Type	Calls
clock_states_F_and_Q	function	22

Lines where the most time was spent
No measurable time spent in this function

Line Number	Code	Calls	Total Time	% Time	Time Plot
16	Q=[q11,q12;q12,q22];	22	0 s	0%	
15	q11=h_0/2*DeltaT+2*h_m1*DeltaT...	22	0 s	0%	
14	q22=h_0/2/DeltaT+4*h_m1+8/3*pi...	22	0 s	0%	
13	q12=h_m1*DeltaT+pi^2*h_m2*(Del...	22	0 s	0%	
All other lines			0 s	0%	
Totals			0 s	0%	

Children (called functions)
No children

genSatPosition.

Parents (calling functions)

Function Name	Function Type	Calls
main	function	666
computeSvTimeTrans	function	3123

Lines where the most time was spent

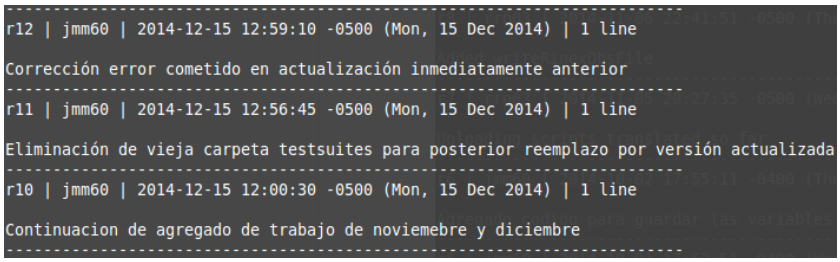
Line Number	Code	Calls	Total Time	% Time	Time Plot
120	svxyz = computeSvPosRin(SQRTS...	121248	64.231 s	43.8%	
139	svenu = xyz2enu(svxyz,usrece);...	121248	61.314 s	41.8%	
140	el = (180/pi)*atan2(svenu(3),n...	121248	4.987 s	3.4%	
93	if isempty(find (TWEV(N,:)~=0,...	121248	2.394 s	1.6%	
141	end	121248	2.344 s	1.6%	
All other lines			11.445 s	7.8%	
Totals			146.714 s	100%	

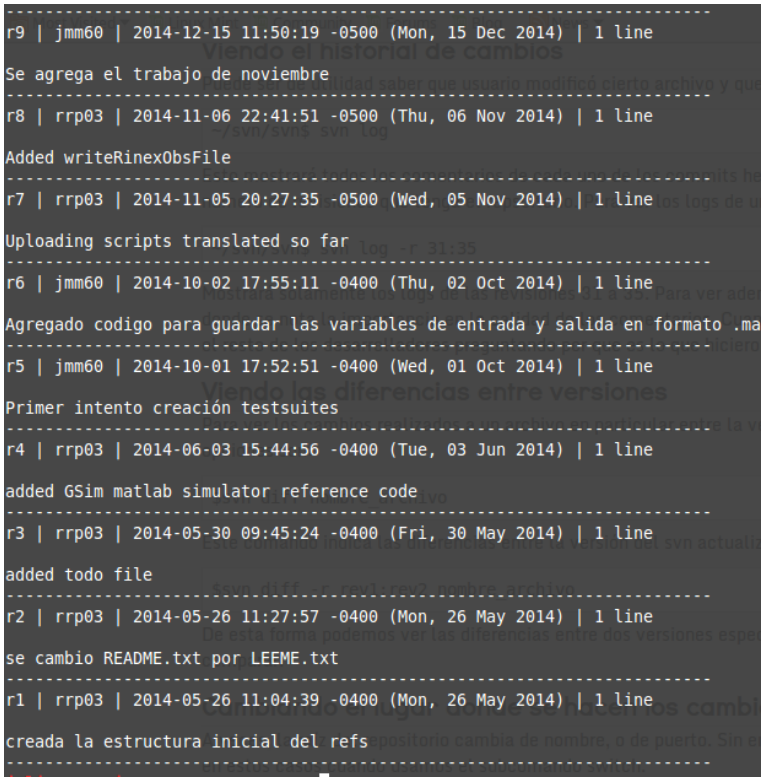
Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
xyz2enu	function	121248	57.225 s	39.0%	
computeSvPosRin	function	121248	48.223 s	32.9%	
Self time (built-ins, overhead, etc.)			41.267 s	28.1%	
Totals			146.714 s	100%	

REGISTRO DE REVISIONES DE SVN

Se presenta el historial de revisiones de Subversion visto desde la línea de comandos de Linux, junto a cada versión puede observarse un comentario breve sobre su naturaleza.





PLANTILLA PARA PRUEBA DE FUNCIONES.

En este anexo se presenta el archivo genérico de prueba en base al cual se diseñaron las pruebas específicas para cada función.

```
import glob
import os
import sys,inspect
# Agrega carpetas requeridas al path
currentdir = os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe
    ())))
parentdir = os.path.dirname(os.path.dirname(currentdir))+'/ruta/al/directorio'
sys.path.insert(0,parentdir)
# Importa funciones necesarias
from script1 import funcion1
from script2 import funcion2
# Funcion a llamar para la prueba
def funcion_test(tol):
        path = ''
        sum1=0
        sum2=0
        # Busca todos los archivos con extensi n mat en la carpeta actual
        for infile in glob.glob(os.path.join('*.mat') ):
                # Carga uno a uno los .mat a una variable temporal
                temp = loadmat(infile)
                # Llama la funci n y almacena su salida
                salida = funcion(temp['entrada'])
                # Verificacion por tolerancia
                if 1 in (abs(temp['salida']-salida)>tol):
                        # Muestra fallida
                        sum1=sum1+1
                else:
                        # Muestra exitosa
                        sum2=sum2+1
        # Impresi n de resultados
        print "Total muestras:", sum1+sum2
        print "Muestras acertadas:" , sum2
```