TRABAJO FIN DE GRADO

Machine Learning aplicado a la propagación de alta precisión de trayectorias de satélites

Realizado por
José Calderón Valdivia

Para la obtención del título de
Doble Grado en Ingeniería Informática - Tecnologías Informáticas y Matemáticas

Dirigido por
Daniel Ayala Hernández
Luis Valencia Cabrera

Realizado en el departamento de
Lenguajes y Sistemas Informáticos

Convocatoria de junio, curso 2022/23

Agradecimientos

Quiero agradecer a mi familia por apoyarme en todo momento y haber estado ahí siempre que lo he necesitado; sin ellos no habría sido posible.

A mis amigos, cuya compañía ha sido un pilar fundamental durante estos años de carrera.

A mis profesores por todo lo que me han enseñado.

A todo el grupo de investigación *DEAL*, que desde el primer día me acogieron como a uno más.

A Rafael Ayala, autor del paquete *asteRisk*, por resolver muchas dudas que surgieron durante el desarrollo del proyecto.

Y especialmente quiero agradecer a mis tutores, Daniel Ayala y Luis Valencia, su ayuda a lo largo de todo el proyecto.

Resumen

La predicción precisa de las posiciones de los satélites es fundamental para una amplia gama de aplicaciones, que van desde las comunicaciones y navegación por satélite hasta la teledetección y la planificación de misiones espaciales. La capacidad de predecir con precisión las trayectorias de los satélites proporciona información crítica para el funcionamiento seguro y eficiente de estos sistemas espaciales. Sin embargo, debido a las diversas fuerzas y perturbaciones que afectan a los satélites en órbita, lograr una predicción precisa enfrenta desafíos técnicos y científicos significativos.

Para la modelización de trayectorias existen principalmente dos tipos de modelos: los analíticos, como SGP4 y SDP4; y los de alta precisión, como HPOP. Como su nombre indica, los de la segunda familia aportan más fiabilidad; sin embargo, el coste computacional es demasiado elevado para realizar predicciones a medio o largo plazo.

En el presente trabajo se utiliza el paquete de R *asteRisk* para la creación de un conjunto de datos de entrenamiento y validación y, posteriormente, desarrollar modelos de redes neuronales para corregir las predicciones realizadas por un modelo analítico y así obtener predicciones precisas en un tiempo reducido.

Previamente a la creación del conjunto de datos se desarrolla un método de optimización de los parámetros del modelo SGP4 y SDP4, de modo que en el conjunto de datos se pudieron incluir tanto las predicciones con los parámetros base como con los parámetros optimizados. Por tanto, también se pudo estudiar cómo variaba la capacidad de predicción del error de las redes neuronales según si los parámetros orbitales de SGP4 y SDP4 habían sido previamente optimizados o no.

Todo esto es posible gracias la gran cantidad de datos disponibles sobre trayectorias de satélites en un formato estandarizado, ya sean archivos RINEX o archivos TLE.

Palabras clave: aprendizaje automático, redes neuronales, propagación, trayectoria, satélite, modelos analíticos, modelos de alta precisión

Abstract

Accurate prediction of satellite positions is crucial for a wide range of applications, ranging from satellite communications and navigation to remote sensing and space mission planning. The ability to accurately predict satellite trajectories provides critical information for the safe and efficient operation of these space systems. However, due to the various forces and perturbations affecting satellites in orbit, achieving accurate prediction faces significant technical and scientific challenges.

For trajectory modelling, there are mainly two types of models: analytical models, such as SGP4 and SDP4; and high-precision models, such as HPOP. As the name suggests, the latter family provides greater reliability; however, the computational cost is prohibitively high for making medium or long-term predictions.

In this work, the R package *asteRisk* is used to create a training and validation dataset and, subsequently, develop neural network models to correct the predictions made by an analytical model, thus obtain accurate predictions in a reduced time frame.

Prior to the creation of the dataset, a parameter optimization method is developed for the SGP4 and SDP4 models, allowing both baseline and optimized parameter predictions to be included in the dataset. Therefore,the study also investigates how the neural networks' prediction error varies depending on whether the orbital parameters of SGP4 and SDP4 have been previously optimized or not.

All of this is made possible by the wealth of available data on satellite trajectories in a standardized format, whether it be RINEX files or TLE files.

Keywords: machine learning, neural networks, propagation, trajectory, satellite, analytical models, high-precision models

Índice general

Índice de figuras

Índice de cuadros

Índice de extractos de código

1. Introducción

Para enmarcar el contexto del proyecto nos remontamos al 4 de octubre de 1957, con el lanzamiento por la Unión Soviética del primer satélite artificial del mundo, el Sputnik-1. Desde entonces se han lanzado 15717 objetos al espacio, de los cuales 11166 permanecen en órbita a día de hoy [14]. Debido a esto resulta de suma importancia poder realizar un seguimiento para planificar su trayectoria y movimientos, de modo que se garantice la eficiencia, seguridad y éxito de misiones espaciales y comunicaciones, entre otras muchas aplicaciones.

La modelización de las órbitas, de forma tradicional, se ha realizado mediante complejos modelos matemáticos como HPOP (High Precision Orbital Propagator), en los que intervienen factores como la posición y masa de los cuerpos celestes del sistema solar, la incidencia de fotones sobre el satélite, el efecto de las mareas, etc. Debido a su complejidad, su aplicación es costosa computacionalmente, y es difícil mejorarlos para subsanar errores en las predicciones.

La propagación de trayectorias de satélites consiste en, a partir de la posición y velocidad de un satélite en un instante, predecir cuál será su trayectoria y velocidad en un instante futuro. Dicha trayectoria se verá determinada por las fuerzas gravitatorias y centrífugas que actúan sobre el satélite, así como otras perturbaciones como la influencia del Sol, la Luna y otros cuerpos celestes. Para esto existen fundamentalmente dos tipos de modelos:

1. Los modelos analíticos (SGP4 y SDP4), o modelos de perturbaciones simplificadas, que modelizan la órbita de forma aproximada, permitiendo propagar de forma muy rápida pero poco precisa a largo plazo.
2. Los modelos de alta precisión (HPOP), que modelizan mediante ecuaciones diferenciales todas las fuerzas que actúan sobre el satélite y, por tanto, contribuyen a su aceleración. Estos modelos son muy precisos pero muy costosos de aplicar para predicciones a largo plazo.

El objetivo de este trabajo se puede resumir en estudiar la viabilidad de utilizar modelos de redes neuronales para corregir las predicciones realizadas por los modelos SGP4 y SDP4, a los que nos referiremos conjuntamente como SGDP4. Podemos distinguir dos grandes bloques, uno dedicado al estudio de los fundamentos de la propagación de trayectorias y otro dedicado a la generación de un conjunto de datos de satélites y al entrenamiento de modelos de redes neuronales con dichos datos, que intenten corregir las predicciones de los modelos analíticos.

En el estudio teórico se exponen los fundamentos matemáticos básicos para comprender el movimiento de los satélites, se definen los formatos estandarizados de intercambio de datos y se termina profundizando en la modelización de órbitas, de modo que sirva para comprender el funcionamiento de los modelos de propagación más utilizados actualmente.

Las contribuciones de este trabajo han sido las siguientes:

- Desarrollo de algoritmos de optimización de parámetros orbitales con el objetivo de reducir el error de la predicción, utilizando una serie de puntos de la trayectoria como referencia.
- Generación de un conjunto de datos de posiciones y velocidades de satélites y astros, con las posiciones predichas por el modelo SGDP4 con los parámetros orbitales previamente optimizados y sin optimizar.
- Entrenamiento de redes neuronales capaces de corregir predicciones realizadas con el modelo SGDP4.

2. Estudio Previo

En este capítulo se tratan los aspectos previos a la ejecución del proyecto, como son el alcance, la metodología empleada, la planificación y estimación de costes. Finalmente se mostrarán los valores obtenidos tras la finalización del proyecto.

2.1. Objetivos

Entre los objetivos del presente trabajo podemos destacar como principales los siguientes:

- El estudio y comprensión de los fundamentos matemáticos de los principales modelos de propagación de trayectorias.
- El desarrollo de modelos de redes neuronales que ayuden a mejorar la precisión de ciertos modelos analíticos (SGP4 y SDP4) de propagación de órbitas de satélites.

También cabe mencionar otros objetivos, de carácter docente, personal y técnico, como pueden ser:

- Mostrar el potencial de las redes neuronales con una aplicación práctica de un peso considerable.
- Poner a prueba mis capacidades tras cinco años de formación universitaria e iniciarme en el mundo de la investigación.
- Contribuir en el área aeroespacial ofreciendo una pequeña solución a uno de los grandes problemas de la industria a día de hoy.

2.2. Metodología

Para la ejecución del presente proyecto se optó por el empleo de metodologías ágiles, ya que permiten adaptar el modo de trabajo a las condiciones del proyecto aportando flexibilidad e inmediatez de respuesta, lo que reduce los costes e incrementa la productividad.

La metodología ágil escogida resulta ser SCRUM, que está especialmente indicada para proyectos en entornos complejos, con requisitos cambiantes o poco definidos y donde la innovación, competitividad, flexibilidad y productividad son fundamentales.

La filosofía de SCRUM es la ejecución de ciclos temporales cortos y de duración fija (iteraciones, comúnmente conocidas como sprints), que suelen ser de 2 semanas, aunque en algunos casos se puede extender hasta 4. Cada iteración tiene que proporcionar un resultado completo, que suponga un incremento de producto final y que sea susceptible de ser entregado con el mínimo esfuerzo al cliente cuando este lo solicite.

Aunque dicha metodología esta orientada al trabajo en equipo se utilizó en el desarrollo de este proyecto individual, con algunas modificaciones como no considerar *Product Owner* o *Scrum master*. Se comenzó con un análisis previo de los requisitos, de manera que se priorizaron y ordenaron. Como fase previa a la ejecución se diseñaron los pasos a seguir para la obtención de datos y la implementación de los modelos de redes neuronales. Después de este diseño, en la etapa de implementación, comenzaron a emplearse las indicaciones Scrum, organizando cada tarea como un sprint, con reuniones semanales de actualización a los tutores y solución de dudas. Al tratarse de un proyecto individual, no aportaban mucho valor los procesos *sprint review* y *sprint retrospective*, por lo que se prescindió de ellos.

2.3. Planificación

La planificación del proyecto se ha inspirado en "Los 7 Pasos del Aprendizaje Automático", de Yufeng Guo, que se resumen en los siguientes puntos:

1. Recopilación de datos
2. Preparación de los datos
3. Elección del modelo
4. Entrenamiento del modelo
5. Evaluación del modelo
6. Ajuste de parámetros
7. Realización de predicciones

A continuación se muestra la distribución de las tareas a realizar para la consecución del proyecto, incluyendo las horas estipuladas para cada una de ellas, teniendo su inicio el día 18 de noviembre del año 2022 y su finalización el día 16 de junio del año 2023.

Tarea	Dedicación estimada	Dedicación real
Investigación	**116 horas**	**114 horas**
Recopilación de artículos	2 horas	2 horas
Lectura en profundidad de artículos	4 horas	6 horas
Fundamentos de las órbitas	30 horas	25 horas
Modelos de propagación	60 horas	65 horas
Resolución numérica de ec. dif.	20 horas	16 horas
Implementación de funciones	**45 horas**	**103 horas**
Obtención de elementos orbitales	7 horas	5 horas
Cálculo del error de predicción	3 horas	2 horas
Optimización de parámetros orbitales	15 horas	60 horas
Representación de órbitas	3 horas	3 horas
Generación de predicciones	10 horas	25 horas
Unificación de los datos	5 horas	5 horas
Normalización de datos	2 horas	3 horas
Obtención de datos	**55 horas**	**84 horas**
Obtención de datos precisos	30 horas	20 horas
Procesado de los datos precisos	5 horas	4 horas
Generación de dataset	20 horas	60 horas
Entrenamiento de modelos	**115 horas**	**52 horas**
Preprocesado de los datos	15 horas	10 horas
Elección de parámetros de las redes	10 horas	2 horas
Entrenamiento de modelos	90 horas	40 horas
Evaluación de resultados	**18 horas**	**20 horas**
Análisis de las métricas	3 horas	2 horas
Desnormalización de resultados	2 horas	1 hora
Corrección de SGDP4	3 horas	2 horas
Evaluación de las correcciones	10 horas	15 horas
Documentación	**87 horas**	**82 horas**
Redacción del estudio teórico	30 horas	35 horas
Redacción del resto de la memoria	50 horas	40 horas
Elaboración de la presentación	7 horas	7 horas
Reuniones	**14 horas**	**26 horas**
Reunión de inicio	1.5 horas	1.5 hora
Reuniones semanales de media hora	22 sem $\rightarrow$ 11 horas	23 horas
Reunión de cierre	1.5 hora	1.5 hora
Total	**450 horas**	**481 horas**

Cuadro 2.1: Tareas y duración

Las reuniones serán con los tutores, Daniel Ayala Hernández y Luis Valencia Cabrera, para tareas de evaluación y asesoramiento. También se prevé que en algunas de ellas esté presente Rafael Ayala Hernández, autor del paquete *asteRisk*, como experto en la materia.

Como puede observarse en la tabla, la planificación temporal real se ha desviado de la prevista en un total de 31 horas. Esto ha sido causado, principalmente, por las complicaciones en las tareas de optimización de los parámetros orbitales y en las de generación del conjunto de datos.

La duración de los *sprints scrum* que se llevaron a cabo se muestra en la siguiente tabla:

Iteración	Inicio - Fin	Dedicación
Artículos y fundamentos matemáticos	18/11/2022 - 10/12/2022	49 horas
Propagación de órbitas	11/12/2022 - 09/01/2023	65 horas
Redacción del estudio teórico	10/01/2023 - 21/01/2023	35 horas
Optimización de parámetros	22/01/2023 - 21/02/2023	71 horas
Obtención de datos	22/02/2023 - 04/03/2023	31 horas
Generación del dataset	05/03/2023 - 02/04/2023	111 horas
Entrenamiento y evaluación de modelos	03/04/2023 - 03/05/2023	72 horas
Finalización de la memoria	04/05/2023 - 09/06/2023	40 horas
Elaboración de la presentación	10/06/2023 - 16/06/2023	7 horas

Cuadro 2.2: Iteraciones Scrum

2.4. Presupuesto

Para el cálculo de costes del proyecto se tendrán en cuenta varias partidas que irán destinadas a cubrir los gastos del personal, la amortización del equipo informático utilizado y el coste de la electricidad y la conexión a Internet.

Los costes de personal los vamos a estimar para un desarrollador contratado a jornada completa que perciba una retribución bruta anual de 21.840 euros, lo que supone un coste mensual de 12 euros por hora trabajada.

Dentro de los costes del equipo informático hay que considerar la amortización del equipo empleado durante los 8 meses de duración del proyecto, con un coste inicial de 1245 euros, cuya vida útil se estima en cuatro años (48 meses), de los cuales ocho meses serán de uso para el proyecto y con un consumo eléctrico aproximado de 200 W/hora. El coste medio del kilovatio por hora se estima en 0.30 euros y para el suministro de conexión a internet se contratará una tarifa de 30 euros al mes.

Además de los gastos previstos se establece un coste de reservas de riesgos de un 10 por ciento del total para solventar las incidencias que puedan surgir durante la ejecución.

De esta manera los costes previstos ascienden a un total de 6221'88€:

Partida	Cálculo	Total
Coste de personal	12€/ h x 450 horas	5400 €
Amortización del equipo	1245 €/ 48 meses x 8 meses de proyecto	207'5€
Suministro eléctrico	0'2 kWh x 0'30 €x 450 horas	27 €
Conexión a internet	30 €/mes / 720 horas/mes x 450 horas	18'75€
	Total parcial	**5656'25 €**
	Reservas (10 %)	**565'63 €**
	Total	**6221'88 €**

Cuadro 2.3: Costes previstos

Y los costes reales finalmente fueron de:

Partida	Cálculo	Total
Coste de personal	12 €/ h x 481 horas	5572 €
Amortización del equipo	1245 €/ 48 meses x 8 meses de proyecto	207'5€
Suministro eléctrico	0'2 kWh x 0'30 €x 481 horas	28'86 €
Conexión a internet	30 €/mes / 720 horas/mes x 481 horas	20'04 €
	Gastos finales	**5828'4 €**
	Total previsto con reservas	**6221'88 €**
	Superávit en costes	**393'48 €**

Cuadro 2.4: Costes reales y desviación económica

Se concluye, por tanto, que se ha producido una desviación de costes de 172'15 euros con respecto a los gastos previstos sin la reserva, lo que representa un 3'04 % de desviación entre dichos costes. Gracias a las reservas, se han podido cubrir dichos gastos y finalmente hay un superávit de 393'48 euros, lo que representa el 6'32 % del presupuesto total.

3. Estudio Teórico

En este capítulo se desarrolla un estudio teórico en el que se introduce la teoría de órbitas y las bases de los principales modelos de propagación de trayectorias. Dicho estudio está basado en [1], [9] y [15].

3.1. Conceptos previos

En esta sección se van a recordar y definir conceptos importantes de cara a la comprensión del estudio teórico y desarrollo del proyecto.

3.1.1. Geometría de la elipse

La comprensión de esta geometría resulta fundamental en este trabajo debido a que las órbitas que describen los satélites alrededor de la Tierra son elipses.

Una **elipse** es el lugar geométrico de los puntos de un plano, tales que la suma de sus distancias a dos puntos fijos, llamados **focos**, siempre es constante. A esta longitud constante se le denomina eje mayor. La mitad de esta longitud se denomina **semieje mayor** y se denota por a.

Al segmento de recta perpendicular al eje mayor cuyos extremos se localizan sobre la elipse se denomina eje menor. Su valor es necesario como dato para poder obtener la ecuación de la elipse. Se denota por b a la mitad de dicho eje, conocido como **semieje menor**.

La **excentricidad**, e, es una cantidad constante para cada elipse, y da una medida de cómo de "achatada" es dicha elipse. Se calcula dividiendo la **semidistancia focal** (de foco a centro), c, entre la longitud del semieje mayor:

$$e = \frac{c}{a} = \frac{\sqrt{a^2 - b^2}}{a}$$

Algunos autores prefieren utilizar el **achatamiento**, en lugar de la excentricidad, para describir la forma. Su fórmula es:

$$f = \frac{a - b}{a}$$

La relación entre ambos parámetros viene dada por la expresión:

$$e^2 = 2f - f^2$$

También se puede definir la elipse como el lugar geométrico de los puntos de un plano para los cuales el cociente entre sus distancias a un punto fijo, **foco**, y a una

recta dada, **directriz**, permanece constante y es igual a la excentricidad de la misma. Cada foco tiene asociada una directriz.

La ecuación general de la elipse es:

$$Ax^2 + By^2 + Cx + Dy + F = 0$$

con $A \neq B$ y del mismo signo.

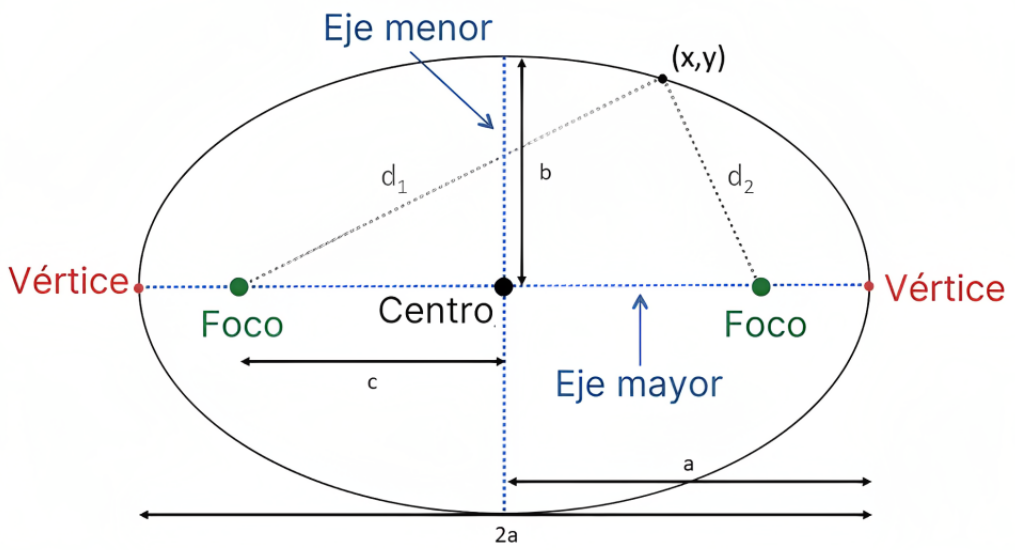

Figura 3.1: Esquema de una elipse

3.1.2. Algunas definiciones

A continuación se definen ciertos conceptos cuyo conocimiento es necesario para una comprensión adecuada de este estudio.

- **Época**: instante de tiempo que indica un momento en el que se conocen la posición y la velocidad del satélite.
- **Perturbación**: modificación que experimenta el movimiento de un astro a lo largo de su órbita como consecuencia de la atracción ejercida por los astros próximos.
- **Órbita osculante**: órbita de Kepler gravitacional (es decir, una elíptica u otra cónica) que tendría el objeto en el espacio alrededor de su cuerpo central si no hubiera perturbaciones.
- **Anomalía media**: ángulo que forma con el eje de la elipse un satélite ficticio que gira con movimiento uniforme sobre una circunferencia cuyo diámetro coincide con el eje principal de la elipse y llamada circunferencia principal.
- **Anomalía excéntrica**: ángulo medido desde el centro de la elipse, que forma la proyección del satélite sobre la circunferencia principal, y el eje de la elipse.

- **Libración lunar**: conjunto de movimientos de oscilación que presenta el disco de la Luna con respecto a un observador ubicado en la Tierra.
- **Ascensión recta**, α: coordenada astronómica utilizada para localizar los astros sobre la esfera celeste. Es equivalente a la longitud terrestre (coordenada geográfica).
- **Declinación**, δ: ángulo que forma un astro con el ecuador celeste. Es equivalente a la latitud.
- **Acimut**: ángulo de una dirección medido en sentido horario a partir del norte geográfico.
- **Apoastro, apoapsis, apoápside, o apocentro**: punto de una órbita elíptica más alejado de su centro gravitatorio. En el caso del Sol se conoce como afelio, de la Tierra como apogeo y de la Luna como aposelenio.
- **Periastro, periapsis, periápside, o pericentro**: punto de una órbita elíptica más cercano a su centro gravitatorio. En el caso del Sol se conoce como perihelio, de la Tierra como perigeo y de la Luna como periselenio.

3.1.3. Leyes de Kepler

Estas leyes describen el movimiento de los cuerpos celestes en el espacio. Aunque originalmente formuladas en torno a los planetas y el Sol, sirven para cualquier cuerpo que orbite alrededor de otro.

Primera Ley de Kepler (1609)

«*Todos los planetas se desplazan alrededor del Sol describiendo órbitas elípticas. El Sol se encuentra en uno de los focos de la elipse.*»

Segunda Ley de Kepler (1609)

«*El radio vector que une un planeta y el Sol recorre áreas iguales en tiempos iguales.*»

Tercera Ley de Kepler (1619)

«*Para cualquier planeta, el cuadrado de su período orbital es directamente proporcional al cubo de la longitud del semieje mayor de su órbita elíptica.*»

$$\frac{T^2}{a^3} = \frac{4\pi^2}{GM}$$

donde G es la constante de Gravitación Universal y M es la masa del planeta.

3.1.4. Ecuación de Kepler

La consecuencia de la segunda ley de Kepler es que el cuerpo no se mueve con un movimiento uniforme, lo cuál dificulta el cálculo de la posición del planeta dentro de su órbita. Por ello aparece la llamada ecuación de Kepler, que nos permite determinar la posición exacta en un instante de tiempo determinado, relacionando la anomalía media, M, de cálculo inmediato, con la anomalía excéntrica, E, cuyo cálculo no es tan directo y que es necesario en los modelos de propagación SGP4 y SDP4. A continuación se deduce dicha ecuación.

Consideremos la órbita definida por una elipse de semieje mayor a y menor b. La masa central, el Sol (F), en torno a la cual orbita la Tierra (Q) se sitúa, de acuerdo con la primera ley de Kepler, en uno de los focos de la elipse, F. Consideremos también la circunferencia circunscrita a dicha elipse, cuyo radio es a y tal que, si el planeta recorre la elipse completa en un período T, un planeta ficticio recorrerá la circunferencia completa en el mismo tiempo.

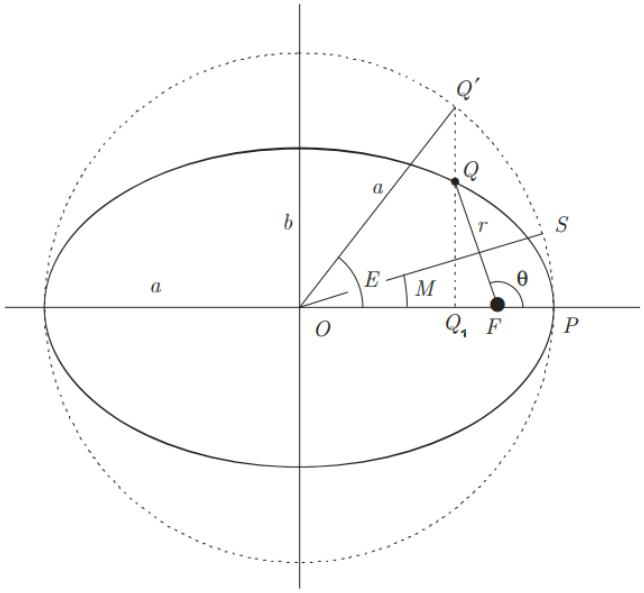

Figura 3.2: Situación de una órbita

Sea el perihelio, P, el punto de partida del movimiento elíptico de la Tierra. El área barrida en un instante t será la superficie delimitada por PFQ. Vamos a relacionar este área con la barrida desde F por un planeta ficticio de referencia, Q', partiendo de P en su movimiento circular uniforme. Dicha área será la limitada por PCS. Por la proporcionalidad de áreas entre elipse y circunferencia sabemos que:

$$PQ_1Q' = PQ_1Q\frac{a}{b}$$

y

$$FQ_1Q' = FQ_1Q\frac{a}{b}$$

Por tanto,

$$PQ_1Q - FQ_1Q' = (PQ_1Q - FQ_1Q)\frac{a}{b}$$

y, así,

$$PFQ' = PFQ\frac{a}{b}$$

Si el período orbital es T, en un tiempo t el área barrida en la elipse será:

$$PFQ = \frac{t}{T}\pi ab,$$

por lo que en la circunferencia será

$$PFQ' = \frac{t}{T}\pi a^2 (I)$$

Por otra parte,

$$PFQ' = PCQ' - FCQ'$$

y

$$PCQ' = \frac{E}{2\pi}\pi a^2 = \frac{E}{2}a^2,$$

donde E está en radianes.

Además,

$$FCQ' = \frac{1}{2}ac \operatorname{sen} E = \frac{1}{2}a^2 e \operatorname{sen} E$$

Luego

$$PFQ' = \frac{E}{2}a^2 - \frac{1}{2}a^2 e \operatorname{sen} E (II)$$

Igualando (I) y (II) llegamos a

$$\frac{t}{T}2\pi = E - e \operatorname{sen} E$$

y, como $M = \frac{t}{T}2\pi$ se tiene

$$M = E - e \operatorname{sen} E,$$

que es la denominada **ecuación de Kepler**.

3.1.5. Constelaciones de satélites

Hay diversas formas de agrupar los satélites, en este trabajo nos interesa mediante constelaciones, que son conjuntos de satélites que trabajan como un único sistema, proporcionando una cobertura global o cuasiglobal en todo momento. Entre ellas destacamos:

- GPS (Global Positioning System), de Estados Unidos.

- GLONASS, desarrollado por la Unión Soviética y administrado hoy en día por Rusia.
- EGNOS (European Geostationary Navigation Overlay Service), de la Unión Europea.
- WASS (Wide Area Augmentation System) de Estados Unidos
- Galileo, de la Unión Europea.
- BeiDou, de China.

La importancia de la clasificación según su constelación se comprenderá mejor posteriormente, cuando se expongan los formatos de intercambio de archivos.

3.2. Sistemas de referencia de coordenadas

Antes de abordar el estudio de los modelos de propagación conviene definir alguno de los sistemas de coordenadas que van a ser utilizados. Nos centraremos en sistemas de referencia geocéntricos, puesto que los satélites que vamos a estudiar orbitan alrededor de la Tierra, inerciales, es decir, sin aceleraciones del sistema. Trabajaremos con sistemas rectangulares o cartesianos, para los que además del origen hay que indicar el plano fundamental y su vector normal o un sistema ortonormal de vectores.

La Tierra y su órbita alrededor del Sol forman la base de los sistemas de coordenadas celestes que se van a considerar en el trabajo. Algunos de los más destacados son:

- **TEME (True Equator, Mean Equinox)**: es un sistema geocéntrico inercial. Resulta complicado encontrar una definición exacta de este sistema en la literatura, pero podemos definirlo como aquel en el cual el eje Z coincide con el eje de rotación de la Tierra, el eje X depende del equinocio vernal medio y el eje Y es el producto de los ejes X y Z. Por ejemplo, NORAD lo utiliza en sus archivos TLE.
- **Sistema de Referencia Geocéntrico Celestial**, **GCRF**: es un marco de coordenadas inercial centrado en la Tierra, donde el origen se coloca en el centro de masa de la Tierra y el marco de coordenadas se fija con respecto a las estrellas, no respecto a la superficie del planeta. El eje X está alineado con el equinocio medio de la Tierra a las 12:00 hora terrestre del 1 de enero del año 2000, y el eje Z está alineado con el eje de rotación de la Tierra. Es casi equivalente al marco de referencia J2000 (también llamado EME2000), y en algunos contextos también se le conoce como marco **ICRF** (aunque en su definición estricta, el origen de coordenadas de éste último se ubica en el baricentro del Sistema Solar).
- **Sistema de Referencia Terrestre Internacional**, **ITRF**: al igual que el GCRF, es un marco de coordenadas inercial centrado en la Tierra y con el origen en el centro de masa de la Tierra. Sin embargo, en este caso el marco gira con respecto a las estrellas, para permanecer fijo con respecto a la superficie de la Tierra a medida que ésta gira. El eje Z se extiende a lo largo del norte verdadero definido por el polo de referencia IERS, y el eje X se extiende hacia la intersección

entre el ecuador y el meridiano de Greenwich en cualquier momento. También se conoce como **ECEF** (Earth-Centered, Earth-Fixed).

Cuando se estén realizando operaciones con posiciones y velocidades hay que poner especial atención en que el sistema de referencia sea el adecuado.

3.3. Sistemas de tiempo

El propósito del tiempo es definir con precisión el momento de un determinado fenómeno. Por lo tanto, los sistemas de tiempo son el equivalente a los sistemas de referencia pero en el aspecto temporal: fijado un instante determinado, ya podemos medir tiempos relativos a partir de él.

En el libro "Principios Matemáticos de la filosofía natural"(1687), Isaac Newton describe dos tipos de tiempo: absoluto, que no puede ser alterado, y relativo, dependiente de un punto prefijado en la línea temporal.

El sistema más utilizado es el Tiempo Universal Coordinado, **UTC**. Surgió en 1962, fruto de la cooperación internacional en un esfuerzo por crear un formato de tiempo estándar y consistente, ya que los que se utilizaban sufrían retrasos debido a las variaciones de la rotación terrestre. El sistema UTC fue finalmente adoptado en 1972.

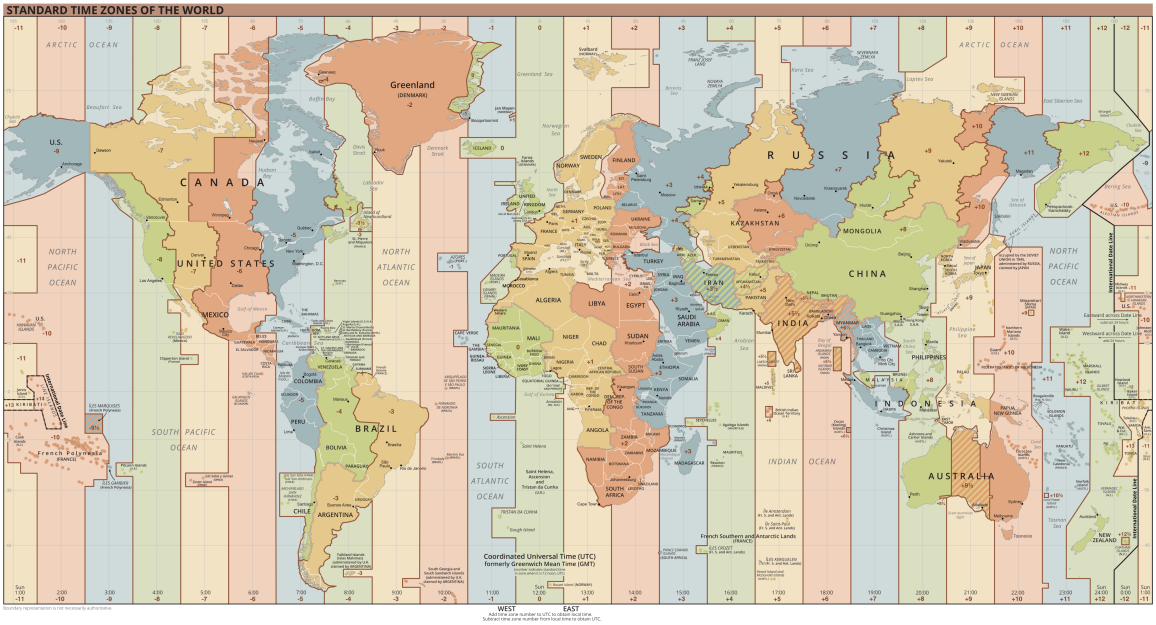

Figura 3.3: Zonas UTC

Los satélites suelen utilizar el sistema de **tiempo GPS** que, como su nombre indica, se introdujo con el sistema de navegación que tiene el mismo nombre. La época (punto de referencia) GPS es el 6 de enero de 1980 a las 0 horas UTC. Debido a la rotación de la Tierra este sistema difiere ligeramente del UTC, por lo que debe ser reajustado diariamente.

En astrodinámica es muy importante la Fecha Juliana o Día Juliano, **JD**, que es el intervalo de tiempo, medido en días, desde el 1 de enero de 4713 A.C. a las 12:00. Esta fecha fue determinada por Joseph Scaliger en 1582.

Si queremos calcular la fecha JD para un año entre el 1900 y el 2100 se puede utilizar la siguiente fórmula:

$$JD = 367(yr) - \left\lfloor \frac{7\left(yr + \left\lfloor \frac{mo+9}{12} \right\rfloor\right)}{4} \right\rfloor + \left\lfloor \frac{275mo}{9} \right\rfloor + d + 1721013{,}5 + \frac{\frac{s}{60} + min}{60} + h}{24}$$

Estos valores son demasiado grandes, lo que puede incitar a aproximar los decimales. Sin embargo, como las unidades son días, no se debe aproximar en exceso. Se recomienda dejar al menos 8 decimales para obtener una precisión del orden de 4×10^{-4} segundos, aunque la aproximación más exacta consiste en separar la fecha JD en parte entera y parte fraccionaria. Para reducir el tamaño de la fecha, la Unión Astronómica Internacional recomienda el uso de la Fecha Juliana Modificada, **MJD**, que se calcula de la siguiente manera:

$$MJD = JD - 2400000{,}5$$

El meteorólogo francés Jean Meeus aportó una fórmula general para el caso de la fecha Juliana, sin restricción alguna. En dicho cálculo consideramos enero y febrero como los meses 13 y 14, respectivamente; así, si mo=1 ó 2, entonces reasignamos $yr = yr - 1$ y $mo = mo + 12$. La fórmula es:

$$JD = \lfloor 365{,}25(yr + 4716) \rfloor + \lfloor 30{,}6001(mo + 1) \rfloor + d + B - 1524{,}5 + C$$

donde

$$B = 2 - \left\lfloor \frac{yr}{100} \right\rfloor + \left\lfloor \frac{\left\lfloor \frac{yr}{100} \right\rfloor}{4} \right\rfloor$$

$$C = \frac{\frac{s}{60} + min}{60} + h}{24}$$

Si se está convirtiendo una fecha anterior a la reforma gregoriana del calendario, es decir, una fecha anterior al 15 de octubre de 1582, se tiene que considerar B=0.

De cara al entrenamiento de las redes neuronales nos interesa poder expresar la fecha como un único número, y que éste no sea excesivamente grande (como ocurre con

la fecha Juliana). Por ello vamos a emplear el **Tiempo Unix** o **Tiempo POSIXct**, que es un número entero que incrementa una unidad cada segundo. El punto de referencia, es decir, el tiempo Unix 0, es el jueves 1 de enero de 1970. El viernes 13 de febrero de 2009, exactamente a las 23:31:30 UTC, el tiempo Unix se igualó a 1234567890.

3.4. Elementos orbitales

También conocidos como KOE (Keplerian Orbital Elements), los elementos orbitales son los parámetros necesarios para identificar de manera biunívoca la órbita de un determinado cuerpo celeste. Consideraremos órbitas de Kepler, es decir, un cuerpo orbita a otro describiendo una elipse, parábola o hipérbola, inscritas en un plano orbital bidimensional en un espacio tridimensional. Al conjunto de estos elementos se le denomina **vector de estado**. Puede variar ligeramente, pero los clásicos, alguno de ellos ya definidos para el caso de la elipse, son:

- **Excentricidad**, e: cuantifica la manera en que la órbita se desvía de una circunferencia.
- **Semieje mayor**, a: distancia entre los cuerpos, no entre su centro de masa.
- **Inclinación**, i: ángulo que forma el plano de la órbita con el de referencia.
- **Longitud del nodo ascendente**, Ω: ángulo desde una dirección de referencia, llamada el origen de la longitud, a la dirección del nodo ascendente, medido en un plano de referencia. El nodo ascendente es el punto donde la órbita del objeto pasa a través del plano de referencia.
- **Argumento del perihelio**, ω: define la orientación de la elipse en el plano orbital, es el ángulo medido desde el nodo ascendente hasta la periapsis (el punto más cercano que el objeto del satélite llega al objeto principal alrededor del cual orbita).
- **Anomalía media de la época**, M_0: es la fracción de un período orbital que ha transcurrido.

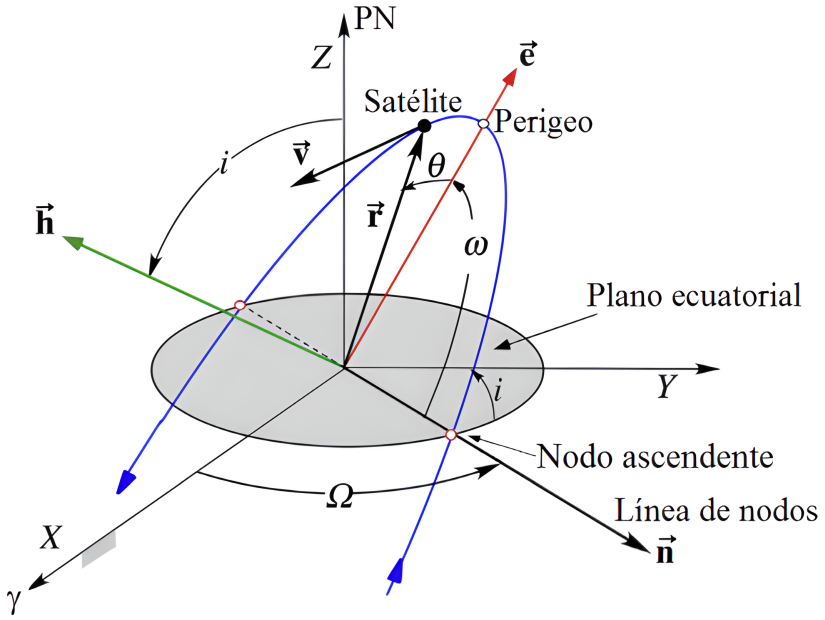

Figura 3.4: Elementos angulares de una órbita

En órbitas ecuatoriales no hay nodo ascendente, por lo que no puede tomarse como referencia para ω; por su parte, en órbitas circulares no hay perigeo, por lo que no puede tomarse como referencia para θ. Por eso en algunas ocasiones (órbitas circulares, ecuatoriales o ambas cosas) se utilizan elementos alternativos:

- Longitud del periapsis: $\bar{\omega} = \Omega + \omega$
- Argumento de latitud: $u = \omega + \theta$
- Longitud verdadera: $L = \Omega + \omega + \theta$

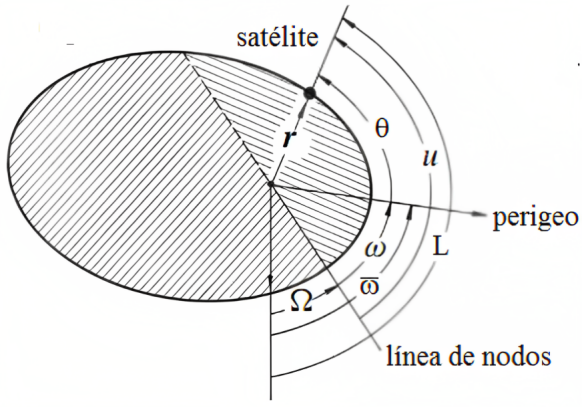

Figura 3.5: Elementos orbitales alternativos

3.5. Formatos de intercambio de datos de posiciones

Gracias a los formatos estandarizados se puede compartir de manera consistente y precisa la posición y otros parámetros relevantes de los satélites. De todos los formatos existentes vamos a destacar los dos utilizados en el desarrollo del proyecto, que se describen a continuación.

3.5.1. TLE (Two-Line Element set)

Codifica una lista de elementos orbitales (vector de estado) de un determinado objeto de la órbita terrestre en una época concreta. Estos elementos pueden ser utilizados para estimar el estado de un satélite en cualquier instante temporal, por ejemplo, utilizando un modelo de perturbaciones simplificadas.

Originalmente fue diseñado para su uso con tarjetas perforadas, aunque se sustituyó por los archivos de texto una vez estas quedaron obsoletas. La Fuerza Espacial de los Estados Unidos (USSF) rastrea todos los satélites y publica los TLEs correspondientes en Celestrak [2].

3.5.2. RINEX (Receiver INdependent EXchange)

Se presentó en el 5º Simposio Geodésico Internacional sobre Posicionamiento Satelital en Marzo de 1989 en Las Cruces, Nuevo México. Su objetivo es almacenar e intercambiar una serie de datos proporcionados por los receptores de los satélites, entre ellos posición y velocidad, de manera estandarizada.

Este formato ha ido evolucionando con el tiempo y dispone de múltiples versiones, teniendo gran importancia la 2.10, compatible con satélites pertenecientes a constelaciones introducidas en el apartado 3.2.4 como GPS, GLONASS, EGNOS y WASS. Actualmente se utilizan más las versiones posteriores a la 3.0, que son compatibles también con las constelaciones Galileo y Beidou.

Tal y como se presentó inicialmente, este formato define tres tipos de archivos:

- **Observación**: contiene información de los observables para cada satélite y constelación, así como las características del receptor y antena empleados o intervalo de tiempo de los datos capturados, entre otros.
- **Navegación**: contiene los datos de orbitales, los parámetros del reloj y la precisión de las medidas de pseudodistancia de los satélites observados.
- **Meteorológicos**: tiene como función simplificar la exportación y procesamiento de datos meteorológicos recogidos por los observatorios.

Todos los archivos se componen de una cabecera con información relacionada con el satélite, el receptor, etc., y un cuerpo con los datos. Para la correcta interpretación de los datos debemos combinar la información de ambas partes. Por ejemplo, en la cabecera se encuentran correcciones como el sesgo del reloj que permite corregir la

época. De especial interés serán los ficheros de navegación, puesto que son los que se utilizaron para la obtención de datos durante el proyecto.

Es importante mencionar que la información contenida en estos archivos varía ligeramente según la constelación, por lo que su lectura y procesado también.

3.6. Modelos de perturbaciones simplificadas

Forman un conjunto de modelos matemáticos que predicen el efecto de las perturbaciones en la trayectoria de los satélites, causadas por la forma de la Tierra, el rozamiento, la radiación solar y los efectos gravitacionales de cuerpos celestes como el Sol y la Luna. Así, nos permiten calcular vectores de estado orbitales, que utilizaremos para obtener la posición y velocidad de un determinado satélite. Dichos modelos son **SGP**, **SGP4** y **SGP8** para objetos cercanos a la Tierra (periodo orbital inferior a 225 minutos, o lo que es lo mismo, altitud inferior a 5.877,5 km, suponiendo órbita circular), y **SDP4** y **SDP8** para objetos lejanos (del espacio profundo). Nos centraremos en el estudio y descripción de los modelos SGP4 y SDP4, que son los más utilizados.

El modelo SGP4 fue desarrollado por Ken Cranford en 1970, fruto de una simplificación de la teoría analítica de Lane y Cranford (1969), que utiliza la solución de Brouwer para su modelo gravitacional.

El modelo SDP4 es una extensión del SGP4 para satélites del espacio profundo. Las ecuaciones del espacio profundo fueron introducidas por Hujsak en 1979 y modelizan los efectos gravitacionales del Sol y la Luna utilizando armónicos terrestres sectoriales y teserales, que tienen especial importancia para órbitas con periodo de medio día o de un día.

Los esféricos armónicos nos permiten representar la parte no-rotacional del potencial gravitatorio terrestre, que cumple la ecuación de Laplace ($\nabla^2 U = 0$), de la siguiente manera:

$$U(r,\theta,\phi) = \sum_{l=0}^{\infty}\sum_{m=0}^{l}\left(\frac{1}{r}\right)^{l+1}\Big[A_l^m\cos(m\phi) + B_l^m\sin(m\phi)\Big]P_l^m(\cos\theta)$$

donde $P_l^m(\cos\theta)$ son los polinomios de Legendre y A_l^m y B_l^m los coeficientes de la expansión armónica.

En el marco rotacional, es decir, si consideramos la rotación de la Tierra, hay que añadir al potencial el término

$$-\frac{1}{2}r^2\sin^2\theta\omega^2$$

Distinguimos tres tipos de armónicos:

- Zonales: en ellos $m = 0$
- Sectoriales: son aquellos en los que $l = m, m \neq 0$

- Teserales: cumplen $l \neq m, m \neq 0$

Tras esta aclaración sobre los esféricos armónicos pasamos a la explicación del modelo SDP4, según aparece en [5], que es muy parecido al SGP4, pero añadiendo una corrección de espacio profundo. Las ecuaciones de perturbación son bastante extensas y no van a ser descritas en este trabajo, pero pueden ser consultadas en [6]. Primero definamos los parámetros y las constantes que necesita el modelo:

- n_0 = movimiento medio del satélite, en radianes/min
- e_0 = excentricidad media de la órbita. Toma valores entre 0 (órbita circular perfecta) y 1 (trayectoria parabólica)
- i_0 = inclinación orbital media, en radianes
- M_0 = anomalía media
- ω_0 = argumento medio del periastro. Para objetos que orbitan el Sol, se llama argumento del perihelio y para objetos que orbitan la Tierra, argumento del perigeo
- Ω_0 = longitud media del nodo
- B^* = coeficiente de rozamiento en unidades de $(radio\ de\ la\ Tierra)^{-1} = er^{-1}$. Es un ajuste del coeficiente balístico del satélite, e indica cómo de susceptible es al rozamiento atmosférico. Puede ser tanto positivo como negativo
- $k_e = \sqrt{GM}$, donde G es la constante de gravitación universal y M es la masa de la Tierra
- a_E = radio ecuatorial de la Tierra. Vale 1 er
- J_2 = segundo armónico gravitacional zonal de la Tierra. Su valor es de 1.082616e-03
- J_3 = tercer armónico gravitacional zonal de la Tierra. Su valor es de -2.53881e-06
- J_4 = cuarto armónico gravitacional zonal de la Tierra. Su valor es de -1.65597e-06
- t_0 = tiempo de la época
- t = tiempo en el que queremos realizar la predicción
- $(t - t_0)$ = tiempo pasado desde la época
- $k_2 = \frac{1}{2} J_2 a_E^2$
- $k_4 = -\frac{3}{8} J_4 a_E^4$
- $A_{3,0} = -J_3 a_E^3$
- q_0 = parámetro para la función de densidad cuyo valor es 1.018814
- s = parámetro para la función de densidad cuyo valor es 1.012229
- $XKMPER$ = kilómetros por unidad de er. Su valor es de 6378,135, que coincide con el radio de la Tierra en el ecuador

Ahora pasamos al modelo. Primero recuperamos el movimiento medio (n_0'') y el semieje mayor (a_0'') originales:

$$a_1 = \left(\frac{k_e}{n_0}\right)^{\frac{2}{3}}$$

$$\delta_1 = \frac{3}{2}\frac{k_2}{a_1^2}\frac{3\cos^2 i_0 - 1}{(1-e_0^2)^{\frac{3}{2}}}$$

$$a_0 = a_1\left(1 - \frac{1}{3}\delta_1 - \delta_1^2 - \frac{134}{81}\delta_1^3\right)$$

$$\delta_0 = \frac{3}{2}\frac{k_2}{a_0^2}\frac{3\cos^2 i_0 - 1}{(1-e_0^2)^{\frac{3}{2}}}$$

$$n_0'' = \frac{n_0}{1+\delta_0}$$

$$a_0'' = \frac{a_0}{1-\delta_0}$$

Si el perigeo de la órbita se encuentra entre 98 y 156 kilómetros, el valor de la constante s debe reemplazarse por:

$$s^* = a_0''(1-e_0) - s + a_E$$

Para perigeo inferior a 98 kilómetros, el valor de s se debe cambiar por:

$$s^* = \frac{20}{XKMPER} + a_E$$

Ahora pasamos a calcular algunas constantes, utilizando los valores apropiados de s:

$$\theta = \cos i_0$$

$$\xi = \frac{1}{a_0'' - s}$$

$$\beta_0 = (1-e_0^2)^{\frac{1}{2}}$$

$$\eta = a_0'' e_0 \xi$$

$$C_2 = (q_0 - s)^4 \xi^4 n_0''(1-\eta^2)^{-\frac{7}{2}}\left[a_0''\left(1 + \frac{3}{2}\eta^2 + 4e_0\eta + e_0\eta^3\right) + \frac{3}{2}\frac{k_2\xi}{(1-\eta^2)}\left(-\frac{1}{2} + \frac{3}{2}\theta^2\right)(8 + 24\eta^2 + 3\eta^4)\right]$$

$$C_1 = B * C_2$$

$$C_4 = 2n_0''(q_0 - s)^4\xi^4 a_0''\beta^2(1-\eta^2)^{-\frac{7}{2}}\left(\left[2\eta(1+e_0\eta)\frac{1}{2}e_0 + \frac{1}{2}\eta^3\right] - \frac{2k_2\xi}{a_0''(1-\eta^2)}\times \left[3(1-3\theta^2)\left(1 + \frac{3}{2}\eta^2 - 2e_0\eta^3\right) + \frac{3}{4}(1-\theta^2)(2\eta^2 - e_0\eta - e_0\eta^3)\cos 2\omega_0\right]\right)$$

$$\dot{M} = \left[1 + \frac{3k_2(-1+3\theta^2)}{2a_0''^2\beta_0^3} + \frac{3k_2^2(13 - 78\theta^2 + 137\theta^4)}{16a_0''^4\beta_0^7}\right]n_0''$$

$$\dot{\omega} = \left[-\frac{3k_2(-1-5\theta^2)}{2a_0''^2\beta_0^4} + \frac{3k_2^2(7-114\theta^2+395\theta^4)}{16a_0''^4\beta_0^8} + \frac{5k_2(3-36\theta^2+49\theta^4)}{4a_0''^4\beta_0^8} \right] n_0''$$

$$\dot{\Omega}_1 = -\frac{3k_2\theta}{a_0''^4\beta_0^4} n_0''$$

$$\dot{\Omega} = \dot{\Omega}_1 + \left[\frac{3k_2^2(4\theta-19\theta^3}{2a_0''^4\beta_0^8} + \frac{5k_4\theta(3-7\theta^2)}{2a_0''^4\beta_0^8} \right] n_0''$$

En este momento se realiza la inicialización de los parámetros que sirven para calcular las ecuaciones de perturbación del espacio profundo.

Los efectos de la gravedad se incluyen en:

$$M_{DF} = M_0 + \dot{M}(t - t_0)$$

$$\omega_{DF} = \omega_0 + \dot{\omega}(t - t_0)$$

$$\Omega_{DF} = \Omega_0 + \dot{\Omega}(t - t_0)$$

El efecto del rozamiento en la longitud del nodo ascendente viene determinado por:

$$\Omega = \Omega_{DF} - \frac{21}{2} \frac{n_0'' k_2 \theta}{a_0''^2 \beta_0^2} C_1 (t - t_0)^2$$

A continuación habría que añadir los efectos del espacio profundo y de la resonancia de largo periodo a los seis elementos orbitales clásicos:

$$a = a_{DS}[1 - C_1(t - t_0)]^2$$

$$e = e_{DS} - B^* C_4 (t - t_0)$$

$$\mathbb{L} = M_{DS} + \omega_{DS} + \Omega_{DS} + n_0'' \left[\frac{3}{2} C_1 (t - t_0)^2 \right]$$

donde a_{DS}, e_{DS}, M_{DS}, ω_{DS} y Ω_{DS} son los valores de n_0, e_0, M_{DF}, ω_{DF} y ΩDF después de aplicar las perturbaciones de a resonancia y el espacio profundo.

Ahora hay que incluir las perturbaciones de espacio profundo debidas al Sol y la Luna. A partir de aquí, denotaremos por n, e, I, ω, Ω y M al movimiento medio, la excentricidad, la inclinación, el argumento de perigeo, la longitud del nodo ascendente y la anomalía media, respectivamente, después de añadir los periodos solares y lunares.

$$a_{xN} = e \cos \omega$$

$$\beta = \sqrt{1 - e^2}$$

$$\mathbb{L}_L = \frac{A_{3,0} \sin i_0}{8 k_2 a \beta^2} e \cos \omega \left(\frac{3 + 5\theta}{1 + \theta} \right)$$

$$a_{yNL} = \frac{A_{3,0} \sin i_0}{4 k_2 a \beta^2}$$

$$\mathbb{L}_T = \mathbb{L} + \mathbb{L}_L$$

$$a_{yN} = e \sin \omega + a_{yNL}$$

Resolvemos la ecuación de Kepler para $E+\omega$ definiendo

$$U = \mathbb{L}_T - \Omega$$

y aplicando el método de aproximaciones sucesivas:

$$(E+\omega)_{i+1} = (E+\omega)_i + \Delta(E+\omega)_i$$

donde

$$\Delta(E+\omega)_{i+1} = \frac{U - a_{yN}\cos(E+\omega)_i + a_{xN}\sin(E+\omega)_i - (E+\omega)_i}{-a_{yN}\sin(E+\omega)_i - a_{xN}\cos(E+\omega)_i + 1}$$

y

$$(E+\omega)_1 = U$$

Las siguientes ecuaciones se utilizan para calcular valores que necesitaremos con posterioridad.

$$e\cos E = a_{xN}\cos(E+\omega + a_{yN}\sin(E+\omega)$$

$$e\sin E = a_{xN}\sin(E+\omega + a_{yN}\cos(E+\omega)$$

$$e_L = (a_{xN}^2 + a_{yN}^2)^{\frac{1}{2}}$$

$$p_L = a(1-e_L^2)$$

$$r = a(1-e\cos E)$$

$$\dot{r} = k_e\frac{\sqrt{a}}{r}e\sin E$$

$$r\dot{f} = k_e\frac{\sqrt{p_L}}{r}$$

$$\cos u = \frac{a}{r}\left[\cos(E+\omega) - a_{xN} + \frac{a_{yN}(e\sin E)}{1+\sqrt{1-e_L^2}}\right]$$

$$\sin u = \frac{a}{r}\left[\sin(E+\omega) - a_{yN} + \frac{a_{xN}(e\sin E)}{1+\sqrt{1-e_L^2}}\right]$$

$$u = \tan^{-1}\left(\frac{\sin u}{\cos u}\right)$$

$$\Delta r = \frac{k_2}{2p_L}(1-\theta^2)\cos 2u$$

$$\Delta u = -\frac{k_2}{4{p_L}^2}(7\theta^2-1)\sin 2u$$

$$\Delta\Omega = \frac{3k_2\theta}{2{p_L}^2}\sin 2u$$

$$\Delta i = \frac{3k_2\theta}{2{p_L}^2}\sin i_0\cos 2u$$

$$\Delta\dot{r} = -\frac{k_2 n}{p_L}(1-\theta^2)\sin 2u$$

$$\Delta r\dot{f} = \frac{k_2 n}{p_L}\left[(1-\theta^2)\cos 2u - \frac{3}{2}(1-3\theta^2)\right]$$

Añadimos estos incrementos para obtener las cantidades osculantes:

$$r_k = r\left[1 - \frac{3}{2}k_2\frac{\sqrt{1-e_L^2}}{{p_L}^2}(3\theta^2-1)\right] + \Delta r$$

$$u_k = u + \Delta u$$

$$\Omega_k = \Omega + \Delta\Omega$$

$$i_k = I + \Delta i$$

$$\dot{r}_k = \dot{r} + \Delta\dot{r}$$

$$r\dot{f}_k = r\dot{f} + \Delta r\dot{f}$$

Los vectores unitarios de orientación se calculan con

$$\vec{U} = \vec{M}\sin u_k + \vec{N}\cos u_k$$

$$\vec{V} = \vec{M}\cos u_k + \vec{N}\sin u_k$$

donde

$$\vec{M} = \begin{pmatrix} -\sin\Omega_k\cos i_k \\ \cos\Omega_k\cos i_k \\ \sin i_k \end{pmatrix}$$

y

$$\vec{N} = \begin{pmatrix} \cos\Omega_k \\ \sin\Omega_k \\ 0 \end{pmatrix}$$

Finalmente, la posición $\vec{r}$ y la velocidad $\dot{\vec{r}}$ se obtienen como

$$\vec{r} = r_k\vec{U}$$

y

$$\dot{\vec{r}} = \dot{r}_k\vec{U} + r\dot{f}_k\vec{V}$$

3.7. Modelos de propagación de alta precisión

Johann Franz Encke (1791-1865) presentó un método de cálculo de órbitas para el caso en el que las perturbaciones fueran pequeñas con respecto al movimiento de los cuerpos. Dicho método se basaba en integrar numéricamente las diferencias entre

la órbita osculante y la órbita real. De este modo se logró suficiente precisión para las limitadas capacidades computacionales de la época.

Philip Herbert Cowell (1870–1949) utilizó una técnica numérica para determinar la órbita del octavo satélite de Júpiter. Con el aumento de la potencia computacional, su técnica ganó cada vez más popularidad hasta el punto de que hoy en día, en astrodinámica, se utiliza la **formulación de Cowell** para establecer las ecuaciones del movimiento, para posteriormente realizar integración numérica.

3.7.1. Formulación de Encke

Gracias al aumento de la capacidad computacional, este método está en desuso, pero debido a su importancia histórica merece una mención. No se basa en integrar todas las fuerzas que actúan sobre el satélite, sino en integrar la diferencia entre la aceleración debida al astro que está en el foco y la aceleración perturbada, siempre y cuando esta sea pequeña. Así se consigue una precisión buena sin mucho coste.

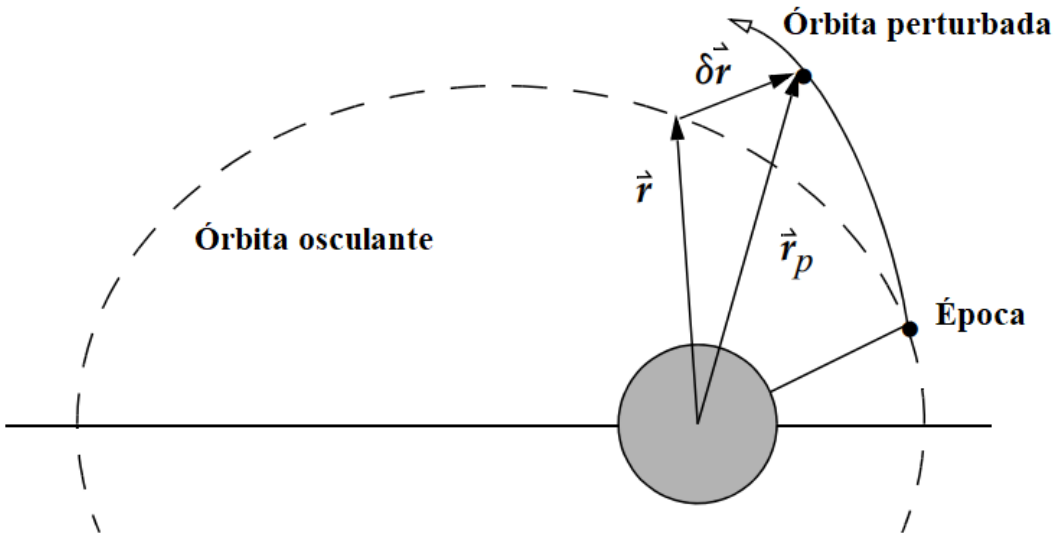

Figura 3.6: Formulación de Encke

La aceleración y la aceleración perturbada son:

$$\ddot{\vec{r}} = -\frac{\mu}{r^3}\vec{r}$$

$$\ddot{\vec{r}}_p = -\frac{\mu}{r_p^3}\vec{r}_p + \vec{a}_p$$

donde $\mu = GM$, con

- $G \approx 6{,}67 \times 10^{-11} \frac{\mathrm{N{\cdot}m^2}}{kg}$ es la constante de gravitación universal
- M es la masa de la Tierra

Por tanto, la diferencia entre las aceleraciones es:

$$\delta\ddot{\vec{r}} = \vec{a}_p + \frac{\mu}{r^3}\left[\left(1 - \frac{r^3}{r_p^3}\right)\vec{r}_p - \delta\vec{r}\right]$$

3.7.2. Formulación de Cowell

La idea detrás de este método es simple pero muy potente, gracias al aumento de la capacidad computacional. Se basa en incorporar aceleraciones a la ecuación del movimiento de dos cuerpos para obtener una ecuación más precisa sobre el movimiento:

$$\ddot{\vec{r}} = -\frac{\mu}{r^3}\vec{r} + \vec{a}_p$$

donde $\vec{a}_p$ es la aceleración total causada por todas las fuerzas que actúan sobre el satélite. La forma concreta de $\vec{a}_p$ dependerá del número de perturbaciones que estemos considerando. Ahora el problema que se nos presenta es la obtención de los datos necesarios para implementar el modelo que queramos. Por ejemplo, no podemos determinar con exactitud los coeficientes gravitacionales de Plutón (debido a que no conocemos con exactitud su radio ecuatorial), luego si los incluimos en el modelo estaremos realizando pequeñas aproximaciones.

La ecuación anterior se conoce como **formulación de Cowell**. Conviene distinguir entre la formulación de Cowell y el método de Cowell. Este último consiste en aplicar el método de las diferencias finitas para integrar la ecuación.

Normalmente se reformulan las tres ecuaciones de segundo orden del movimiento en un sistema de seis ecuaciones de primer orden, de modo que el problema pueda ser abordado con una gama más amplia de métodos de integración numérica. Dicho sistema se conoce como una variación de la formulación de Cowell:

$$\vec{X} = \begin{bmatrix} \vec{r} \\ \vec{v} \end{bmatrix}$$

$$\dot{\vec{X}} = \begin{bmatrix} \vec{v} \\ -\dfrac{\mu\vec{r}}{r^3} + \vec{a}_p \end{bmatrix}$$

3.7.3. HPOP

El método de propagación de alta precisión HPOP se basa en la formulación de Cowell, teniendo en cuenta una multitud de fuerzas que actúan sobre el satélite. Dicha ecuación quedaría del siguiente modo:

$$\ddot{\vec{r}} = -\frac{\mu}{r^3}\vec{r} + \sum_{cc \in CC} \vec{a}_{cc} + \vec{a}_{drag} + \vec{a}_{srp}$$

donde CC son los cuerpos celestes del Sistema Solar distintos a la Tierra, es decir, el Sol, la Luna y todos los planetas menos la Tierra. A continuación se explican estas perturbaciones adicionales, no sin antes mencionar que las coordenadas y velocidades ingresadas y calculadas con este método están en el marco de referencia GCRF.

Campos gravitatorios

Se consideran los efectos de los campos gravitatorios del Sol, la Luna y el resto de planetas del Sistema Solar. Esta aceleración la podemos expresar como:

$$\vec{a}_{cc} = -\frac{GM}{r^2}\frac{\vec{r}}{\|\vec{r}\|}$$

donde

- G es la constante de Gravitación Universal
- M es la masa del cuerpo celeste
- $\vec{r}$ es el vector posición del satélite respecto del cuerpo celeste

Rozamiento atmosférico

La ecuación básica del rozamiento aerodinámico es:

$$\vec{a}_{drag} = -\frac{1}{2}\frac{c_D A}{m}\rho v_{rel}^2 \frac{\vec{v}_{rel}}{\|\vec{v}_{rel}\|}$$

donde

- c_D es el coeficiente de rozamiento
- ρ es la densidad atmosférica
- A es el área de la sección transversal, es decir, el área de la sección normal al vector velocidad del satélite
- m es la masa del satélite
- $\vec{v}_{rel}$ es la velocidad relativa a la atmósfera

Cabe mencionar que normalmente $\dfrac{m}{c_D A}$ se conoce como coeficiente balístico, **BC**.

Si queremos calcular la velocidad relativa a la atmósfera, sin tener en cuenta el movimiento del viento, podemos utilizar la siguiente expresión:

$$\vec{v}_{rel} = \frac{d\vec{r}}{dt} - \vec{\omega}_{\oplus} \times \vec{r} = \begin{pmatrix} \frac{dx}{dt} + \vec{\omega}_{\oplus} y \\ \frac{dy}{dt} - \vec{\omega}_{\oplus} x \\ \frac{dz}{dt} \end{pmatrix}$$

Si tenemos en cuenta el movimiento del viento la expresión quedaría:

$$\vec{v}_{rel} = \begin{pmatrix} \frac{dx}{dt} + \vec{\omega}_{\oplus} y + v_w[-cos(\alpha)sen(\delta)cos(\beta_w) - sen(\alpha)sen(\beta_w)] \\ \frac{dy}{dt} - \vec{\omega}_{\oplus} x + v_w[-sen(\alpha)sen(\delta)cos(\beta_w) - cos(\alpha)sen(\beta_w)] \\ \frac{dz}{dt} + +v_w[cos(\delta)cos(\beta_w)] \end{pmatrix}$$

donde

- α es la ascensión recta
- δ es la declinación
- $\vec{v}_w$ es la velocidad del viento
- β_w es el acimut

Presión de la radiación solar

La aceleración debida a la radiación solar se puede expresar como:

$$\vec{a}_{srp} = -\frac{p_{srp} c_R A_{\odot}}{m} \frac{\vec{r}_{sat\odot}}{\|\vec{r}_{sat\odot}\|}$$

donde

- p_{srp} es la presión solar
- c_R es la reflectividad
- $A_{\odot}$ es el área de sección transversal expuesta al Sol
- $\vec{r}_{sat\odot}$ es el vector velocidad respecto al Sol

Mareas

Podemos distinguir entre **mareas terrestres**, que pueden ser deformaciones de la Tierra causadas principalmente por la atracción gravitatoria de la Luna y el Sol o por el efecto centrífugo de la rotación, y **mareas oceánicas**, especialmente influenciadas por la Luna. Las expresiones matemáticas de estas perturbaciones se pueden expresar mediante polinomios de Legendre y son muy complejas, por lo que se omiten en el presente trabajo.

3.7.4. Resolución numérica de ecuaciones diferenciales

Los métodos de resolución numérica de ecuaciones diferenciales forman un conjunto de herramientas matemáticas que nos permiten encontrar aproximaciones numéricas a las soluciones de ecuaciones diferenciales. También se conoce como integración numérica aunque este nombre puede confundirse con el cálculo aproximado de integrales definidas. Su importancia radica en que la mayoría de las veces no se puede resolver analíticamente las ecuaciones diferenciales. Para profundizar en el tema se puede consultar [4] y [8].

A continuación se presentan el método de Runge-Kutta de cuarto orden, que es el que utiliza la implementación de HPOP de *asteRisk*, y el método de Runge-Kutta-Felhberg, que supone una mejora del anterior.

Método de Runge-Kutta (de 4º orden)

Posiblemente las fórmulas de Runge-Kuta son los métodos de integración numérica más conocidos, desarrollados a principios del siglo XX por Carl Runge y Wilhen Kutta, siendo aplicaciones de las series de Taylor. A continuación se describirá el método de cuarto orden.

Consideremos la ecuación diferencial de primer orden

$$y' = f(x, y)$$

con la condición inicial

$$y(x_0) = y_0$$

y siendo

Consideremos el siguiente problema de valores iniciales:

$$\begin{cases} y' = f(x, y) \\ y(x_0) = y_0 \end{cases}$$

donde $f \in C^5$, es decir, la quinta derivada de f existe y es continua y elijamos una anchura de paso h y sean $x_i = x_0 + ih$ e $y_i = y(x_i)$, para $i = 0, 1, 2....$ Consideremos:

$$\begin{cases} k_1^{(i)} = f\left(x_i, y_i\right) \\ k_2^{(i)} = f\left(x_i + \dfrac{h}{2}, y_i + \dfrac{k_1^{(i)}}{2}\right) \\ k_3^{(i)} = f\left(x_i + \dfrac{h}{2}, y_i + \dfrac{k_2^{(i)}}{2}\right) \\ k_4^{(i)} = f\left(x_i + h, y_i + k_3^{(i)}\right) \end{cases}$$

Así, el procedimiento de Runge-Kutta de cuarto orden es

$$y_{i+1} = y_i + \frac{1}{6}\left(k_1^{(i)} + 2k_2^{(i)} + 2k_3^{(i)} + k_4^{(i)}\right)$$

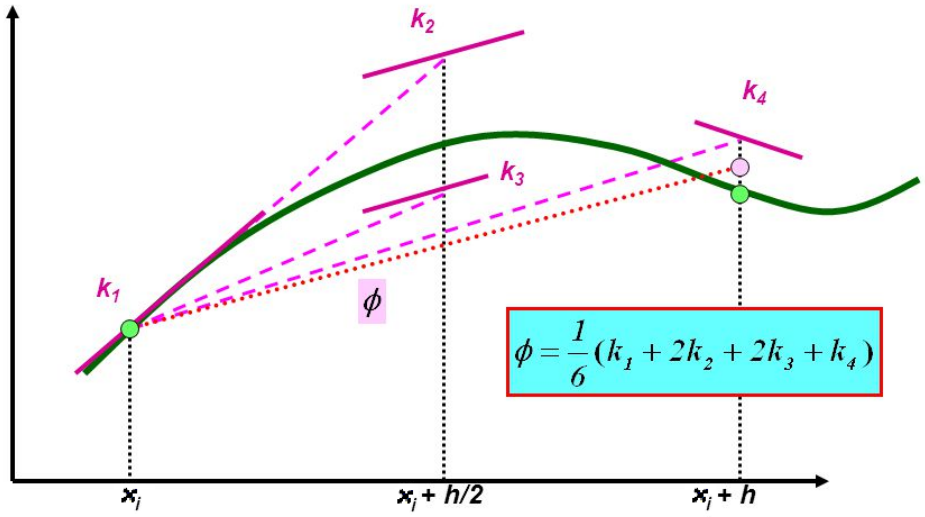

Figura 3.7: Representación gráfica del método de Runge-Kutta de cuarto orden

Método de Runge-Kutta-Felhberg

También conocido como **método de Runge-Kutta encajado** o **método de Felhberg**, emplea un tamaño de paso variable. Supone una buena mejora del algoritmo ajustando el valor del paso para mantener pequeños los errores locales de truncamiento. Consiste en aplicar los métodos de cuarto y quinto orden de Runge-Kutta y compararlos. Este método es muy bueno para utilizar en la cerca del apoastro, pero no se recomienda su uso cerca del periastro, donde el satélite se mueve más rápido. Se consideran los siguientes valores:

$$\begin{cases} k_1 = f(x_i, y_i) \\ k_2 = f\left(x_i + \frac{1}{4}h, y_i + \frac{k_1}{4}\right) \\ k_3 = f\left(x_i + \frac{3}{8}h, y_i + \frac{3}{32}k_1 + \frac{9}{32}k_2\right) \\ k_4 = f\left(x_i + \frac{12}{13}h, y_i + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3\right) \\ k_5 = f\left(x_i + h, y_i + \frac{439}{216}k_1 - 8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4\right) \\ k_6 = f\left(x_i + \frac{1}{2}h, y_i - \frac{8}{27}k_1 + 2k_2 + \frac{3554}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5\right) \end{cases}$$

De este modo, utiliza un método de Runge-Kutta con error de truncamiento local de orden 5,

$$\tilde{y}_{i+1} = y_i + \frac{16}{135}k_1 + \frac{6656}{12825}k_3 + \frac{28561}{56430}k_4 - \frac{9}{50}k_5 + \frac{2}{55}k_6$$

para calcular el error local en un método tipo Runge-Kutta de orden 4:

$$y_{i+1} = y_i + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5$$

La diferencia de estas aproximaciones es:

$$\Delta_{45} = \frac{1}{360}k_1 - \frac{128}{4275}k_3 - \frac{2197}{75240}k_4 + \frac{1}{50}k_5 + \frac{2}{55}k_6$$

Si este valor es mayor que la tolerancia, habrá que ver cómo se modifica el paso h, en función del siguiente parámetro:

$$s = 0{,}8408\left[\frac{10^{-8}}{\Delta_{45}}h\right]^{1/4}$$

Teniendo en cuenta unos valores mínimo, $h_{min} = \frac{h}{64}$ y máximo, $h_{max} = 64h$, para el tamaño del paso, determinamos los cambios de la siguiente forma:

- Si $s < 0{,}75$ y $h > 2h_{min}$ entonces $h = \frac{h}{2}$.
- Si $s > 1{,}75$ y $2h < 2h_{max}$ entonces $h = 2h$.

y volveríamos a calcular los k_i y repetir el proceso.

Si Δ_{45} es menor o igual que la tolerancia, hemos encontrado una aproximación aceptable, que viene dada por:

$$y = y_0 + \frac{25}{216}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5$$

La ventaja de este método es que sólo se requieren seis evaluaciones de f en cada paso. Los métodos de Runge-Kutta de órdenes 4 y 5, utilizados juntos requieren de cuatro evaluaciones de f para el método de cuarto orden y seis para el de quinto orden, para un total de 10 evaluaciones por paso. Por lo tanto, el método de Runge-Kutta-Fehlberg supone una disminución del 40 % de las evaluaciones de función sobre el uso de un par de métodos de cuarto y quinto orden.

4. Análisis del problema

En este capítulo explicaremos cuáles son los requisitos del presente proyecto y haremos un breve repaso al estado del arte mencionando las soluciones actuales más relevantes para el problema.

4.1. Requisitos de información

Los requisitos de información son:

- Almacenar la posición y velocidad de los satélites a lo largo de muchos instantes de tiempo (épocas).
- Almacenar la posición y velocidad de los astros del Sistema Solar en las mismas épocas elegidas para los satélites.

4.2. Requisitos funcionales

Los requisitos funcionales son:

- Generación de un conjunto de datos a partir de posiciones precisas de satélites.
- Optimización de parámetros orbitales de modo que se reduzca el error de la predicción en una serie de puntos.
- El modelo de red neuronal debe ofrecer una corrección de la predicción de la trayectoria de los satélites realizada por el modelo SGDP4.

4.3. Requisitos no funcionales

Los requisitos no funcionales son:

- El conjunto de datos debe ser amplio y de calidad. Tiene que haber consistencia de los sistemas de referencia espaciales y temporales de todos los datos que contiene y especificándose claramente cuáles son.
- Los datos deben estar bien estructurados y ser fácilmente accesibles.
- El tiempo para realizar una predicción con la red neuronal no debe ser superior a 3 segundos, puesto que el objetivo es tener una estimación precisa en poco tiempo.

4.4. Estado del arte

A diferencia de la modelización basada en la física, los métodos de aprendizaje automático no necesitan simular las condiciones ambientales ni todos los factores que influyen en un satélite para predecir su posición de un modo explícito. Estos modelos aprenden en base a grandes cantidades de datos observados, similar a la cognición humana en el aprendizaje de experiencias pasadas para predecir eventos futuros. Un ejemplo práctico de esto es que con recibir las posiciones de un determinado planeta, una red puede predecir con bastante precisión el efecto que tendrá el campo gravitatorio de dicho planeta sobre la trayectoria de un satélite que lo orbite, considerando que no actúan fuerzas externas.

Algunos de los modelos utilizados hoy en día son los basados en redes neuronales recurrentes, tales como las LSTM, para predecir los elementos orbitales en un época futura [10]. También existen modelos que utilizan una hibridación de redes LSTM y regresión lineal para realizar una predicción sobre los parámetros orbitales [13].

Otras soluciones que se han aportado buscan predecir el error cometido en la predicción con redes neuronales densas [12] o utilizan máquinas de vector soporte (SVM) [11], que es muy resistente a los valores atípicos, para predecir el estado orbital.

Por último, también se han propuesto modelos híbridos basados en *autoencoders* y *random forest* para reducir el error de las predicciones realizadas por SGP4 [7].

5. Diseño de la solución

A continuación explicaremos las decisiones que se han ido tomando, tanto para obtener los datos como para implementar y entrenar los modelos. Principalmente han estado influidas por la precisión de los datos y el tiempo de cómputo.

5.1. Idea inicial

Tras comenzar con el estudio teórico y la lectura de diversos artículos científicos, los más relevantes citados en la sección de estado del arte, se decide implementar y entrenar modelos de red neuronal multicapa (**perceptrón multicapa**) y red neuronal recurrente (**RNN**). La primera, como una idea básica que sirva para ver la efectividad y viabilidad de un modelo base, y la segunda para intentar aprovechar la periodicidad de las órbitas de los satélites, en las que pueden existir secuencias de estados de diversos elementos que influencien la trayectoria. Dichos modelos deben recibir los datos reales en una época y las predicciones realizadas con SGDP4 en otra, y predecir el error de dicha predicción, de modo que sirva para corregirla.

La obtención de la verdadera posición y velocidad, tanto para la época base como para la de la predicción, esta último respecto a la cuál se calcula el error, se va a realizar con el método HPOP. La ventaja de utilizar dicho método es que, aunque no realiza una predicción perfecta, es posible aplicarlo a partir de cualquier punto inicial, por lo que, en un principio, es sencillo generar conjuntos de datos o *datasets*.

El modelo SGDP4 necesita los elementos orbitales del satélite, cuya estimación se realiza a partir de la posición y velocidad del mismo en la época en cuestión, que podría variar ligeramente respecto de la órbita que mejor se ajuste a la trayectoria en su conjunto. Por tanto, se propone optimizar los parámetros para reducir el error en la predicción del modelo SGDP4, de acuerdo a un número determinado de puntos a pasado y futuro desde la época en cuestión, en los que se conoce la posición real o una aproximación muy precisa realizada con modelos de alta precisión como HPOP. Por ejemplo, optimizar de modo que se reduzca el error en 10 puntos a futuro en la época, espaciados 30 minutos. Para estar seguros de que la optimización es correcta, se propone disponer de una función que nos permita representar tres trayectorias: la real, la predicha por SGDP4 con los parámetros previos al ajuste, y posteriores al ajuste. Si el ajuste ha sido satisfactorio, esta última estará más cercana a la real que con los parámetros previos al ajuste.

Debido a la complejidad del problema, no se propone crear modelos generales que sirvan para cualquier satélite, sino para un satélite determinado. De otra manera, la red tendría que aprender a realizar predicciones sobre satélites en condiciones muy dispares, con parámetros físicos diferentes (por ejemplo, la superficie, que influye en el rozamiento). Al limitarse a un satélite, el modelo se debería ajustar a medida para las condiciones de dicho satélite, sin necesidad de conocerlas explícitamente.

La entrada de los modelos más básicos es la siguiente:

- Fecha de la época base
- Posición en la época base
- Fecha de la época de la predicción
- Posición predicha por SGDP4

donde cada fecha será un número entero (formato UNIX) y la posición se compondrá de 3 números correspondientes a las coordenadas en metros en el marco de referencia GCRF. Una vez entrenados estos primeros modelos, se probará a incluir como parámetros de entrada a la red las velocidades en el momento inicial, con las unidades en metros por segundo y en el sistema GCRF. También se entrenaran modelos que reciban, en vez de la posición predicha, el movimiento predicho.

La salida de la red es el vector error de la predicción, es decir, una tripleta donde cada elemento representa la desviación de la predicción en las coordenadas X, Y y Z, respectivamente, de modo que sumando ese vector con la predicción se obtiene la corrección de la misma.

Se pretende entrenar modelos con los parámetros sin optimizar y con los parámetros optimizados para poder evaluar el efecto de dicho ajuste.

Para la elección de las fechas de la predicción, se decidió escoger 10, a partir de cada punto base a intervalos de tiempo de 30 minutos, 1, 2, 5 y 10 horas y 1, 2, 5, 10 y 30 días, de modo que se pudiera evaluar el rendimiento del modelo, tanto para predicciones a corto como a largo plazo.

La estructura de datos elegida para el del dataset fue el *dataframe*, pero el coste computacional de la función HPOP y de la optimización de parámetros orbitales, sumado a la necesidad de conocer ciertos parámetros del satélite que no son comúnmente conocidos para aplicar HPOP, indicaban que la creación de un conjunto de datos de calidad no era factible.

5.2. Modificación del diseño inicial

Para evitar tener que utilizar la función HPOP, se propone el uso de archivos de navegación RINEX que, además, contienen con exactitud plena las posiciones y velocidades de los satélites. Así, se intentará reducir el error de SGDP4 con respecto a la posición real y no a una predicción de alta precisión.

Para la generación del *dataset*, se buscó primero un repositorio del que descargar los datos. A continuación hubo que averiguar de qué satélite había más datos, y posteriormente guardar sus posiciones, velocidades y épocas en un *dataframe*. Dicho *dataframe* sirvió de ayuda para generar el *dataset*, ya que en él estaban las efemérides que utilizamos para obtener y aproximar los elementos orbitales que necesita el modelo SGDP4. A continuación se detalla el proceso.

El repositorio debía contener archivos RINEX versión 2.10, que es la soportada por el paquete *asteRisk*, por lo que el escogido fue el del Instituto de Estadística y Cartografía de la Junta de Andalucía [3]. Concretamente, se utilizaron los datos procedentes del receptor de la estación de Córdoba, situada en la Consejería de Fomento y Vivienda. Dicho repositorio dispone de datos sobre satélites GPS y GLONASS.

Para descargar los archivos se utilizó *web scrapping*. Posteriormente se realizó su proceso y análisis, para determinar cuál era el satélite con más mensajes RINEX disponibles y generar el *dataset* con sus datos. El procesado se ha realizado con ayuda de la función de lectura RINEX incluida en el paquete *asteRisk*. Para la elección del satélite se procedió a la lectura de todos los archivos, contando el número de mensajes RINEX que había de cada satélite, de modo que al terminar se podía obtener sin dificultad cuál era el satélite del que más información había. Cabe mencionar que el tratamiento de los GPS y los GLONASS había que hacerlo por separado, ya que los campos de la lista devuelta por las respectivas funciones de lectura eran distintos.

El satélite con más información disponible resultó ser el Kosmos 2514, de la constelación GLONASS y con código identificativo 17. Para almacenar sus datos en un *dataframe* se realizó de nuevo la lectura de todos los archivos y se fue guardando la posición y velocidad, separadas en coordenadas, y la época de los mensajes correspondientes a este satélite, convirtiendo previamente la posición y velocidad al marco de referencia GCRF (puesto que estaban en el ITRF). Así, dicho conjunto de datos tenía la siguiente estructura:

Época	Posición	Velocidad

Cuadro 5.1: Datos obtenidos de los ficheros RINEX del satélite

Una vez tuvimos el *dataframe* con las posiciones, velocidades y épocas pasamos a la generación del *dataset* para entrenar los modelos. La generación se realizó en dos pasos. En primer lugar, se guardaron los datos de la fecha inicial, la posición y velocidad iniciales reales, la fecha de la predicción, la posición y velocidad reales en esta segunda fecha y la posición y velocidad predichas en ese instante con los parámetros optimizados. En el segundo paso, se añadieron las predicciones con los parámetros sin optimizar.

La elección de las fechas base se realizó tomando de manera aleatoria 5000 fechas entre las del *dataframe* del satélite. En la optimización de los elementos orbitales y la elección de fechas para las predicciones se mantuvo la idea inicial, con dos pequeñas modificaciones, motivadas por la posible falta de datos en una fecha concreta, ya que no sabemos si disponemos de ella exactamente. Estos dos cambios fueron:

- La optimización de los parámetros orbitales en cada punto se realizó en base a los 5 anteriores, él mismo y los 5 posteriores, ordenados por fecha.
- En el tiempo a predecir se permitió una desviación del 30 %, es decir, para predecir a 100 minutos, se utilizó la fecha del conjunto más cercana, siempre y cuando ésta distara un máximo de 130 minutos de la correspondiente.

De este modo, el dataset tomó esta estructura, donde si aparece "optim" quiere decir que se han empleado los parámetros optimizados:

Época inicial	Posición inicial	Velocidad inicial
Época de la predicción	Posición predicha optim	Velocidad predicha optim

Cuadro 5.2: Dataset con datos reales y predicciones SGDP4 con parámetros optimizados

Una vez tuvimos este *dataset* faltaba añadir las predicciones de SGDP4 con los parámetros sin optimizar. Para ello se iba recorriendo las filas de dicho conjunto de datos y con los valores de la época, posición y velocidad iniciales, y la época de la predicción realizamos la predicción en cuestión y la añadimos al *dataset*, adquiriendo la siguiente estructura:

Época inicial	Posición inicial	Velocidad inicial
Época de la predicción	Posición predicha optim	Velocidad predicha optim
-	Posición predicha	Velocidad predicha

Cuadro 5.3: Dataset con posiciones reales y predicciones SGPD4 con parámetros optimizados y sin optimizar

5.3. Creación de los modelos

Para la creación y entrenamiento de modelos se opta por TensorFlow y Keras con R, por lo que hay que utilizar la versión de estas librerías en dicho ecosistema. Además, se considera el uso de CUDA y cuDNN para acelerar el proceso de entrenamiento.

Para la separación de los datos en conjuntos de entrenamiento y test se utilizó la librería *rsample*.

Antes de comenzar con el entrenamiento de los modelos debemos normalizar los datos, es decir, reducir su rango y que así estén aproximadamente en la misma escala, lo que ayudará a la red a realizar mejores predicciones. La normalización elegida fue el **escalado robusto** (Robust Scaler), que consiste en sustraer la mediana y escalar los datos de acuerdo al rango intercuartílico (IQR). Si lo queremos expresar en una fórmula quedaría:

$$\frac{x_i - Q_2(x)}{Q_3(x) - Q_1(x)}$$

La ventaja del escalado robusto radica en que los valores que utiliza para escalar (mediana y rango intercuartílico) no se ven afectados por los valores atípicos (extremos), luego mantiene la distancia relativa entre los valores prácticamente intacta.

Notar que los parámetros de este método de normalización, es decir, la mediana y el rango intercuartílico, se calculan sobre el conjunto de entrenamiento y son los que se le aplica al conjunto de test y a cualquier otro dato de entrada con el que se vayan a realizar predicciones.

La estructura de la red neuronal implementada y entrenada se compone de una capa de entrada, con tantas neuronas como datos de entrada haya, 3 capas densas

con activación *relu* y 256, 128 y 64 neuronas, respectivamente, y la capa de salida con 3 neuronas, una por cada coordenada (X, Y, Z), sin función de activación (i.e. la identidad), ya que se quiere realizar regresión para estimar el error de la predicción para cada coordenada.

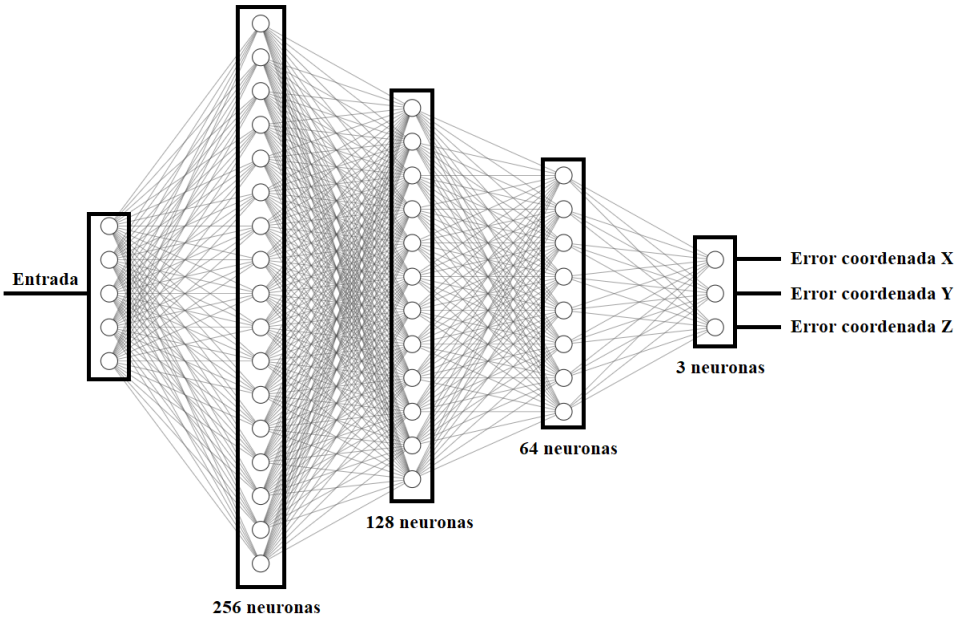

Figura 5.1: Estructura de las redes neuronales implementadas

Como función de pérdida se eligió el error cuadrático medio, MSE. La técnica de optimización elegida fue ADAM (Momento Adaptativo) y la métrica fue el error absoluto medio, MAE, para tener una idea de cuánto nos desviamos del valor real en su propia escala. Tras varias pruebas con algunas combinaciones, el tamaño de *batch* elegido fue de 128 y el número de *epochs* del entrenamiento de 100.

La división en conjuntos de entrenamiento y prueba se realizó con estratificación, considerando el tiempo a predecir, es decir, la diferencia entre la fecha de la predicción y la fecha base.

5.4. Evaluación de resultados

Con la métrica empleada MAE podemos tener cierta idea del rendimiento del modelo, pero no tiene una interpretación fácil debido a que los datos se encuentran normalizados. Por tanto, se decide desnormalizar la salida de la red, para obtener una predicción del error en la escala original y, posteriormente, sumar ese error a la predicción de SGDP4 para obtener su corrección.

Tras obtener la predicción de la posición corregida se procede a calcular el error medio con respecto a las posiciones reales y compararlo con el de las predicciones sin corregir, para cuantificar realmente si mejora la predicción de la posición y cuánto lo hace.

5.5. Ampliación de la experimentación

Después de obtener los primeros modelos se procedió añadir datos sobre los astros del sistema solar al *dataset*, ya que sus campos gravitatorios y movimientos oscilatorios (como la libración lunar) tienen efecto sobre los satélites.

La obtención de estos datos se realizó utilizando el modelo *JPL DE440*, proporcionado por el Jet Propulsion Laboratory de la NASA e incluido como una función en el paquete *asteRisk* bajo el nombre de *JPLephemerides*.

Previamente a la adición al conjunto de datos de los satélites, se creó otro *dataset* auxiliar para los astros, con la posición y velocidad de todos los astros del Sistema Solar salvo la Tierra (puesto que al estar en el marco GCRF valdrían 0) y la libración lunar y sus derivadas primera y segunda, en todas las fechas que se encontraban entre las fechas base del *dataset* de los satélites. Las posiciones y velocidades separadas en sus coordenadas de los ejes X, Y y Z, y la libración lunar en sus componentes ϕ, θ y ψ. De este modo, la estructura de este *dataset* quedó de la siguiente forma:

Época	Posición del Sol	Velocidad del Sol
-	Posición de Mercurio	Velocidad de Mercurio
-	Posición de Venus	Velocidad de Venus
-	Posición de Marte	Velocidad de Marte
-	Posición de Júpiter	Velocidad de Júpiter
-	Posición de Saturno	Velocidad de Saturno
-	Posición de Urano	Velocidad de Urano
-	Posición de Neptuno	Velocidad de Neptuno
-	Posición de Plutón	Velocidad de Plutón
Libración Lunar	d(Libración Lunar)	dd(Libración Lunar)

Cuadro 5.4: Dataset con datos de los astros

Una vez generado este *dataset* se procede a unirlo con el de las posiciones de satélites, para así obtener un conjunto de datos compuesto por: una época inicial, posición y velocidad del satélite; los datos mencionados anteriormente de los astros en dicha época, la época en la que realizar la predicción y la posición y velocidad predichas por SGDP4, tanto con los parámetros optimizados como con los parámetros sin optimizar en dicha época. Las fechas mencionadas estaban guardadas en formato cadena de texto (para que sea legible en caso de que haya que depurar) y en formato UNIX, y el resto de datos guardados por componentes y en el marco de referencia GCRF. Este nuevo *dataset* nos sirvió para generar nuevos modelos que recibieran como entrada lo mismo que los anteriores y, además, los datos de los astros en las fechas iniciales.

No se guardaron en el *dataset* los errores de las predicciones, porque tras realizar pruebas de rendimiento, era más rápido leer el fichero en el que estaba guardado y añadirlos según fuera necesario que almacenarlos en el propio *dataset*.

6. Problemas encontrados

En esta sección se expondrán las complicaciones que tuvieron lugar durante el desarrollo del proyecto, así como la solución aportada para cada una de ellas.

6.1. Ajuste de parámetros

El primer problema aparece con la optimización de los parámetros orbitales. Específicamente, se detectó que el paquete no trataba algunos casos minoritarios, la mayoría de los cuales eran con valores cercanos a la frontera de su rango, en los que se producía una división entre 0 que causaba error. Dicho error fue reportado para su futuro arreglo por parte del autor del paquete. Para poder realizar el ajuste adecuadamente se propuso utilizar un manejador de excepciones que devolviera un valor muy alto en la función de error a minimizar en caso de fallo, de modo que nunca se eligiera la combinación de parámetros en cuestión.

El segundo problema surge en algunas predicciones que a corto plazo se realizaban sin problema, pero cuando se aumentaba unas horas el tiempo de la predicción daba error. Se notificó también al autor del paquete para su arreglo. Este fallo apareció cuando se recorría una lista de fechas para realizar la predicción en ellas y añadirlas al *dataset*, por lo que la solución implementada fue utilizar un manejador de excepciones que en caso de fallo, diera como predicción *NA*.

6.2. Representación de órbitas

Al representar las trayectorias obtenidas por SGDP4 con los parámetros optimizados y sin optimizar, así como las generadas por HPOP, se observó que la trayectoria de los parámetros optimizados difería excesivamente de las otras dos, lo cual es signo de algún problema. Se baraja que puedan ocurrir tres cosas:

- La función de optimizar no sea correcta
- No se esté haciendo bien el cambio de sistemas de coordenadas
- El método de estimación de parámetros iniciales a partir de una posición y velocidad iniciales, que había sido añadido recientemente al paquete, no fuera correcto

Para poder descubrir cuál era el problema hubo que realizar una profunda investigación, que incluyó depurar el método de estimación de parámetros y multitud de pruebas con la función de optimizar y con cambios de sistemas de coordenadas para comprobar que las transformaciones entre marcos de referencia eran correctas. Durante este proceso se identificó un error en un cambio de coordenadas que no se estaba haciendo correctamente; sin embargo, el error global no se debía a ese fallo. Finalmente se descubrió que el error provenía del formato de fecha, que por defecto es CET o CEST

y no UTC (difieren en 1 ó 2 horas), por lo que al hacer el cambio de sistema no se estaba pasando adecuadamente la época.

6.3. Repositorios

La búsqueda de un repositorio con archivos RINEX versión 2.10, compatible con las funciones de lectura de *asteRisk*, no fue fácil, debido a que la mayoría de los repositorios de las agencias estatales están en la versión 3 y las que se encontraban de versión 2 eran de la 2.11 en adelante. En los repositorios de la NASA sí que había de la versión 2.10, pero su estructura de directorios era compleja y no resultaba sencillo distinguir dicha versión de la 2.11. Finalmente se encontró el repositorio del Instituto de Estadística y Cartografía de la Junta de Andalucía, donde estaban separados completamente los ficheros de versión 2 de los de la 3 y, además, los de la versión 2 eran concretamente la 2.10.

6.3.1. Estación de la Universidad de Sevilla

Los archivos RINEX que se descargaron la primera vez correspondían a la estación de la Universidad de Sevilla, pero durante el generación del conjunto de datos aparecieron errores inesperados que, tras depurar el código e investigar, resultaron estar causados por la presencia de observaciones de datos repetidas, es decir, varios mensajes RINEX iguales. Se procedió de nuevo a la lectura de los datos del satélite, puesto que no deberían aparecer mensajes duplicados en dichos archivos, y resultó que todas las fechas, épocas de las efemérides, habían cambiado. Tras esto se empezó un análisis más profundo y durante la lectura manual de un archivo RINEX apareció el problema: ciertos campos de la cabecera, tales como las correcciones temporales, se encontraban vacíos, lo que ocasionaba que, durante el lectura que realiza la función del paquete, las variables correspondientes tomaran el valor *NULL* y el cálculo de la fecha, al aplicar la corrección del bias, no se realizaba correctamente.

Después de analizar la situación se decidió cambiar la estación de la que se descargaron los datos, escogiéndose finalmente la de Córdoba, cuyos archivos RINEX sí disponían de las cabeceras completas y no mostraban ningún problema.

7. Implementación

En este capítulo se presentarán las principales herramientas empleadas en el desarrollo del proyecto, así como las funciones implementadas y los extractos de código elaborados. Entre dichas funciones se encuentran las que han sido utilizadas en la creación del *dataset* utilizado para entrenar la red neuronal. Mención especial merece la creación de dicho conjunto de datos, puesto que supone una contribución de gran peso, que involucra ajustes de parámetros complejos y se ha estructurado de un modo que simplifica su tratamiento, encontrándose todos los datos de magnitudes equiparables en el mismo marco de referencia y con las mismas unidades.

7.1. Herramientas

7.1.1. R

R es un lenguaje de programación sumamente potente para realizar distintos cálculos científicos, numéricos y estadísticos, así como para crear gráficas y figuras de gran calidad.

Surge de la mano de Robert Gentleman y Ross Ihaka del Departamento de Estadística de la Universidad de Aucklanden en 1993 como una reimplementación de software libre del lenguaje S, adicionado con soporte para ámbito estático. Se trata de uno de los lenguajes de programación más utilizados en investigación científica, siendo además muy popular en los campos de aprendizaje automático, minería de datos, bioinformática o econometría, entre otros. Todo esto gracias a la posibilidad de utilizar multitud de paquetes con amplias funcionalidades de cálculo y representación gráfica.

Se distribuye bajo la licencia GNU GPL y está disponible para los sistemas operativos Windows, Macintosh, Unix y GNU/Linux.

Los paquetes de R más utilizados para este proyecto se describen seguidamente.

7.1.2. asteRisk y asteRiskData

El paquete de R *asteRisk* proporciona funcionalidades básicas para el cálculo de posiciones de satélites a partir de un vector de estado. Incluye implementaciones de:

- Los modelos de perturbaciones simplificadas SGP4 y SDP4, incluida una función *sgdp4* que elige el más adecuado de los dos.
- Un modelo de alta precisión (HPOP).
- Utilidades para leer archivos TLE y RINEX.
- Cambio de coordenadas entre sistemas de referencias.

- Modelos de posiciones de los astros, como el JPL DE440.

Muchas de las funcionalidades del paquete requieren de multitud de coeficientes y otros datos, como información sobre las mareas o tormentas solares, por lo que también es necesario el paquete *asteRiskData*.

7.1.3. Dplyr

Forma parte del ecosistema de paquetes tidyverse, que trabajan conjuntamente para propiciar una ciencia de datos mejor estructurada y que incluye dplyr, tidyr, readr o ggplot2, entre otros.

El paquete dplyr es un versión optimizada del paquete plyr. Contiene funciones muy potentes que sería muy complicado replicar, otra contribución importante es que proporciona una "gramática"para manipular y realizar operaciones con *dataframes*. Esto es muy útil, ya que proporciona una abstracción que anteriormente no existía y permite que el código sea más legible. También cabe destacar que las funciones de dicho paquete son muy rápidas, puesto que están implementadas con el lenguaje C++.

7.1.4. Plotly

Con Plotly se pueden construir gráficas de alta calidad e incorporar interactividad en los gráficos, mostrar la información que contiene cada dato, mostrar datos filtrando por algún factor, etc. Además, los gráficos son ajustables (*responsive* en inglés), es decir, se adaptan a las dimensiones de la ventana en que aparecen.

7.1.5. Reticulate

Reticulate proporciona una interfaz para utilizar Python desde R. Este paquete será utilizado internamente por las librerías tensorflow y keras de R.

7.1.6. Python

Python es un lenguaje de programación enfocado en la legibilidad del código. Tiene su origen a finales de los años ochenta en el Stichting Mathematisch Centrum (CWI) de Países Bajos. Su creador fue Guido van Rossum, que se inspiró en el lenguaje ABC y le puso el nombre debido a su afición por los humoristas Monty Python. Dispone de un catálogo muy amplio de librerías en multitud de campos, desde la inteligencia artificial hasta la optimización. A continuación se describen las librerías de Python que van a ser utilizadas en este trabajo.

7.1.7. Beautiful Soup

Es una librería concebida para analizar textos HTML, incluso aunque el marcado sea incorrecto. Por lo tanto, es ampliamente utilizada para realizar web scrapping.

7.1.8. TensorFlow

Es una libería para aprendizaje automático. Desarrollada por Google y de códgio abierto permite la creación y el entrenamiento de redes neuronales de un modo muy cómodo.

Pese a que no está adaptado como tal el paquete a R, sí existe un paquete que realiza vincula R con Python y permite utilizar TensorFlow con dicho lenguaje.

7.1.9. CUDA y cuDNN

Son dos tecnologías desarrolladas por NVIDIA para acelerar el procesamiento en paralelo en las tarjetas gráficas (GPUs) y mejorar así el rendimiento en aplicaciones de aprendizaje profundo (deep learning.)

7.2. Desarrollo

7.2.1. Instalación de asteRisk y asteRiskData

Lo primero que debemos hacer es instalar los paquetes de R *asteRisk* y *asteRiskData*. Para ello ejecutamos los siguientes comandos:

```
1 install.packages("asteRisk")
2 install.packages("asteRiskData", repos="https://rafael-ayala.github.io/
    drat/")
```

Extracto de código 7.1: Instalación asteRisk y asteRiskData

Cada vez que se carga *asteRiskData* aparece por consola un aviso de que se ejecute el comando *getLatestSpaceData()* para obtener los datos más recientes de tormentas solares, mareas, etc. La instrucción sería la siguiente:

```
1 getLatestSpaceData(target="all")
```

Extracto de código 7.2: Obtención de datos recientes de los astros

7.2.2. Cálculo de elementos clásicos orbitales

El marco de referencia que se decidió utilizar es el GCRF y como el paquete *asteRisk* no dispone de una función que, a partir de la posición y velocidad de un

satélite en dicho sistema, obtenga los elementos orbitales, debemos implementarla. En dicha función será necesario realizar la conversión de GCRF a TEME, que involucra el uso de cuaterniones para aplicar las rotaciones pertinentes, y posteriormente utilizar la función *ECItoKOE* con la conversión de unidades de medida adecuadas.

```r
get_params_orbitales <- function(dateTime, pos, vel){
  # Comienza la transformacion a TEME
  # Hace falta la fecha en MJD_UTC.
  date_mjd_utc <- dateTimeToMJD(dateTime, timeSystem = "UTC")
  # Cuaternion con "rotacion necesaria"
  quaternion <- asteRisk:::rotationGCRFtoTEME(date_mjd_utc)
  # Convierte el cuaternion a "Directed Cosine Matrix"
  dcm <- asteRisk:::quaternionToDCM(quaternion)
  # Obtencion del TEME, %*% es la multiplicacion de matrices
  pos <- as.numeric(dcm %*% pos)
  vel <- as.numeric(dcm %*% vel)
  koe <- ECItoKOE(pos, vel)
  # semiMajorAxisToMeanMotion devuelve la rotacion en vueltas por dia. Se
    multiplica al final para transformar a radianes por minuto
  n0 <- asteRisk:::semiMajorAxisToMeanMotion(koe$semiMajorAxis)*((2*pi)/(
   1440))
  e0 <- koe$eccentricity
  i0 <- koe$inclination
  M0 <- koe$meanAnomaly
  omega0 <- koe$argumentPerigee
  OMEGA0 <- koe$longitudeAscendingNode
  return(c(n0, e0, i0, M0, omega0, OMEGA0))
}
```

Extracto de código 7.3: Obtención de parámetros orbitales a partir de posición y velocidad en GCRF

7.2.3. Calcular el error medio de predicción

Puesto que la idea principal del proyecto es mejorar la precisión de SGDP4 frente a HPOP, resulta natural calcular el error cometido con SGDP4 en la predicción. La primera implementación fue una función que recibía dos conjuntos de posiciones, pareadas según la época, es decir, el instante de tiempo, y calculaba la distancia media entre las posiciones. La utilidad inicial era utilizarla con una lista de predicciones de HPOP y otra de SGDP4, aunque finalmente no se utilizaó para HPOP sino que sirvió de ayuda en la interpretación de los resultados de los modelos, para ver el como variaba el error antes y después de aplicar la corrección de la red.

```r
calcula_error_medio <- function(posiciones_reales, posiciones_predichas)
   {

  error_prediccion <- 0
  num_posiciones <- nrow(posiciones_reales)
```

```
  for (i in 1:num_posiciones) {
    pos_real <- posiciones_reales[i,]
    pos_pred <- posiciones_predichas[i,]
    dif <- c(pos_real$X, pos_real$Y, pos_real$Z) - c(pos_pred$X, pos_pred$Y, pos_pred$Z)
    error_prediccion <- error_prediccion + sum(dif*dif)^0.5
  }

  return(error_prediccion/num_posiciones)
}
```

Extracto de código 7.4: Cálculo de error medio entre dos conjuntos de posiciones

Posteriormente se implementó otra función que recibía una serie de posiciones, la fecha base de a la que corresponden esas posiciones, los elementos orbitales, el coeficiente de rozamiento y la lista de tiempos a propagar. De este modo se iban realizando las predicciones puntuales con SGDP4 y se calculaba el error de este método respecto del conjunto de posiciones pasado como parámetro.

Notar que el valor *posiciones* que recibe la función debe ser un *dataframe* en el que haya, al menos, una columna para cada coordenada (X, Y, Z) y estén expresadas en metros.

```
get_error_medio <- function(posiciones, fecha_base, params_orbitales, Bstar, lista_tiempos) {

  error_prediccion <- 0
  num_posiciones <- nrow(posiciones)
  for (i in 1:num_posiciones) {
    tiempo_prediccion <- lista_tiempos[i]
    prediction_TEME <- sgdp4(n0=params_orbitales[1],
                             e0=params_orbitales[2],
                             i0=params_orbitales[3],
                             M0=params_orbitales[4],
                             omega0=params_orbitales[5],
                             OMEGA0=params_orbitales[6],
                             Bstar=Bstar,
                             initialDateTime=fecha_base,
                             targetTime=tiempo_prediccion)
    #Queremos las predicciones en metros
    prediction_GCRF <- TEMEtoGCRF(prediction_TEME$position*1000,
                                  prediction_TEME$velocity*1000,
                                  as.POSIXct(fecha_base, tz="UTC")+tiempo_prediccion)
    print(prediction_GCRF$position)
    punto <- posiciones[i,]
    dif <- c(punto$X, punto$Y, punto$Z) - prediction_GCRF$position
    error_prediccion <- error_prediccion + sum(dif*dif)^0.5
  }
```

```
25
26   return(error_prediccion/num_posiciones)
27 }
```

Extracto de código 7.5: Cálculo de error medio SGDP4 respecto a unas posiciones dadas

7.2.4. Optimización de parámetros orbitales

La optimización de parámetros debe realizarse con el objetivo de reducir el error de la predicción respecto a la posición real, por lo que vamos a necesitar una función que calcule dicho error. Aunque la función anterior lo calcula, no nos sirve porque no recibe como parámetros estrictamente aquellos que se quieren optimizar (los elementos orbitales), lo cual es necesario al utilizar la función *optimize* de R. Así, los únicos parámetros de esta función de error serán los parámetros orbitales, y las posiciones de HPOP las recibirá como variables globales, por lo que tendremos que crear una función auxiliar.

Para la implementación de la función de optimización se han elaborado otras funciones auxiliares que se describen a continuación.

Obtención de cotas para los parámetros orbitales

La función de optimización necesita conocer el rango de cada parámetro a optimizar, por lo que se han implementado dos funciones distintas. La primera devuelve el rango completo en el que pueden estar los elementos orbitales, mientras que la segunda recibe unos parámetros concretos y porcentaje, que define cuántos permitimos que se desvíen, siempre y cuando esa variación quede dentro del dominio de los parámetros. La utilidad de esta función es facilitar el ajuste de los elementos orbitales en casos en los que se esperara que los valores optimizados no disten mucho de los valores originales.

Cabe mencionar que se ha sumado o restado 0.0001 a los valores frontera del rango para evitar problemas con casos extremos y el Bstar se ha acotado entre 0.0005 y -0.0005, porque el coeficiente de rozamiento en las capas altas de la atmósfera es muy pequeño y así acotamos el espacio de búsqueda.

```
1  get_cotas_params_orbit  <- function(){
2
3    n0_inf <- 0+0.0001
4    n0_sup <- pi-0.0001
5    e0_inf <- 0+0.0001
6    e0_sup <- 1-0.0001
7    i0_inf <- 0+0.0001
8    i0_sup <- 2*pi-0.0001
9    M0_inf <- 0+0.0001
10   M0_sup <- 2*pi-0.0001
11   omega0_inf <- 0+0.0001
```

```
  omega0_sup <- 2*pi-0.0001
  OMEGA0_inf <- 0+0.0001
  OMEGA0_sup <- 2*pi-0.0001
  Bstar_inf <- -0.0005
  Bstar_sup <- 0.0005

  cotas_inf <- c(n0_inf, e0_inf, i0_inf, M0_inf, omega0_inf, OMEGA0_inf,
   Bstar_inf)
  cotas_sup <- c(n0_sup, e0_sup, i0_sup, M0_sup, omega0_sup, OMEGA0_sup,
   Bstar_sup)

  cotas <- data.frame("cotas_inf" = cotas_inf,
                      "cotas_sup" = cotas_sup)
  return(cotas)
}
```

Extracto de código 7.6: Obtención de cotas máximas

```
get_cotas_params_orbit_porc  <- function(params, porcentaje_error=0.2){

  if(porcentaje_error<0){
    porcentaje_error <- 0.2
  } else if(porcentaje_error>1){
    porcentaje_error <- 0.8
  }

  n0_inf <- max(0.0001,(1-porcentaje_error)*params[1])
  n0_sup <- min(pi-0.0001, (1+porcentaje_error)*params[1])
  e0_inf <- max(0.0001, (1-porcentaje_error)*params[2])
  e0_sup <- min(1-0.0001, (1+porcentaje_error)*params[2])
  i0_inf <- max(0.0001, (1-porcentaje_error)*params[3])
  i0_sup <- min(2*pi-0.0001, (1+porcentaje_error)*params[3])
  M0_inf <- max(0.0001, (1-porcentaje_error)*params[4])
  M0_sup <- min(2*pi-0.0001, (1+porcentaje_error)*params[4])
  omega0_inf <- max(0.0001, (1-porcentaje_error)*params[5])
  omega0_sup <- min(2*pi-0.0001, (1+porcentaje_error)*params[5])
  OMEGA0_inf <- max(0.0001, (1-porcentaje_error)*params[6])
  OMEGA0_sup <- min(2*pi-0.0001, (1+porcentaje_error)*params[6])
  Bstar_inf <- -0.0005
  Bstar_sup <- 0.0005

  cotas_inf <- c(n0_inf, e0_inf, i0_inf, M0_inf, omega0_inf, OMEGA0_inf,
   Bstar_inf)
  cotas_sup <- c(n0_sup, e0_sup, i0_sup, M0_sup, omega0_sup, OMEGA0_sup,
   Bstar_sup)

  cotas <- data.frame("cotas_inf" = cotas_inf,
                      "cotas_sup" = cotas_sup)
```

```
29
30   return(cotas)
31 }
```

Extracto de código 7.7: Obtención de cotas con porcentaje

Optimización de parámetros orbitales

La función de optimización recibe como parámetros el *dataset* base del que obtener datos, la fecha base en la que queremos optimizar los parámetros, un número de puntos que indica cuántas épocas (fechas) se van a utilizar para optimizar y el Bstar (coeficiente de rozamiento) inicial. La idea es que, dadas una fecha, una posición y una velocidad, podamos obtener los parámetros orbitales y con ellos una órbita completa; sin embargo, esta órbita puede que no sea la que mejor se ajusta a la real, de modo que conviene realizar un ajuste de parámetros para reducir el error de esta órbita respecto a la real.

El número de puntos nos indica cuantas fechas, además de la fecha base, vamos a elegir. Si dicho número es n, se escogerán las $\frac{n}{2}$ últimas fechas presentes en el *dataset* antes de la fecha base y las $\frac{n}{2}$ siguientes a dicha fecha, por orden cronológico.

La optimización se realiza en base a un número de puntos, es decir, fechas, que recibe como parámetro, eligiéndose la mitad de ellos anteriores a la fecha elegida como base y la otra mitad posteriores.

Notar también que el Bstar (coeficiente de rozamiento del satélite) se le pasa como parámetro, además de los orbitales, ya que es otro factor que influencia en la trayectoria y es usado por el algoritmo SGDP4, pero no se puede obtener a partir de la posición y velocidad.

Una vez determinadas estas fechas, se procede a la definición de las cotas con una de las funciones anteriores. A partir de ella, se continúa con la implementación de una función auxiliar que calcula el error recibiendo sólo los parámetros orbitales y que utiliza un manejador para devolver un número muy elevado en caso de algún problema en la predicción, como se explica en la sección de Problemas. Cabe mencionar que el modo de optimización elegido, L-BFGS-B, solo admite valores finitos, por lo que el manejador no puede devolver Infinito.

```
1 aprox_params_orbit <- function(dataset, fecha, num_puntos, Bstar){
2
3   puntos_ant <- dataset %>% filter(ephemerisUTCTime<fecha) %>% slice_max
      (order_by=ephemerisUTCTime, n=num_puntos/2)
4   punto_fecha <- dataset %>% filter(ephemerisUTCTime==fecha)
5   puntos_post <- dataset %>% filter(ephemerisUTCTime>fecha) %>% slice_
      min(order_by=ephemerisUTCTime, n=num_puntos/2)
6   puntos <- rbind(puntos_ant, punto_fecha, puntos_post)
7   puntos <- arrange(puntos, ephemerisUTCTime)
8
```

```r
punto_base <- punto_fecha

params <- get_params_orbitales(dateTime = punto_base$ephemerisUTCTime,
 pos = c(punto_base$X, punto_base$Y, punto_base$Z), vel = c(punto_base$
 dX, punto_base$dY, punto_base$dZ))
params <- c(params, Bstar)
cotas <- get_cotas_params_orbit(params=params)
cotas_inf <- cotas$cotas_inf
cotas_sup <- cotas$cotas_sup

calcula_error_aprox_try_catch <- function(params){
  error_prediccion <- 0
  tryCatch(
    {
      for (i in 1:nrow(puntos)) {
        punto <- puntos[i,]
        point_TEME <- sgdp4(n0=params[1],
                            e0=params[2],
                            i0=params[3],
                            M0=params[4],
                            omega0=params[5],
                            OMEGA0=params[6],
                            Bstar=params[7],
                            initialDateTime=punto_base$ephemerisUTCTime
 ,
                            targetTime=punto$ephemerisUTCTime
        )
        point_GCRF <- TEMEtoGCRF(point_TEME$position*1000, point_TEME$
 velocity*1000, punto$ephemerisUTCTime)
        dif <- c(punto$X, punto$Y, punto$Z) - point_GCRF$position
        error_prediccion <- error_prediccion + sum(dif*dif)^0.5
      }
      return(error_prediccion)
    },
    error=function(e){
      return(9999999999)
    },
    warning=function(w){
      return(9999999999)
    }
  )
}
optim_params <- optim(par=params,
                      fn=calcula_error_aprox_try_catch,
                      gr=NULL, method="L-BFGS-B",
                      lower=cotas_inf,
                      upper=cotas_sup)
```

```
  return(optim_params$par)
}
```

Extracto de código 7.8: Optimización parámetros orbitales

7.2.5. Representación de órbitas

Para comprobar que la optimización de los parámetros funcionaba bien se implementó una función que puede representar hasta tres órbitas. De este modo se podía utilizar para comprobar que la optimización estaba teniendo lugar adecuadamente. Se utilizará el paquete *plotly*.

```
dibuja_trayectorias <- function(posiciones_A, posiciones_B, posiciones_C)
    {

  data_A <- data.frame(posiciones_A$X, posiciones_A$Y, posiciones_A$Z)

  fig <- plot_ly(data_A, x = ~X, y = ~Y, z = ~Z, type = 'scatter3d', mode
     = 'lines',
                 opacity = 1, line = list(color = 'blue', width = 3,
    reverscale = FALSE),
                 name="Trayectoria A")

  if(!is.null(posiciones_B)){
    data_B <- data.frame(posiciones_B$X, posiciones_B$Y, posiciones_B$Z)
    fig <- fig %>% add_trace(data_B, x = ~x_pred_GCRF, y = ~y_pred_GCRF,
     z = ~z_pred_GCRF,
                             type = 'scatter3d', mode = 'lines', opacity
    = 1,
                             line = list(color = 'red', width = 3,
    reverscale = FALSE),
                             name="Trayectoria B")
  }
  if(!is.null(posiciones_B)){
    data_C <- data.frame(posiciones_C$X, posiciones_C$Y, posiciones_C$Z)
    fig <- fig %>% add_trace(data_B, x = ~x_pred_GCRF, y = ~y_pred_GCRF,
     z = ~z_pred_GCRF,
                             type = 'scatter3d', mode = 'lines', opacity
    = 1,
                             line = list(color = 'green', width = 3,
    reverscale = FALSE),
                             name="Trayectoria C")
  }
  return(fig)
}
```

Extracto de código 7.9: Representación de gráficas

7.2.6. Creación del primer *dataset*

La función que se implementó devolvía un *dataset* que incluye las posiciones y las velocidades de las predicciones realizadas por HPOP y SGDP4 a partir de una fecha inicial en una serie de tiempos. Sin embargo, debido al coste computacional que tiene ejecutar HPOP, se vió que no resultaba viable generar los datos de esta forma y se decidió utilizar los ficheros de datos RINEX que, además, contienen las posiciones y velocidades exactas, salvo pequeños errores de observación.

7.2.7. Obtención de datos RINEX

Para la obtención de los ficheros de datos RINEX se realizó *web scrapping*, con la librería `BeautifulSoup` en Python, ya que había multitud de ficheros y descargarlos a mano no era factible. A continuación se muestra la estructura de directorios del repositorio del Instituto de Estadística y Cartografía de la Junta de Andalucía. Dentro de cada fichero ZIP se encuentran cuatro archivos: un archivo html especificando las características de la antena receptora de las señales, el archivo de datos meteorológicos y los archivos de observaciones GPS y GLONASS.

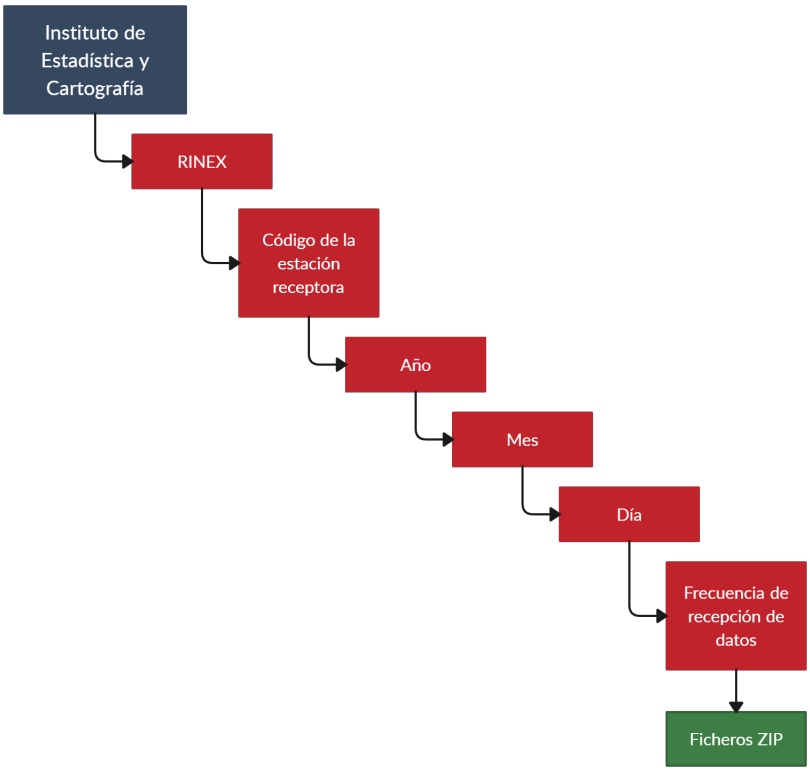

Figura 7.1: Estructura de directorios del repositorio de RINEX versión 2 de la Junta de Andalucía

```
1 import requests
2 from bs4 import BeautifulSoup
```

```
from datetime import date, timedelta
from zipfile import ZipFile
from tqdm import tqdm
from pathlib import Path
from os import rename, remove

main_path = Path(__file__).parent.resolve()

year = 2022
start_date = date(year, 1, 1)
end_date = date(year+1, 1, 1)

BASE_URL = "https://www.juntadeandalucia.es/
    institutodeestadisticaycartografia/rinex/CRDB"

def find_zip_link_in_url(url:str):
    reqs = requests.get(url)
    soup = BeautifulSoup(reqs.text, 'html.parser')

    for link in soup.find_all('a'):
        u = link.get('href')
        if u[-4:] == ".zip":
            return u

def daterange(start_date, end_date):
    for n in range(int((end_date - start_date).days)):
        yield start_date + timedelta(n)

for date in tqdm(daterange(start_date, end_date)):
    day = str(date.day)
    month = str(date.month)

    day = f"0{day}" if len(day) == 1 else day
    month = f"0{month}" if len(month) == 1 else month

    url = f"{BASE_URL}/{year}/{month}/{day}/24H_30seg"

    t = find_zip_link_in_url(url)

    if(t is None):
        with open(f"{main_path}/exceptions.txt", "a") as excep_file:
            excep_file.write(f"problem with date: {day}/{month}/{year},
    url is: {url} \n")
        continue

    r = requests.get(t, allow_redirects=True)
    zip_dest = f'./temp/{day}-{month}-{year}.zip'
```

```
48      open(zip_dest, 'wb').write(r.content)
49
50      with ZipFile(zip_dest, 'r') as zip:
51          filelist = zip.namelist()
52          for f in filelist:
53              if f[-1:] == "g":
54                  zip.extract(f, f"{main_path}/downloads")
55                  rename(f"{main_path}/downloads/{f}", f"{main_path}/
    downloads/{day}-{month}-{year}.g")
56
57              elif f[-1:] == "n":
58                  zip.extract(f, f"{main_path}/downloads/")
59                  rename(f"{main_path}/downloads/{f}", f"{main_path}/
    downloads/{day}-{month}-{year}.n")
60
61      remove(zip_dest)
```

Extracto de código 7.10: Obtención ed ficheros RINEX de la Junta de Andalucía

Cabe mencionar que las cabeceras de archivos RINEX contienen datos importantes para su interpretación, tales como correcciones temporales que dependen de múltiples factores y cambian entre archivos, por lo que no se pueden unificar varios archivos en uno y deben ser almacenados y tratados por separado.

Con el código anterior, cambiando el valor de *year*, obtenemos los archivos de navegación RINEX de satélites GPS y GLONASS de la estación del Instituto de Estadística y Cartografía de la Junta de Andalucía ubicada en de Córdoba, de los años 2019 a 2022.

7.2.8. Análisis y procesado de los datos

Una vez descargados todos los archivos RINEX, debemos proceder a su análisis, con el fin de determinar cuál es el satélite del que más datos se dispone y utilizarlo para entrenar la red neuronal.

El análisis se realiza leyendo todos los archivos y contando cuántos mensajes RINEX de cada satélite hay. Debido a que las funciones de lectura del paquete devuelven valores distintos según sea GPS o GLONASS tenemos que implementar funciones distintas para cada constelación.

Primero necesitamos una función auxiliar que devuelve un vector con los códigos de los satélites que aparecen en un determinado archivo, repetidos según el número de mensajes en los que se encuentre. La función que cuente el número de mensajes por satélite hará uso de lo anterior.

```
1 get_satellites_GPS_archivo <- function(rinexData){
2   count <- 0
3   prncodes <- c()
4   for (i in 1:length(rinexData$messages)) {
```

```
    prncodes <- c(prncodes, rinexData$messages[[i]]$satellitePRNCode)
  }
  return(prncodes)
}

get_num_satellites_GPS_ficheros <- function(ruta_base, ficheros){
  satelitesGPS <- c()
  for (f in ficheros){
    print(f)
    faux <- paste0(ruta_base, f)
    if (file.exists(faux)) {
      aux <- readGPSNavigationRINEX(faux)
      satelitesGPS <- c(satelitesGPS, get_satellites_GPS_archivo(aux))
    } else {
      print("NO EXISTE")
    }
  }
  return(table(satelitesGPS))
}
```

Extracto de código 7.11: Cálculo de número de mensajes RINEX por satélites GPS

```
get_satellites_GLONASS_archivo <- function(rinexData){
  satelliteNumbers <- c()
  for (i in 1:length(rinexData$messages)) {
    satelliteNumbers <- c(satelliteNumbers, rinexData$messages[[i]]$
   satelliteNumber)
  }
  return(satelliteNumber)
}

get_num_satellites_GLONASS_ficheros <- function(ruta_base, ficheros){
  satelitesGLONASS <- c()

  for (f in ficheros){
    print(f)
    faux <- paste0(ruta_base, f)
    if (file.exists(faux)) {
      aux <- readGLONASSNavigationRINEX(faux)
      satelitesGLONASS <- c(satelitesGLONASS, get_satellites_GLONASS_
   archivo(aux))
    } else {
      print("NO EXISTE")
    }
  }
  return(table(satelitesGLONASS))
}
```

Extracto de código 7.12: Cálculo de número de mensajes RINEX por satélites

GLONASS

Después de ver que el satélite con más datos era el Kosmos 2514, de la constelación GLONASS, con código identificativo 17, se procedió a leer de nuevo todos los archivos y almacenar la posición, velocidad, aceleración y época de los mensajes de navegación RINEX. Para ello se implementó la siguiente función:

```
get_datos_GLONASS <- function(ruta_base, ficheros, codigo_satelite){
  dataset <- data.frame("X" = numeric(),
                        "Y" = numeric(),
                        "Z" = numeric(),
                        "dx" = numeric(),
                        "dy" = numeric(),
                        "dz" = numeric(),
                        "ephemerisUTCTime" = character())
  for (f in ficheros){
    print(f)
    faux <- paste0(ruta_base, f)
    if (file.exists(faux)) {
      rinexData <- readGLONASSNavigationRINEX(faux)
      for (i in 1:length(rinexData$messages)) {
        satelliteData <- rinexData$messages[[i]]
        if (satelliteData$satelliteNumber==codigo_satelite){
          position_ITRF <- c(satelliteData$positionX, satelliteData$
    positionY, satelliteData$positionZ)
          velocity_ITRF <- c(satelliteData$velocityX, satelliteData$
    velocityY, satelliteData$velocityZ)
          epoch <- as.character.Date(as.POSIXct(satelliteData$
    ephemerisUTCTime, tz="UTC"))
          coordinates_GCRF <- ITRFtoGCRF(position_ITRF, velocity_ITRF,
    epoch)
          new_row <- data.frame("X" = coordinates_GCRF$position[1]*1000,
                                "Y" = coordinates_GCRF$position[2]*1000,
                                "Z" = coordinates_GCRF$position[3]*1000,
                                "dX" = coordinates_GCRF$velocity[1]*1000,
                                "dY" = coordinates_GCRF$velocity[2]*1000,
                                "dZ" = coordinates_GCRF$velocity[3]*1000,
                                "ephemerisUTCTime" = as.character.Date(as
    .POSIXct(satelliteData$ephemerisUTCTime, tz="UTC")))
          dataset <- rbind(dataset, new_row)
        }
      }
    } else {
      print("NO EXISTE")
    }
  }
  return(dataset)
}
```

Extracto de código 7.13: Obtención de datos de satélite GLONASS por código identificativo

7.2.9. Creación del *dataset* con las predicciones

Debido a la optimización de los parámetros orbitales, se implementaron varias funciones auxiliares, que posteriormente fueron utilizadas en la función principal que genera el conjunto de datos. También se hizo uso de las librerías *dplyr* y *tidyr*. Todos los *datasets* parciales que se iban construyendo, así como el definitivo, se fueron guardando en archivos de texto, para una lectura fácil y rápida con la función *read.table*.

Debido a la gran cantidad de columnas del *dataset*, se implementó una función para crear un *dataframe* vacío, con los campos que iba a tener:

```
create_clean_df <- function(){
  return(
    data.frame("UTCTime_base" = character(),
               "unixTime_base" = numeric(),
               "X_base_real" = numeric(),
               "Y_base_real" = numeric(),
               "Z_base_real" = numeric(),
               "dX_base_real" = numeric(),
               "dY_base_real" = numeric(),
               "dZ_base_real" = numeric(),
               "time_to_predict" = numeric(),
               "UTCTime_predict" = character(),
               "unixTime_prediict" = numeric(),
               "X_predict_real" = numeric(),
               "Y_predict_real" = numeric(),
               "Z_predict_real" = numeric(),
               "dX_predict_real" = numeric(),
               "dY_predict_real" = numeric(),
               "dZ_predict_real" = numeric(),
               "X_predict_sgdp4_optim" = numeric(),
               "Y_predict_sgdp4_optim" = numeric(),
               "Z_predict_sgdp4_optim" = numeric(),
               "dX_predict_sgdp4_optim" = numeric(),
               "dY_predict_sgdp4_optim" = numeric(),
               "dZ_predict_sgdp4_optim" = numeric()
    )
  )
}
```

Extracto de código 7.14: Creación del *dataframe* base

Otra función implementada es para buscar la fecha más cercana a una dada en los archivos RINEX, dentro de un margen de error. Esto es necesario porque en dichos archivos sólo disponemos de un número finito de valores, aunque pueda ser muy grande. Así, si queremos realizar una predicción en una época concreta, lo más probable es que no dispongamos de datos de dicha época. Cuando esto ocurra, deberemos obtener los de la época más cercana disponible, siempre que esa época no sea muy lejana a la primera. La fecha que se le pasa como parámetro tiene que ser de tipo carácter, para evitar problemas con *difftime*.

```
busca_fecha <- function(dataset, fecha, margen_error_fecha){
  diferencias_tiempos <- abs(as.POSIXct(dataset$ephemerisUTCTime, tz='UTC
    ')-as.POSIXct(fecha, tz='UTC'))
  fecha_siguiente <- dataset$ephemerisUTCTime[which.min(diferencias_
    tiempos)]
  dif_fecha_siguiente <- difftime(fecha_siguiente, fecha, units="secs")
  if (dif_fecha_siguiente < margen_error_fecha) {
    return(fecha_siguiente)
  } else {
    return("MARGEN SUPERADO")
  }
}
```

Extracto de código 7.15: Búsqueda de fecha más cercana en un dataset

También se desarrolló una función para crear un *dataframe* con las predicciones realizadas con SGDP4 a partir de una sola fecha en una serie de tiempos indicados, para utilizarla después en la función para crear el conjunto de datos completo. También recibe como parámetros un *dataset* base con las datos exactos para poder optimizar los parámetros, un margen de error para la búsqueda de la fecha más cercana al tiempo de predicción, el número de puntos con los que aproximar, además del de la fecha base, y el coeficiente de rozamiento Bstar. Se ha tenido que añadir un manejador de excepciones, ya que en algunas ocasiones se llegaba a valores de los parámetros que, aunque válidos, no eran admisibles por la implementación de la función *sgdp4* de *asteRisk* para realizar predicciones a largo plazo con ellos.

```
get_df_predictions_from_one_point_sgdp4 <- function(dataset, fecha_base,
    elementos_orbitales, lista_tiempos, margen_error, puntos_aprox, Bstar)
    {

  ## fecha_base es una fecha del dataset

  #Inicializamos el dataframe a devolver
  res_df <- create_clean_df()

  #Obtenemos los datos para dicha fecha
  data_base <- dataset[dataset$ephemerisUTCTime==fecha_base,][1,]
  #Nos quedamos con la primera fila en caso de repetidos

  fechas_a_predecir <- c()
```

```
13  for (t in lista_tiempos){
14    #Calculamos la fecha en la que vamos a generar la prediccion
15    fecha_a_buscar <- as.character.Date(as.POSIXct(fecha_base, tz='UTC')+
    t)
16    margen_error_fecha <- (1+margen_error)*t
17
18    #Si dicha fecha no se encuentra en el dataset cogemos la primera
    siguiente, siempre que no se distancie mucho
19    siguiente_fecha <- busca_fecha(dataset, fecha_a_buscar, margen_error_
    fecha)
20
21    if (siguiente_fecha!="MARGEN SUPERADO") {
22      fechas_a_predecir <- c(fechas_a_predecir, siguiente_fecha)
23    }
24  }
25
26  fechas_a_predecir <- fechas_a_predecir[!duplicated(fechas_a_predecir)]
27
28  for (fecha_prediccion in fechas_a_predecir) {
29    tryCatch({
30      point_TEME <- sgdp4(n0=elementos_orbitales[1], e0=elementos_
    orbitales[2], i0=elementos_orbitales[3],
31                          M0=elementos_orbitales[4], omega0=elementos_
    orbitales[5],
32                          OMEGA0=elementos_orbitales[6], Bstar=elementos_
    orbitales[7],
33                          initialDateTime=fecha_base, targetTime=fecha_
    prediccion)
34      point_GCRF <- TEMEtoGCRF(point_TEME$position*1000, point_TEME$
    velocity*1000, fecha_prediccion)
35    },
36    error=function(e){
37      point_GCRF <- list(NA, NA)
38      names(point_GCRF) <- c("position", "velocity")
39    },
40    warning=function(w){
41      point_GCRF <- list(NA, NA)
42      names(point_GCRF) <- c("position", "velocity")
43    })
44
45    data_predicted_real <- dataset[dataset$ephemerisUTCTime==fecha_
    prediccion, ]
46
47    new_row <- data.frame("ephemerisUTCTime_base" = fecha_base,
48                          "unixTime_base" = as.numeric(as.POSIXct(fecha_
    base, tz="UTC")),
49                          "X_base_real" = data_base$X,
```

```
                            "Y_base_real" = data_base$Y,
                            "Z_base_real" = data_base$Z,
                            "dX_base_real" = data_base$dX,
                            "dY_base_real" = data_base$dY,
                            "dZ_base_real" = data_base$dZ,
                            "time_to_predicts_mins" = abs(as.POSIXct(fecha_
    base, tz='UTC')-as.POSIXct(fecha_prediccion, tz='UTC')),
                            "ephemerisUTCTime_predict" = fecha_prediccion,
                            "unixTime_predict" = as.numeric(as.POSIXct(
    fecha_prediccion, tz="UTC")),
                            "X_predict_real" = data_predicted_real$X,
                            "Y_predict_real" = data_predicted_real$Y,
                            "Z_predict_real" = data_predicted_real$Z,
                            "dX_predict_real" = data_predicted_real$dX,
                            "dY_predict_real" = data_predicted_real$dY,
                            "dZ_predict_real" = data_predicted_real$dZ,
                            "X_predict_sgdp4_optim" = point_GCRF$position[1
    ],
                            "Y_predict_sgdp4_optim" = point_GCRF$position[2
    ],
                            "Z_predict_sgdp4_optim" = point_GCRF$position[3
    ],
                            "dX_predict_sgdp4_optim" = point_GCRF$velocity[
    1],
                            "dY_predict_sgdp4_optim" = point_GCRF$velocity[
    2],
                            "dZ_predict_sgdp4_optim" = point_GCRF$velocity[
    3]
     )
     res_df <- rbind(res_df, new_row)
  }
  return(res_df)
}
```

Extracto de código 7.16: Creación de dataset a partir de una única fecha

Con las funciones anteriores ya podemos implementar la función que genera el conjunto de datos a partir de los parámetros de entrada iniciales, que son el *dataset* de las posiciones (obtenido de los ficheros RINEX), un número de puntos que indica cuántas fechas se tomarán como base para realizar las predicciones en los periodos indicados en la lista de tiempos, un margen de error para la búsqueda de las fechas, el número de puntos empleado en la aproximación y el coeficiente de rozamiento, que por defecto estará a 0 (ya que no lo conocemos en la mayoría de los casos).

```
get_df_predictions_sgdp4 <- function(dataset, num_puntos, lista_tiempos,
    margen_error, puntos_aprox, Bstar=0){

  puntos_aleatorios <- sample(1:nrow(dataset),  num_puntos, replace=FALSE
    )
```

```
  fechas_base <- dataset[puntos_aleatorios,]$ephemerisUTCTime

  res_df <- create_clean_df()

  for (i in 1:length(fechas_base)) {
    print(paste("Punto", i, "de", num_puntos))
    fecha <- fechas_base[i]
    print(fecha)
    elementos_orbitales <- aprox_params_orbit(dataset=dataset,
                                              fecha=fecha,
                                              num_puntos=puntos_aprox,
                                              Bstar=0)
    res_parcial <- get_df_predictions_from_one_point_sgdp4(dataset =
   dataset, fecha_base = fecha, elementos_orbitales = elementos_orbitales
    , lista_tiempos = lista_tiempos, margen_error = margen_error, puntos_
    aprox = puntos_aprox, Bstar = Bstar)
    res_df <- rbind(res_df, res_parcial)
  }
  return(res_df)
}
```

Extracto de código 7.17: Creación de dataset

El valor por defecto del parámetro Bstar se ha establecido a 0, ya que en la mayoría de los casos no vamos a conocerlo y así, al optimizar se acercará al valor real.

Es posible que en el conjunto de datos generado con la función anterior aparezca algún *NA*, por lo explicado anteriormente. Se decidió guardar esas observaciones faltantes por si en algún momento se quiere realizar un análisis acerca de cuándo falla la función del paquete. Por ejemplo, podría interesar estimar cuál debería ser la lista de tiempos a predecir para maximizar el número de datos sin que haya problemas con la implementación del modelo SGDP4. De todos modos, para eliminar las filas con observaciones faltantes, bastó con utilizar la función *na.omit* de R.

Cabe mencionar que se almacenó en el *dataset* un campo *time_to_predict*, que indica el tiempo de diferencia con el que se buscó esa fecha en el conjunto de datos anterior. La finalidad de esta variable es por si hiciera falta depurar algo.

7.2.10. Adición de las predicciones con parámetros sin optimizar

Las predicciones con los parámetros optimizados reducen el error a medio plazo, ya que se ajustan los parámetros a un fragmento de órbita más extensa, pero lo aumentan a corto, lo cual puede ser señal de que la órbita se modifica demasiado al cambiar los parámetros. Por tanto, también se incluyeron en el *dataset* las posiciones predichas por SGDP4 con los parámetros sin optimizar, para evaluar el comportamiento de la red neuronal.

```r
add_sgdp4_sin_optimizar <- function(data, Bstar=0){

  data$X_predict_sgdp4 <- NA
  data$Y_predict_sgdp4 <- NA
  data$Z_predict_sgdp4 <- NA
  data$dX_predict_sgdp4 <- NA
  data$dY_predict_sgdp4 <- NA
  data$dZ_predict_sgdp4 <- NA

  for (i in 1:nrow(data)) {
    print(i)
    fecha_base <- data$ephemerisUTCTime_base[i]
    fecha_prediccion <- data$ephemerisUTCTime_predict[i]

    pos <- c(data$X_base_real[i], data$Y_base_real[i], data$Z_base_real[i])
    vel <- c(data$dX_base_real[i], data$dY_base_real[i], data$dZ_base_real[i])

    elementos_orbitales <- get_params_orbitales(fecha_base, pos, vel)

    point_TEME <- sgdp4(n0=elementos_orbitales[1], e0=elementos_orbitales[2], i0=elementos_orbitales[3],
                        M0=elementos_orbitales[4], omega0=elementos_orbitales[5],
                        OMEGA0=elementos_orbitales[6], Bstar=0,
                        initialDateTime=fecha_base, targetTime=fecha_prediccion)
    point_GCRF <- TEMEtoGCRF(point_TEME$position*1000, point_TEME$velocity*1000, fecha_prediccion)

    data$X_predict_sgdp4[i] <- point_GCRF$position[1]
    data$Y_predict_sgdp4[i] <- point_GCRF$position[2]
    data$Z_predict_sgdp4[i] <- point_GCRF$position[3]
    data$dX_predict_sgdp4[i] <- point_GCRF$velocity[1]
    data$dY_predict_sgdp4[i] <- point_GCRF$velocity[2]
    data$dZ_predict_sgdp4[i] <- point_GCRF$velocity[3]
  }
  return(data)
}
```

Extracto de código 7.18: Añadir predicciones SGDP4 sin optimizar parámetros

Posteriormente a la generación de los datos, estos fueron almacenados en un archivo de texto con el comando *write.table(datos, file = "traintest.txt", row.names = FALSE)*.

7.2.11. Instalación de TensorFlow y Keras

Para poder entrenar redes neuronales con R, necesitamos las librerías TensorFlow y Keras instaladas en el equipo. En R utilizaremos una librería que utiliza la interfaz para Python *reticulate* para aprovechar los paquetes anteriores.

Pese a que la librería de TensorFlow para R tiene una función que instala Python y TensorFlow, se producen errores. Procedemos a instalarlos por nuestra cuenta en el equipo. Una vez hecho esto ya podemos instalar las librerías en R.

```
1 install.packages("tensorflow")
2 install.packages("keras")
```

Extracto de código 7.19: Instalación de TensorFlow y Keras en R

7.2.12. Instalación de CUDA y cuDNN

Si queremos reducir los tiempos de entrenamiento drásticamente, debemos aprovechar la potencia de las tarjetas gráficas. Por ello, al ser la gráfica del equipo de la marca Nvidia, se instalaron CUDA (Compute Unified Device Architecture) y cuDNN (CUDA Deep Neural Network), poniendo especial atención a que las versiones fueran estables y adecuadas para todos los componentes del equipo.

7.2.13. Implementación de Robust Scaler

Para normalizar fue escogida la técnica de escalado robusto. Después de buscar entre los paquetes de R uno que contara con una implementación del mismo y no encontrarlo, se decidió realizar la implementación. De esta forma, además, disponemos de la mediana y el IQR de los datos de entrenamiento para normalizar los de test y otros datos e incluso desnormalizarlos después para analizar los resultados. El código de la normalización se muestra con el entrenamiento de los modelos, en la sección 7.2.14.

7.2.14. Entrenamiento de los primeros modelos

A continuación se mostrarán las partes del *script* con el que se entrenaron los modelos. Será código general, que se puede modificar de modo sencillo para cambiar los parámetros de entrada. Los pasos para implementar y entrenar son los siguientes:

1. Lectura de datos y adición de errores
2. Partición en datos de entrenamiento y test
3. Selección del campos del *dataset* para la entrada del modelo
4. Normalización
5. Creación del modelo
6. Compilación y entrenamiento del modelo

Lectura de datos y adición de errores

En primer lugar hubo que leer los datos del archivo de texto en el que se guardó el *dataset*, "traintest.txt", y añadir los errores de predicción en cada coordenada, tal y como se explicó en la parte de diseño.

```
#Lectura de datos
datos <- read.table("traintest.txt", header=TRUE)
```

Extracto de código 7.20: Lectura del dataset

```
datos$error_X_predict <- datos$X_predict_real-datos$X_predict_sgdp4
datos$error_Y_predict <- datos$Y_predict_real-datos$Y_predict_sgdp4
datos$error_Z_predict <- datos$Z_predict_real-datos$Z_predict_sgdp4
```

Extracto de código 7.21: Adición de errores de la predicción con parámetros sin optimizar

```
datos$error_X_predict_optim <- datos$X_predict_real-datos$X_predict_sgdp4
    _optim
datos$error_Y_predict_optim <- datos$Y_predict_real-datos$Y_predict_sgdp4
    _optim
datos$error_Z_predict_optim <- datos$Z_predict_real-datos$Z_predict_sgdp4
    _optim
```

Extracto de código 7.22: Adición de errores de la predicción con parámetros optimizados

También se entrenaron algunos modelos que recibían el movimiento predicho, es decir, la diferencia entre la posición predicha y la posición inicial, en lugar de la posición. Para ellos había que incluir uno de los siguientes códigos, según estemos considerando las predicciones con los parámetros optimizados o sin optimizar:

```
datos$mov_X_sgdp4 <- datos$X_predict_sgdp4 - datos$X_base_real
datos$mov_Y_sgdp4 <- datos$Y_predict_sgdp4 - datos$Y_base_real
datos$mov_Z_sgdp4 <- datos$Z_predict_sgdp4 - datos$Z_base_real
```

Extracto de código 7.23: Adición de los movimientos predichos con parámetros sin optimizar

```
datos$mov_X_sgdp4_optim <- datos$X_predict_sgdp4_optim - datos$X_base_
    real
datos$mov_Y_sgdp4_optim <- datos$Y_predict_sgdp4_optim - datos$Y_base_
    real
datos$mov_Z_sgdp4_optim <- datos$Z_predict_sgdp4_optim - datos$Z_base_
    real
datos$tiempo_a_predecir <- datos$unixTime_predict-datos$unixTime_base
```

Extracto de código 7.24: Adición de los movimientos predichos con parámetros optimizados

Se añadió una variable auxiliar que fue empleada posteriormente para estratificar la división del conjunto.

```
datos$tiempo_a_predecir <- datos$unixTime_predict-datos$unixTime_base
```

Extracto de código 7.25: Adición de la variable utilizada para estratificar

Partición en datos de entrenamiento y test y selección de campos

La división en conjunto de entrenamiento y de test se realizó con el paquete *rsample*, tomando un 50 % para cada uno y estratificando según la variable *tiempo_a_predecir*, que se corresponde con la diferencia entre la fecha de la predicción y la fecha base. Esta división se guardó para entrenar y validar todos los modelos con la misma separación del conjunto.

```
library(rsample)
id_partition <- initial_split(data = datos, prop = 0.5, strata=tiempo_a_
    predecir, breaks = 5)
df_train <- training(id_partition)
write.table(df_train, file = "conjuntoentrenamiento.txt", row.names =
    FALSE)
df_test <- testing(id_partition)
write.table(df_train, file = "conjuntotest.txt", row.names = FALSE)
```

Extracto de código 7.26: Separación estratificada en entrenamiento y test

El *dataframe* resultante tuvo que ser convertido a una matriz sin nombres de columnas, para que la transformación a tensor que implementa TensorFlow tuviera lugar satisfactoriamente.

Para seleccionar los campos del *dataset* que se iban a incluir en el modelo se utilizó la librería *dplyr*, por la legibilidad del código resultante de utilizarla. Se muestra el código de un ejemplo concreto.

```
library(rsample)

df_test <- read.table("conjuntoentrenamiento.txt", header=TRUE)

x_train <- select(df_train,unixTime_base,
                  X_base_real,
                  Y_base_real,
                  Z_base_real,
                  dX_base_real,
                  dY_base_real,
                  dZ_base_real,
                  unixTime_predict,
                  X_predict_sgdp4,
                  Y_predict_sgdp4,
                  Z_predict_sgdp4
                  )
x_train <- as.matrix(unname(x_train))
```

```
y_train <- select(df_train, error_X_predict, error_Y_predict, error_Z_
    predict)
y_train <- as.matrix(unname(y_train))


df_test <- read.table("conjuntotest.txt", header=TRUE)

x_test <- select(df_train,unixTime_base,
                  X_base_real,
                  Y_base_real,
                  Z_base_real,
                  dX_base_real,
                  dY_base_real,
                  dZ_base_real,
                  unixTime_predict,
                  X_predict_sgdp4,
                  Y_predict_sgdp4,
                  Z_predict_sgdp4
                  )
x_test <- as.matrix(unname(x_test))
y_test <- select(df_test, error_X_predict, error_Y_predict, error_Z_
    predict)
y_test <- as.matrix(unname(y_test))
```

Extracto de código 7.27: División en conjuntos de entrenamiento y test

En el caso de que el modelo sea para la predicción con los parámetros optimizados, error_X_predict deberá cambiarse por error_X_predict_optim y lo mismo con las coordenadas Y y Z.

Normalización

Tal y como se explicó en la parte de diseño, la normalización elegida fue el escalado robusto y tuvo que implementarse debido a que no se encontró ningún paquete adecuado.

```
#Normalizacion con Robust Scaler
#Los parametros se obtienen del conjunto de entrenamiento
iqr_x <- apply(x_train, 2, IQR)
iqr_y <- apply(y_train, 2, IQR)

mediana_x <- apply(x_train, 2, median)
mediana_y <- apply(y_train, 2, median)

#Normalizamos el conjunto de entrenamiento
x_train_scaled <- matrix(nrow=nrow(x_train), ncol=ncol(x_train))
y_train_scaled <- matrix(nrow=nrow(y_train), ncol=ncol(y_train))
for (i in 1:ncol(x_train)) {
```

```
  x_train_scaled[,i] <- (x_train[,i]-mediana_x[i]) / iqr_x[i]
}
for (i in 1:ncol(y_train)) {
  y_train_scaled[,i] <- (y_train[,i]-mediana_y[i]) / iqr_y[i]
}

#Normalizamos el conjunto de test
x_test_scaled <- matrix(nrow=nrow(x_test), ncol=ncol(x_test))
y_test_scaled <- matrix(nrow=nrow(y_test), ncol=ncol(y_test))
for (i in 1:ncol(x_test)) {
  x_test_scaled[,i] <- (x_test[,i]-mediana_x[i]) / iqr_x[i]
}
for (i in 1:ncol(y_test)) {
  y_test_scaled[,i] <- (y_test[,i]-mediana_y[i]) / iqr_y[i]
}
```

Extracto de código 7.28: Normalización con escalado robusto

Creación y entrenamiento de modelos

Como se indicó en la sección de diseño, las capas de los modelos van a ser las siguientes:

- La capa de entrada con tantas neuronas como campos del *dataset* hallamos elegido.
- Tres capas ocultas densamente conectadas, con 256, 128 y 64 neuronas, respectivamente, y función de activación *relu*.
- La capa de salida con 3 neuronas y sin función de activación (es decir, la identidad). La salida de estas neuronas serán la estimación del error de la predicción en cada una de las coordenadas X, Y y Z.

```
num_neuronas_entrada <- ncol(x_train)
#Modelo
model <- keras_model_sequential() %>%
  layer_flatten(name="input",
                input_shape = num_neuronas_entrada) %>%
  layer_dense(name="dense_1",
              units = 256,
              activation = "relu") %>%
  layer_dense(name="dense_2",
              units = 128,
              activation = "relu") %>%
  layer_dense(name="dense_3",
              units = 64,
              activation = "relu") %>%
  layer_dense(name="output",
              units=3)
```

```
17
18 summary(model)
```

Extracto de código 7.29: Creación del modelo y resumen del mismo

Para el entrenamiento de los mismos se utilizaron un tamaño de *batch* de 128 y el número de *epochs* será de 100, después de haber realizado pruebas con otras combinaciones de parámetros y observar que esta era la que mejor funcionaba. La función de pérdida escogida fue el error cuadrático medio, mientras que la métrica fue el error absoluto medio.

```
1 #Compilacion del modelo
2 model %>%
3   compile(
4     optimizer = "adam",
5     loss = "mse", #"mean_squared_error",
6     metrics = "mae" #"mean_absolute_error"
7   )
8
9 #Entrenamiento del modelo
10 history <- model %>%
11   fit(
12     x = x_train_scaled,
13     y = y_train_scaled,
14     batch_size = 128,
15     epochs = 100,
16     verbose = 2
17   )
```

Extracto de código 7.30: Compilación y entrenamiento del modelo

7.2.15. Obtención de los datos de los astros

Como se comentó en la parte de diseño, se utiliza la función *JPLephemerides* del paquete *asteRisk*. El procedimiento es muy similar al de la generación de los otros conjuntos de datos.

Como se puede observar no se incluyeron los datos de la Tierra, puesto que al estar utilizando el sistema de referencia GCRF, las coordenadas de la posición y las componentes de la velocidad son todas 0.

```
1 datos <- read.table("traintest.txt", header=TRUE)
2
3 datos_astros <- data.frame()
4
5 fechas_sin_repetir <- datos$ephemerisUTCTime_base[!duplicated(datos$
    ephemerisUTCTime_base)]
6
```

```
for (fecha in fechas_sin_repetir) {
  MJD_UTC <- dateTimeToMJD(fecha, timeSystem = "UTC")
  ephemerides <- JPLephemerides(MJD_UTC, timeSystem = "UTC", centralBody=
   "Earth")

  new_row <- data.frame("UTCTime" = fecha,
                        "unixTime" = as.numeric(as.POSIXct(fecha, tz="UTC
   ")),
                        "X_Sun" = ephemerides$positionSun[1],
                        "Y_Sun" = ephemerides$positionSun[2],
                        "Z_Sun" = ephemerides$positionSun[3],
                        "dX_Sun" = ephemerides$velocitySun[1],
                        "dY_Sun" = ephemerides$velocitySun[2],
                        "dZ_Sun" = ephemerides$velocitySun[3],
                        "ddX_Sun" = ephemerides$accelerationSun[1],
                        "ddY_Sun" = ephemerides$accelerationSun[2],
                        "ddZ_Sun" = ephemerides$accelerationSun[3],
                        "X_SunSSBarycentric" = ephemerides$
   positionSunSSBarycentric[1],
                        "Y_SunSSBarycentric" = ephemerides$
   positionSunSSBarycentric[2],
                        "Z_SunSSBarycentric" = ephemerides$
   positionSunSSBarycentric[3],
                        "dX_SunSSBarycentric" = ephemerides$
   velocitySunSSBarycentric[1],
                        "dY_SunSSBarycentric" = ephemerides$
   velocitySunSSBarycentric[2],
                        "dZ_SunSSBarycentric" = ephemerides$
   velocitySunSSBarycentric[3],
                        "ddX_SunSSBarycentric" = ephemerides$
   accelerationSunSSBarycentric[1],
                        "ddY_SunSSBarycentric" = ephemerides$
   accelerationSunSSBarycentric[2],
                        "ddZ_SunSSBarycentric" = ephemerides$
   accelerationSunSSBarycentric[3],
                        "X_Mercury" = ephemerides$positionMercury[1],
                        "Y_Mercury" = ephemerides$positionMercury[2],
                        "Z_Mercury" = ephemerides$positionMercury[3],
                        "dX_Mercury" = ephemerides$velocityMercury[1],
                        "dY_Mercury" = ephemerides$velocityMercury[2],
                        "dZ_Mercury" = ephemerides$velocityMercury[3],
                        "ddX_Mercury" = ephemerides$accelerationMercury[1
   ],
                        "ddY_Mercury" = ephemerides$accelerationMercury[2
   ],
                        "ddZ_Mercury" = ephemerides$accelerationMercury[3
   ],
```

```
                    "X_Venus" = ephemerides$positionVenus[1],
                    "Y_Venus" = ephemerides$positionVenus[2],
                    "Z_Venus" = ephemerides$positionVenus[3],
                    "dX_Venus" = ephemerides$velocityVenus[1],
                    "dY_Venus" = ephemerides$velocityVenus[2],
                    "dZ_Venus" = ephemerides$velocityVenus[3],
                    "ddX_Venus" = ephemerides$accelerationVenus[1],
                    "ddY_Venus" = ephemerides$accelerationVenus[2],
                    "ddZ_Venus" = ephemerides$accelerationVenus[3],
                    "X_Mars" = ephemerides$positionMars[1],
                    "Y_Mars" = ephemerides$positionMars[2],
                    "Z_Mars" = ephemerides$positionMars[3],
                    "dX_Mars" = ephemerides$velocityMars[1],
                    "dY_Mars" = ephemerides$velocityMars[2],
                    "dZ_Mars" = ephemerides$velocityMars[3],
                    "ddX_Mars" = ephemerides$accelerationMars[1],
                    "ddY_Mars" = ephemerides$accelerationMars[2],
                    "ddZ_Mars" = ephemerides$accelerationMars[3],
                    "X_Jupiter" = ephemerides$positionJupiter[1],
                    "Y_Jupiter" = ephemerides$positionJupiter[2],
                    "Z_Jupiter" = ephemerides$positionJupiter[3],
                    "dX_Jupiter" = ephemerides$velocityJupiter[1],
                    "dY_Jupiter" = ephemerides$velocityJupiter[2],
                    "dZ_Jupiter" = ephemerides$velocityJupiter[3],
                    "ddX_Jupiter" = ephemerides$accelerationJupiter[1
    ],
                    "ddY_Jupiter" = ephemerides$accelerationJupiter[2
    ],
                    "ddZ_Jupiter" = ephemerides$accelerationJupiter[3
    ],
                    "X_Saturn" = ephemerides$positionSaturn[1],
                    "Y_Saturn" = ephemerides$positionSaturn[2],
                    "Z_Saturn" = ephemerides$positionSaturn[3],
                    "dX_Saturn" = ephemerides$velocitySaturn[1],
                    "dY_Saturn" = ephemerides$velocitySaturn[2],
                    "dZ_Saturn" = ephemerides$velocitySaturn[3],
                    "ddX_Saturn" = ephemerides$accelerationSaturn[1],
                    "ddY_Saturn" = ephemerides$accelerationSaturn[2],
                    "ddZ_Saturn" = ephemerides$accelerationSaturn[3],
                    "X_Uranus" = ephemerides$positionUranus[1],
                    "Y_Uranus" = ephemerides$positionUranus[2],
                    "Z_Uranus" = ephemerides$positionUranus[3],
                    "dX_Uranus" = ephemerides$velocityUranus[1],
                    "dY_Uranus" = ephemerides$velocityUranus[2],
                    "dZ_Uranus" = ephemerides$velocityUranus[3],
                    "ddX_Uranus" = ephemerides$accelerationUranus[1],
                    "ddY_Uranus" = ephemerides$accelerationUranus[2],
```

```
                        "ddZ_Uranus" = ephemerides$accelerationUranus[3],
                        "X_Neptune" = ephemerides$positionNeptune[1],
                        "Y_Neptune" = ephemerides$positionNeptune[2],
                        "Z_Neptune" = ephemerides$positionNeptune[3],
                        "dX_Neptune" = ephemerides$velocityNeptune[1],
                        "dY_Neptune" = ephemerides$velocityNeptune[2],
                        "dZ_Neptune" = ephemerides$velocityNeptune[3],
                        "ddX_Neptune" = ephemerides$accelerationNeptune[1
    ],
                        "ddY_Neptune" = ephemerides$accelerationNeptune[2
    ],
                        "ddZ_Neptune" = ephemerides$accelerationNeptune[3
    ],
                        "X_Pluto" = ephemerides$positionPluto[1],
                        "Y_Pluto" = ephemerides$positionPluto[2],
                        "Z_Pluto" = ephemerides$positionPluto[3],
                        "dX_Pluto" = ephemerides$velocityPluto[1],
                        "dY_Pluto" = ephemerides$velocityPluto[2],
                        "dZ_Pluto" = ephemerides$velocityPluto[3],
                        "ddX_Pluto" = ephemerides$accelerationPluto[1],
                        "ddY_Pluto" = ephemerides$accelerationPluto[2],
                        "ddZ_Pluto" = ephemerides$accelerationPluto[3],
                        "X_Moon" = ephemerides$positionMoon[1],
                        "Y_Moon" = ephemerides$positionMoon[2],
                        "Z_Moon" = ephemerides$positionMoon[3],
                        "dX_Moon" = ephemerides$velocityMoon[1],
                        "dY_Moon" = ephemerides$velocityMoon[2],
                        "dZ_Moon" = ephemerides$velocityMoon[3],
                        "ddX_Moon" = ephemerides$accelerationMoon[1],
                        "ddY_Moon" = ephemerides$accelerationMoon[2],
                        "ddZ_Moon" = ephemerides$accelerationMoon[3],
                        "lunar_libration_Phi" = ephemerides$
    lunarLibrationAngles[1],
                        "lunar_libration_Theta" = ephemerides$
    lunarLibrationAngles[2],
                        "lunar_libration_Psi" = ephemerides$
    lunarLibrationAngles[3],
                        "lunar_libration_dPhi" = ephemerides$
    lunarLibrationAnglesDerivatives[1],
                        "lunar_libration_dTheta" = ephemerides$
    lunarLibrationAnglesDerivatives[2],
                        "lunar_libration_dPsi" = ephemerides$
    lunarLibrationAnglesDerivatives[3],
                        "lunar_libration_ddPhi" = ephemerides$
    lunarLibrationAnglesSecondDerivatives[1],
                        "lunar_libration_ddTheta" = ephemerides$
    lunarLibrationAnglesSecondDerivatives[2],
```

```
                        "lunar_libration_ddPsi" = ephemerides$
    lunarLibrationAnglesSecondDerivatives[3]
  )
  datos_astros <- rbind(datos_astros, new_row)
}

write.table(datos_astros, file = "datosastros.txt", row.names = FALSE)
```

Extracto de código 7.31: Obtención de los datos de los astros

7.2.16. Adición de los datos de los astros al *dataset*

Una vez que tenemos los datos de los astros del Sistema Solar procedemos a incluirlos en el *dataset*. El código empleado es el siguiente:

```
datos <- read.table("traintest.txt", header=TRUE)
datos_astros <- read.table("datosastros.txt", header=TRUE)

datos_ampliados <- data.frame()

for (i in 1:nrow(datos)) {
  # Un contador para tener referencia de en que punto del proceso nos
    encontramos
  if (i%%1000==0) {
    print(i)
  }
  datos_satelite <- datos[i,]
  datos_astros_satelite <- datos_astros[(datos_astros$UTCTime==datos_
    satelite$ephemerisUTCTime_base), -1]
  new_row <- cbind(datos_satelite, datos_astros_satelite)
  datos_ampliados <- rbind(datos_ampliados, new_row)
}

write.table(datos_ampliados, file = "traintestconastros.txt", row.names =
    FALSE)
```

Extracto de código 7.32: Añadir datos de los astros al dataset

7.2.17. Creación de modelos con datos de los astros

La única diferencia con la creación de los modelos sin los datos de los astros está en la selección de campos del *dataframe*. A continuación se muestra un ejemplo de selección de campos.

```
library(dplyr)
datos <- select(datos, unixTime_base,
                X_base_real,
```

```
                Y_base_real,
                Z_base_real,
                dX_base_real,
                dY_base_real,
                dZ_base_real,
                X_Sun,
                Y_Sun,
                Z_Sun,
                dX_Sun,
                dY_Sun,
                dZ_Sun,
                ddX_Sun,
                ddY_Sun,
                ddZ_Sun,
                X_Venus,
                Y_Venus,
                Z_Venus,
                dX_Venus,
                dY_Venus,
                dZ_Venus,
                ddX_Venus,
                ddY_Venus,
                ddZ_Venus,
                X_Moon,
                Y_Moon,
                Z_Moon,
                dX_Moon,
                dY_Moon,
                dZ_Moon,
                ddX_Moon,
                ddY_Moon,
                ddZ_Moon,
                lunar_libration_Phi,
                lunar_libration_Theta,
                lunar_libration_Psi,
                lunar_libration_dPhi,
                lunar_libration_dTheta,
                lunar_libration_dPsi,
                lunar_libration_ddPhi,
                lunar_libration_ddTheta,
                lunar_libration_ddPsi,
                X_Mars,
                Y_Mars,
                Z_Mars,
                dX_Mars,
                dY_Mars,
                dZ_Mars,
```

```
                ddX_Mars,
                ddY_Mars,
                ddZ_Mars,
                X_Jupiter,
                Y_Jupiter,
                Z_Jupiter,
                dX_Jupiter,
                dY_Jupiter,
                dZ_Jupiter,
                ddX_Jupiter,
                ddY_Jupiter,
                ddZ_Jupiter,
                unixTime_predict,
                X_predict_sgdp4,
                Y_predict_sgdp4,
                Z_predict_sgdp4,
                dX_predict_sgdp4,
                dY_predict_sgdp4,
                dZ_predict_sgdp4,
                error_X_predict,
                error_Y_predict,
                error_Z_predict
)
```

Extracto de código 7.33: Elección de campos del dataset con los astros

7.2.18. Evaluación de los modelos

Para la evaluación de los modelos creados se evaluaron las métricas sobre el conjunto de test. Como dichas métricas provenían de datos normalizados no eran fáciles de interpretar, por lo que se procedió a la desnormalización y a calcular el error medio de la predicción antes y después de aplicar la corrección de la red neuronal.

```
result <- model %>% evaluate(x_test, y_test)
result["mae"]
```

Extracto de código 7.34: Evaluación de métricas sobre el conjunto de test

```
predictions_normalized <- model %>% predict(x_test_scaled)
```

Extracto de código 7.35: Predicciones con la red

Recordemos que la desnormalización tiene que realizarse con los parámetros obtenidos del conjunto de entrenamiento.

```
predictions <- matrix(nrow=nrow(predictions_normalized), ncol=ncol(
    predictions_normalized))
for (i in 1:ncol(predictions_normalized)) {
```

```
  predictions[,i] <- predictions_normalized[,i]*iqr_y[i] + mediana_y[i]
}
```

Extracto de código 7.36: Desnormalización de las predicciones

Para la corrección de las predicciones de SGDP4 basta sumarle la predicción realizada por la red, desnormalizada.

```
predicciones_sin_corregir <- select(df_test, X_predict_sgdp4_optim,
                                             Y_predict_sgdp4_optim,
                                             Z_predict_sgdp4_optim)
colnames(predicciones_sin_corregir) <- c("X","Y","Z")


predicciones_corregidas <- data.frame("X" = numeric(),
                                      "Y" = numeric(),
                                      "Z" = numeric())

for (i in 1:nrow(predicciones_sin_corregir)) {
  new_row <- data.frame("X" = predicciones_sin_corregir[i,]$X + error_
    predictions_df[i,]$X,
                        "Y" = predicciones_sin_corregir[i,]$Y + error_
    predictions_df[i,]$Y,
                        "Z" = predicciones_sin_corregir[i,]$Z + error_
    predictions_df[i,]$Z)
  predicciones_corregidas <- rbind(predicciones_corregidas, new_row)
}
```

Extracto de código 7.37: Corrección de predicción SGDP4

```
real_test <- select(df_test, X_predict_real,
                             Y_predict_real,
                             Z_predict_real)
colnames(real_test) <- c("X","Y","Z")
```

Extracto de código 7.38: Obtención de posiciones reales en el conjunto de test

De este modo ya podemos hacer uso de la función que recibía dos conjuntos con posiciones y calculaba el error medio entre las pareadas.

7.3. Conclusiones

En este capítulo se han expuesto los detalles de la implementación, documentando las funciones empleadas para todas las tareas, de modo que los avances realizados puedan ser utilizados en el futuro. De este modo, para la optimización de parámetros y la creación de nuevos conjuntos de datos de satélites se podrá reutilizar el código, ahorrando tiempo y costes.

8. Pruebas y experimentos

En este capítulo se tratarán los modelos entrenados, explicando qué parámetros reciben y evaluando la validez y eficacia de su uso.

Cuando se mencione que un modelo recibe una posición, un movimiento o una velocidad, nos estaremos refiriendo a que recibe como entrada las 3 coordenadas o componentes de dicha magnitud (X, Y y Z), análogamente con la ostras magnitudes como la libración lunar. Las fechas se considerarán en formato UNIX.

Todos los modelos tuvieron como parámetros de entrada la fecha base de la predicción, la posición del satélite en dicha fecha y la fecha en la que realizar la predicción. Así, nos referimos a ellos como **parámetros base**.

Separaremos los modelos en dos grandes familias, aquellos que no reciben los datos de los astros y aquellos que sí.

Dentro de los que no reciben datos de los astros, primero consideramos los modelos que reciben la posición predicha y luego pasaremos a aquellos que reciben el movimiento. A su vez, dentro de ellos separaremos los que involucran a las predicciones con los parámetros optimizados de las que se han realizado sin optimizarlos previamente. Todos los modelos irán acompañados de una gráfica que muestra cómo ha variado el error absoluto medio (MAE) a lo largo de su entrenamiento. No se muestra la evolución de la pérdida (*loss*) durante el entrenamiento porque la función elegida fue el error cuadrático medio (MSE), que muestra un comportamiento muy similar al MAE y su interpretación menos intuitiva.

Previamente a la evaluación de los modelos comentaremos los errores medios de las predicciones antes de aplicar la corrección de las redes neuronales.

8.1. Error medio de las predicciones SGDP4

Se calculó el error medio de las predicciones realizadas con SGDP4 con respecto a la posición real. Para dicho cálculo se tuvo en cuenta los datos del conjunto de prueba, de modo que posteriormente se pudiera comparar con el de las predicciones tras corregir el error con las redes neuronales. Los resultados se recogen en la siguiente tabla:

Predicción SGDP4	**Error medio (metros)**
Sin optimizar elementos orbitales	578619
Con los elementos orbitales optimizados	25022408

Cuadro 8.1: Error medio de las predicciones antes de corregirlas con los resultados de la red

Se observa que el error tras optimizar los parámetros es superior al otro, seguramente es debido a que la optimización se realiza en el entorno del punto base de la

predicción, de modo que la órbita que mejor se ajuste a ellos no sea la que mejor lo haga a la trayectoria real, ocasionando errores muy grandes en predicciones a largo plazo.

8.2. Evaluación de los modelos sin datos de los astros

En primer lugar se analizan los modelos que reciben las predicciones con los parámetros sin optimizar.

8.2.1. Parámetros orbitales sin optimizar

Modelos que reciben la posición predicha

El primer modelo que recibe los parámetros base y la predicción realizada con SGDP4 sin optimizar los parámetros orbitales previamente. Sobre el conjunto de entrenamiento obtuvo un MAE de 13.6329, mientras que sobre el de test fue 14.39973. El error medio de la predicción de posiciones después de la corrección fue de 647040.5, que empeora la predicción.

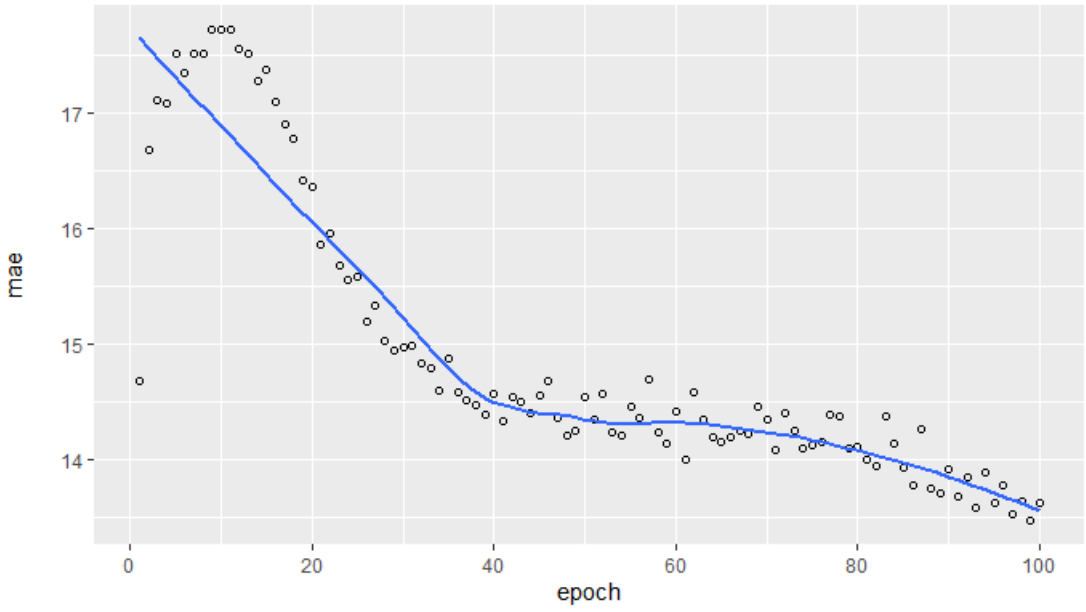

Figura 8.1: Evolución del MAE recibiendo los parámetros base y la posición predicha con los parámetros orbitales sin optimizar

Posteriormente se añadió la velocidad en el instante inicial como parámetro de entrada, obteniendo así un modelo con un MAE de 12.0608 sobre entrenamiento y 12.98055 sobre test. El error medio de la predicción de posiciones con la corrección fue

de 583126.3, que tampoco mejora la predicción, pero se nota mejoría con respecto al anterior.

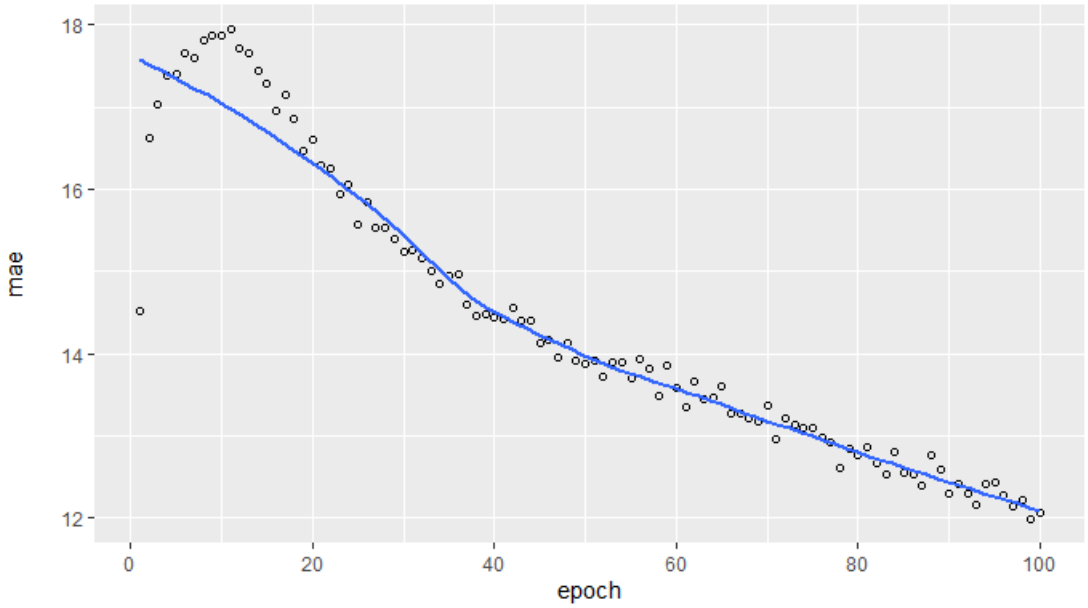

Figura 8.2: Evolución del MAE recibiendo los parámetros base, la velocidad inicial y la posición predicha con los parámetros orbitales sin optimizar

Modelos que reciben el movimiento predicho

El modelo que recibía los parámetros base junto con el movimiento predicho con parámetros sin optimizar obtuvo un MAE de 12.8487 en el conjunto de entrenamiento y de 12.28928 en el de test. El error medio de la posición con la corrección pasó a ser de 559453, con lo que conseguimos un primer modelo que mejoraba algo las predicciones.

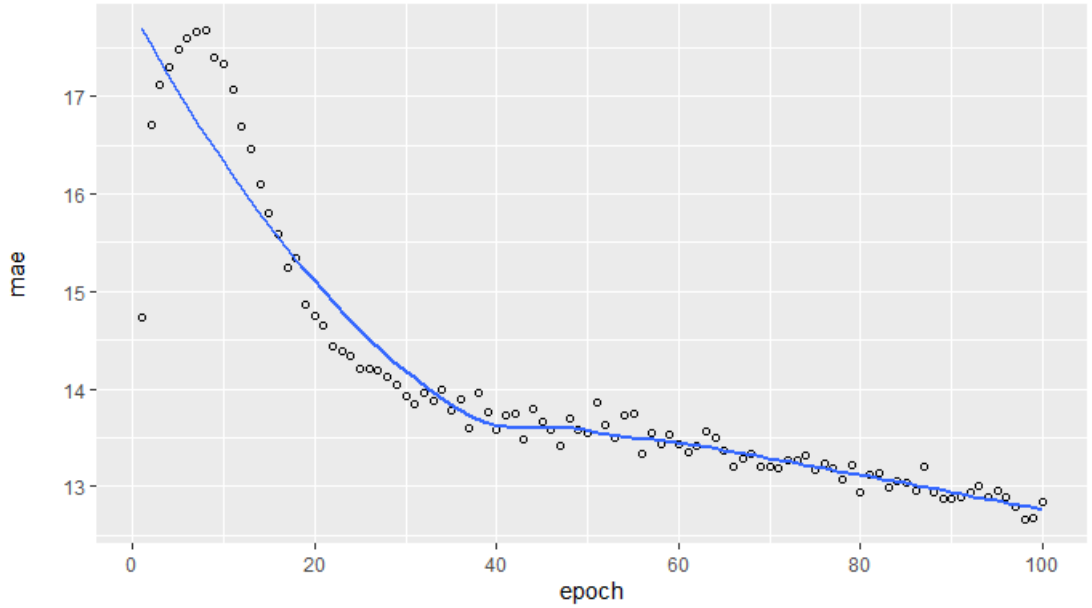

Figura 8.3: Evolución del MAE recibiendo los parámetros base y el movimiento predicho con los parámetros orbitales sin optimizar

Tras añadir la velocidad inicial, el MAE de entrenamiento fue de 13.0186 y el de test de 13.93012. El error medio de la posición con la corrección pasó a ser de 631188.6, que empeora la predicción.

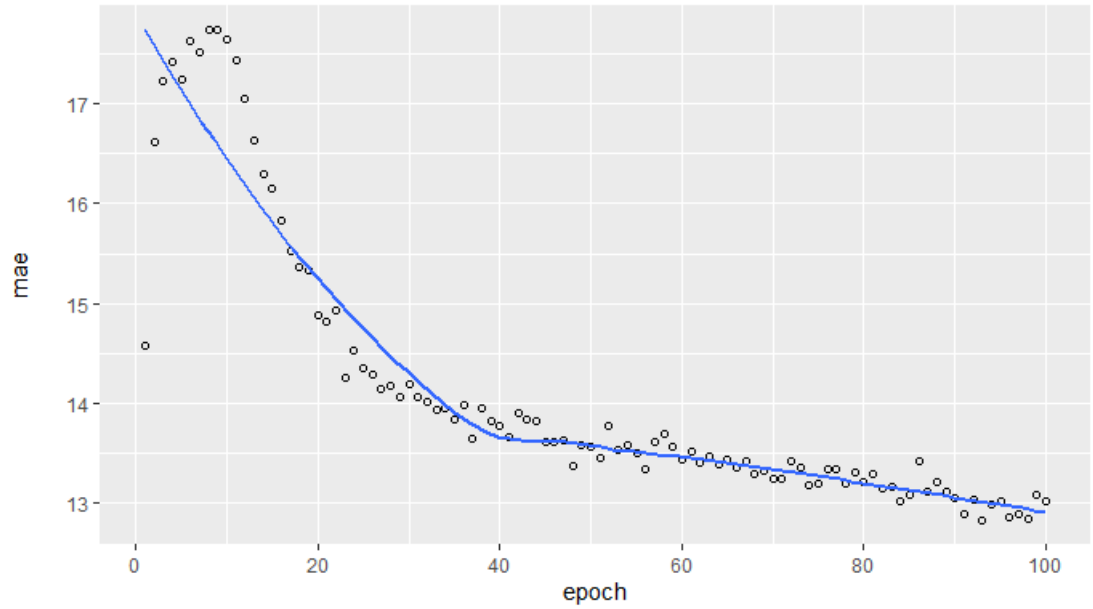

Figura 8.4: Evolución del MAE recibiendo los parámetros base, la velocidad inicial y el movimiento predicho con los parámetros orbitales sin optimizar

Agrupando estos resultados:

Entradas extra del modelo	MAE	Error medio de la predicción corregida
Posición predicha sin optimizar parámetros	14.4	647040.5
Posición predicha sin optimizar parámetros Velocidad inicial	12.98	583126.3
Movimiento predicho sin optimizar parámetros	12.29	559453
Movimiento predicho sin optimizar parámetros Velocidad inicial	13.93	631188.6

Cuadro 8.2: Comparativa de modelos para predicciones sin optimizar parámetros

8.2.2. Parámetros orbitales optimizados

Modelos que reciben la posición predicha

El primer modelo recibe los parámetros base y la predicción realizada con SGDP4 los parámetros orbitales optimizados. Sobre el conjunto de entrenamiento obtuvo un MAE de 0.3435, mientras que sobre el de test fue 0.4059753. El error medio tras corregir las posiciones fue de 17331023.

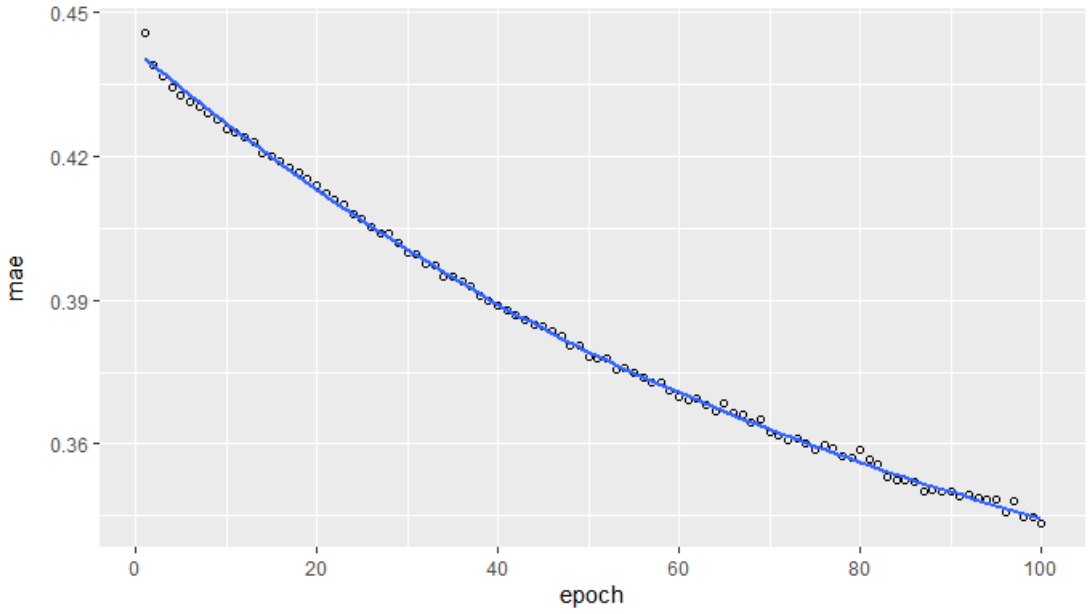

Figura 8.5: Evolución del MAE recibiendo los parámetros base y la posición predicha con los parámetros orbitales optimizados

Posteriormente se añadió la velocidad en el instante inicial como parámetro de entrada, obteniendo así un modelo con un MAE de 0.3427 sobre entrenamiento y 0.4188422 sobre test. Ahora el error medio tras la corrección fue de 17869264 metros.

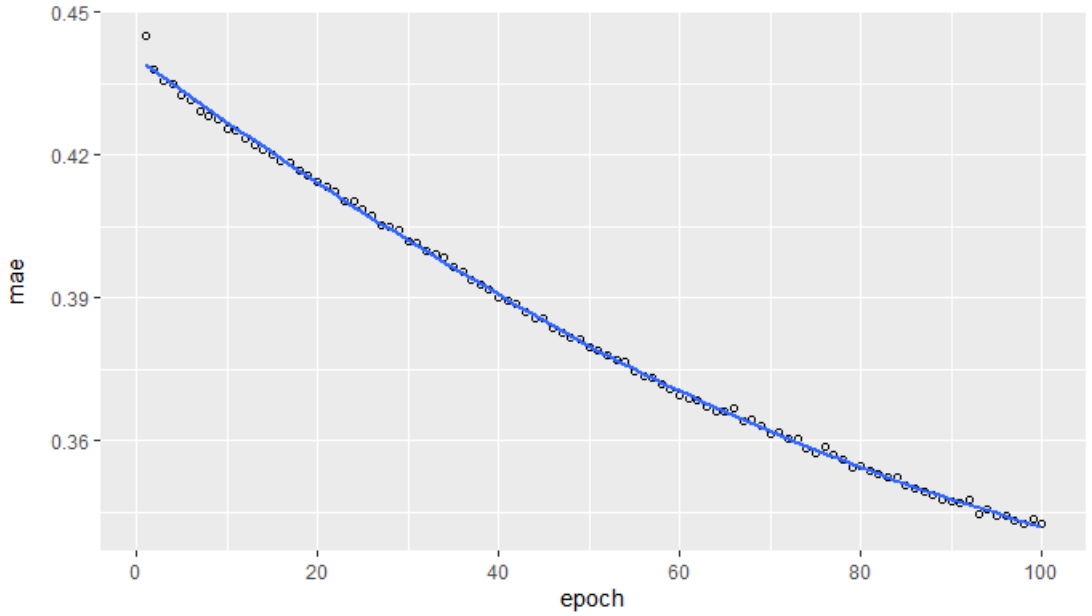

Figura 8.6: Evolución del MAE recibiendo los parámetros base, la velocidad inicial y la posición predicha con los parámetros orbitales optimizados

Modelos que reciben el movimiento predicho

El modelo que recibía los parámetros base más el movimiento predicho con parámetros optimizados obtuvo un MAE de 0.3588 en el conjunto de entrenamiento y de 0.3927993 en el de test. El error medio de la posición con la corrección pasó a ser de 16831337 metros, con lo que mejora.

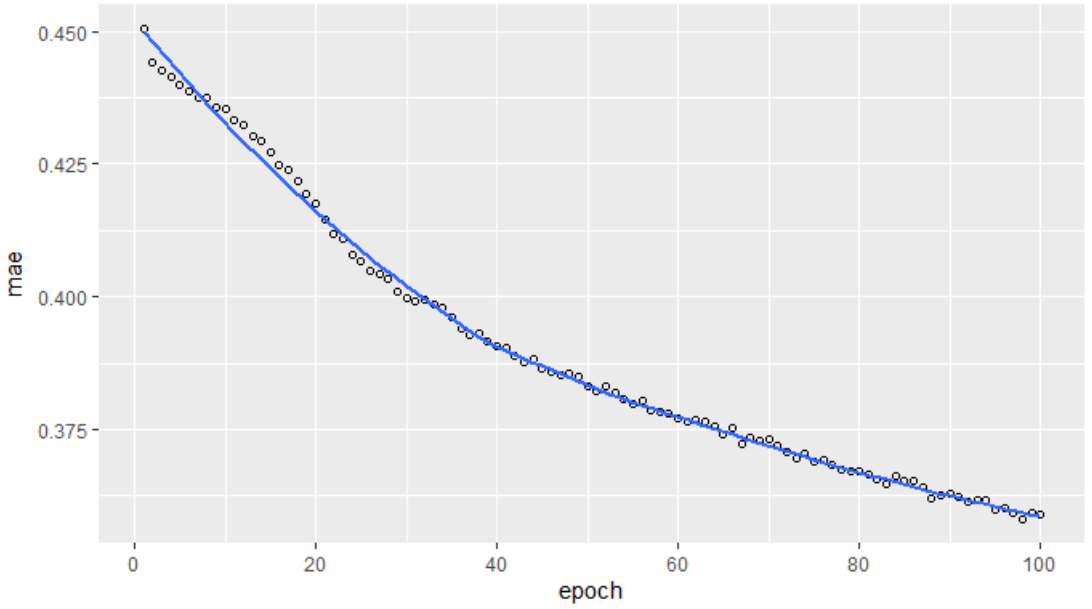

Figura 8.7: Evolución del MAE recibiendo los parámetros base y el movimiento predicho con los parámetros orbitales optimizados

Tras añadir la velocidad inicial, el MAE del entrenamiento fue de 0.3598 y el de test de 0.3921267 . El error medio de la posición con la corrección pasó a ser de 16814065 metros.

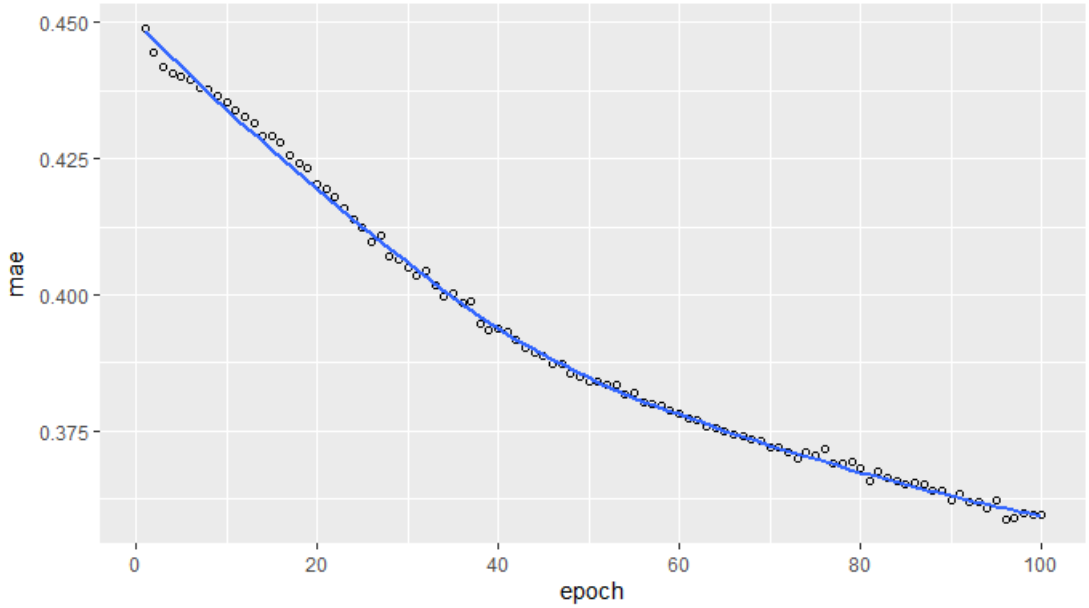

Figura 8.8: Evolución del MAE recibiendo los parámetros base, la velocidad inicial y el movimiento predicho con los parámetros orbitales optimizados

Agrupando estos resultados:

Entradas extra del modelo	**MAE**	**Error medio de la predicción corregida**
Posición predicha con parámetros optim.	0.41	17331023
Posición predicha con parámetros optim. Velocidad inicial	0.42	17869264
Movimiento predicho con parámetros optim.	0.39	16831337
Movimiento predicho con parámetros optim. Velocidad inicial	0.39	16814065

Cuadro 8.3: Comparativa de modelos para predicciones con parámetros optimizados

8.3. Evaluación de los modelos con datos de los astros

A la vista de los resultados anteriores, se decide entrenar modelos que reciben el movimiento predicho junto con datos de los astros.

8.3.1. Parámetros orbitales sin optimizar

El modelo que recibe el movimiento predicho, las posiciones de todos los astros, la libración lunar y el movimiento predicho sin optimizar parámetros obtuvo un MAE en el entrenamiento de 9.3852 y de 12.10751 en el test. El error medio tras la corrección fue de 540895.5 metros.

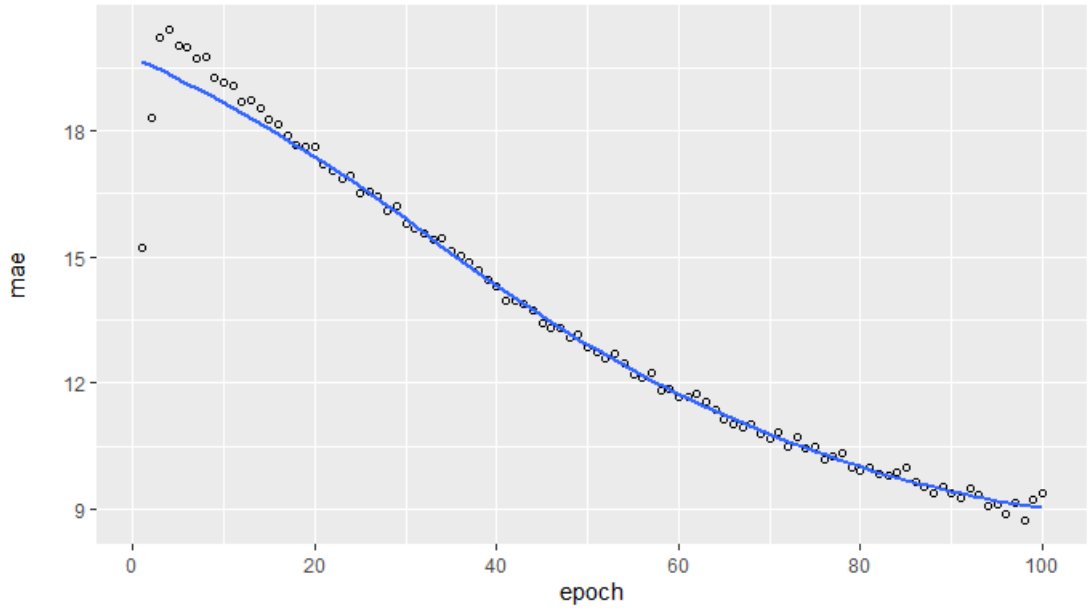

Figura 8.9: Evolución del MAE recibiendo los parámetros base, los datos de todos los astros del Sistema Solar y el movimiento predicho con los parámetros orbitales sin optimizar

Al añadir la velocidad inicial a las entradas del anterior modelo, el nuevo modelo entrenado tuvo un MAE de 9.5772 sobre le conjunto de entrenamiento y 11.55312 sobre test. El error medio tras la corrección fue de 519358.5 metros.

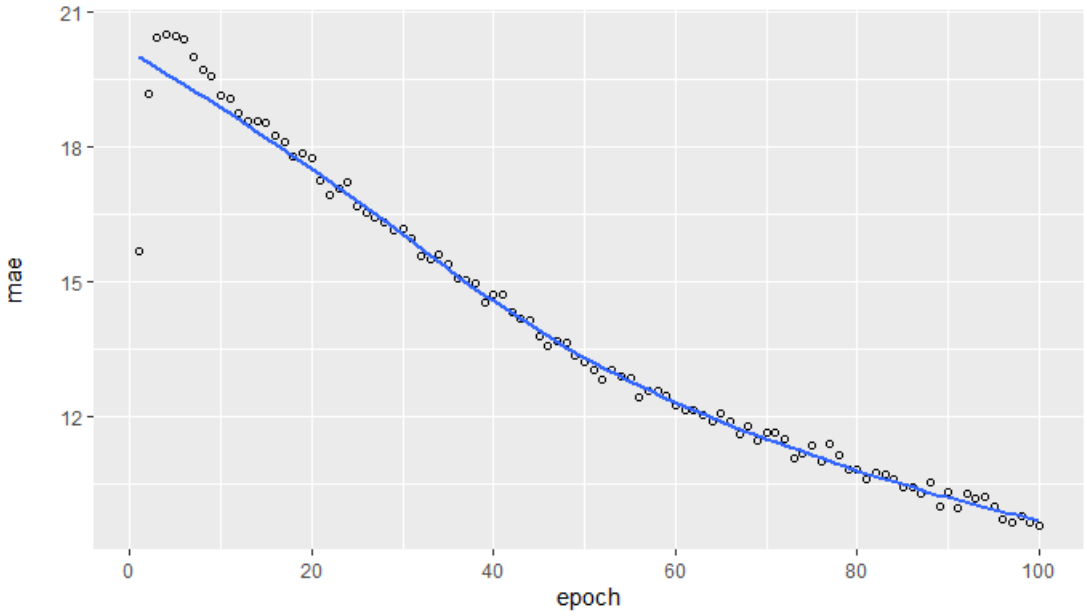

Figura 8.10: Evolución del MAE recibiendo los parámetros base, la velocidad inicial, los datos de todos los astros del Sistema Solar y el movimiento predicho con los parámetros orbitales sin optimizar

El modelo que recibe el movimiento predicho, la velocidad inicial, las posiciones del Sol, Venus, la Luna, Marte, Júpiter y el movimiento predicho sin optimizar obtuvo un MAE de 7.8431 sobre el conjunto de entrenamiento y de 9.702709 sobre el de test. El error medio de la predicción tras aplicar la corrección fue de 433590.8 metros.

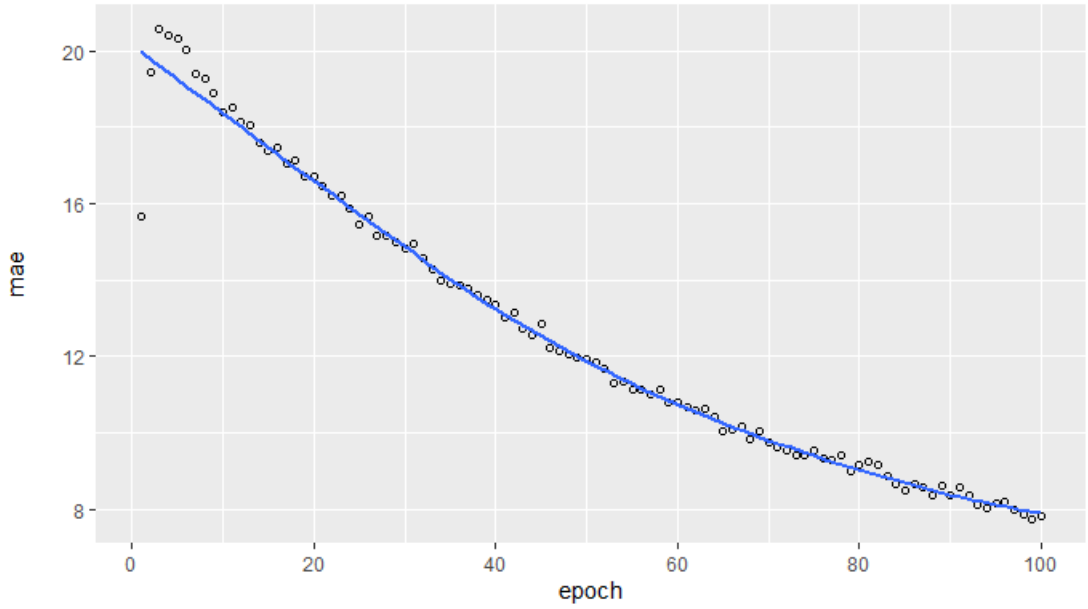

Figura 8.11: Evolución del MAE recibiendo los parámetros base, la velocidad inicial, las posiciones del Sol, Venus, la Luna, Marte, Júpiter y el movimiento predicho con los parámetros orbitales sin optimizar

Agrupando estos resultados:

Entradas extra del modelo	MAE	Error medio de la predicción corregida
Movimiento predicho sin optimizar parámetros Posiciones de todos los astros del Sistema Solar Libración lunar	12.11	540895.5
Movimiento predicho sin optimizar parámetros Velocidad inicial Posiciones de todos los astros del Sistema Solar Libración lunar	11.55	519358.5
Movimiento predicho sin optimizar parámetros Velocidad inicial Posiciones del Sol, la Luna y Júpiter	9.7	433590.8

Cuadro 8.4: Comparativa de modelos para predicciones con parámetros sin optimizar y datos de los astros

8.3.2. Parámetros orbitales optimizados

El modelo que recibe el movimiento predicho, las posiciones de todos los astros, la libración lunar y el movimiento predicho optimizado obtuvo un MAE de 0.3509 en entrenamiento y de 0.4538257 en test. El error medio de la predicción tras la corrección fue de 19314206 metros.

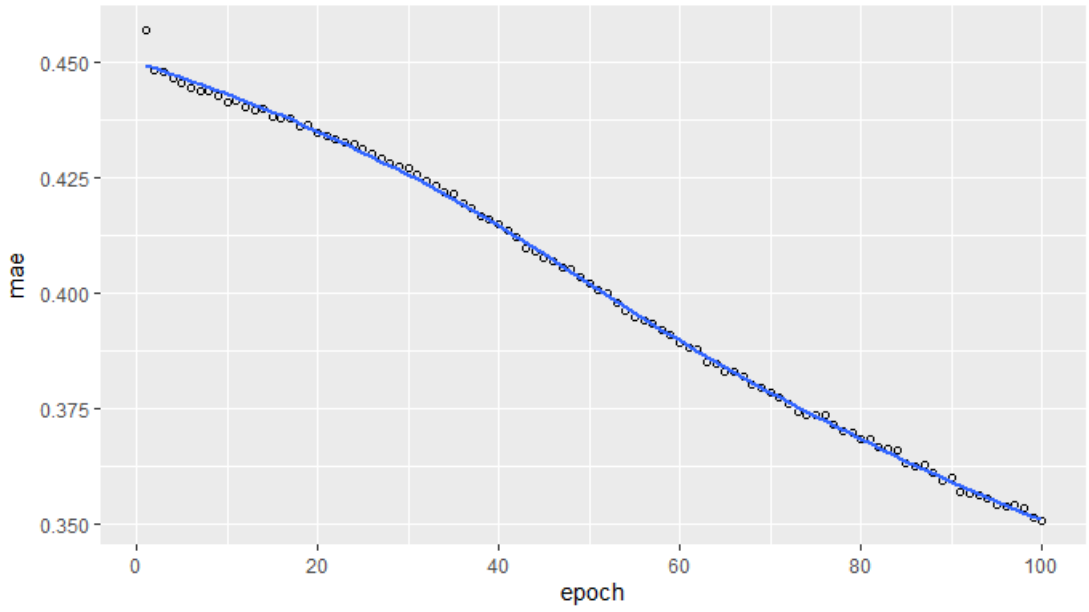

Figura 8.12: Evolución del MAE recibiendo los parámetros base, los datos de todos los astros del Sistema Solar y el movimiento predicho con los parámetros orbitales optimizados

El modelo que recibe el movimiento predicho, la velocidad inicial, las posiciones

de todos los astros, la libración lunar y el movimiento predicho optimizado obtuvo un MAE de 0.3532 en entrenamiento y de 0.4563037 en test. El error medio de la predicción tras la corrección fue de 19404345 metros.

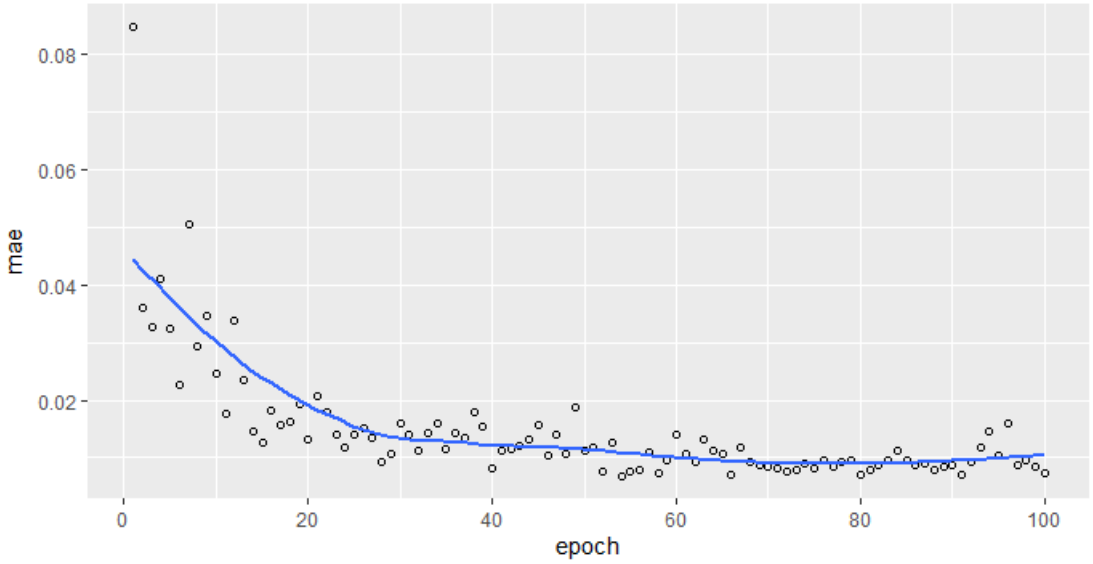

Figura 8.13: Evolución del MAE recibiendo los parámetros base, la velocidad inicial los datos de todos los astros del Sistema Solar y el movimiento predicho con los parámetros orbitales optimizados

El modelo que recibe el movimiento predicho, la velocidad inicial, las posiciones del Sol, Venus, la Luna, Marte, Júpiter y el movimiento predicho con los parámetros optimizados obtuvo un MAE en entrenamiento de 0.3395 y en test de 0.4589353. El error medio tras la predicción fue de 19520238 metros.

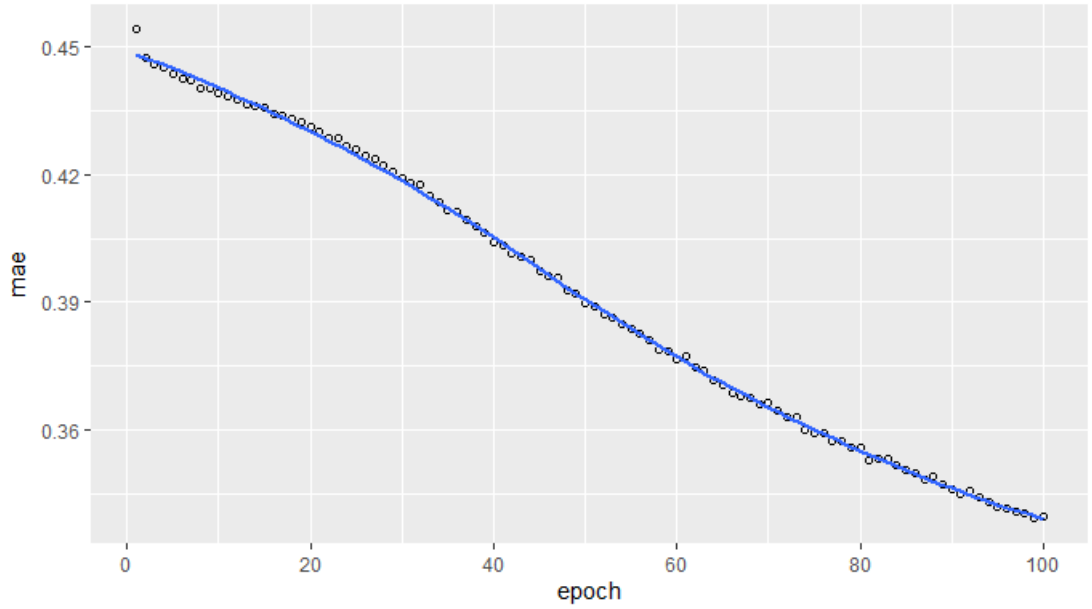

Figura 8.14: Evolución del MAE recibiendo los parámetros base, la velocidad inicial, las posiciones del Sol, Venus, la Luna, Marte, Júpiter y el movimiento predicho con los parámetros orbitales optimizados

Agrupando estos resultados:

Entradas extra del modelo	**MAE**	**Error medio de la predicción corregida**
Movimiento predicho con parámetros optim. Posiciones de todos los astros del Sistema Solar Libración lunar	0.45	19314206
Movimiento predicho con parámetros optim. Velocidad inicial Posiciones de todos los astros del Sistema Solar Libración lunar	0.46	19404345
Movimiento predicho con parámetros optim. Velocidad inicial Posiciones del Sol, la Luna y Júpiter	0.46	19520238

Cuadro 8.5: Comparativa de modelos para predicciones con parámetros optimizados y datos de los astros

8.4. Conclusiones

Tras entrenar diversos modelos y analizar el error metido cometido en la predicción tras aplicar la corrección de los mismos concluimos que es viable el uso de redes neuronales para mejorar la predicción del modelo SGDP4.

Extradas extra del modelo	**Parámetros optimizados**	**MAE**	**Error medio de la predicción corregida**	**ΔError(%)**
Posición predicha	No	14.4	647040.5	11.82
Posición predicha Velocidad inicial	No	12.98	583126.3	0.78
Movimiento predicho	No	12.29	559453	-3.31
Movimiento predicho Velocidad inicial	No	13.93	631188.6	9.09
Posición predicha	Sí	0.41	17331023	-30.74
Posición predicha Velocidad inicial	Sí	0.42	17869264	-28.59
Movimiento predicho	Sí	0.39	16831337	-32.73
Movimiento predicho Velocidad inicial	Sí	0.39	16814065	-32.8
Movimiento predicho Posiciones de los astros Libración lunar	No	12.11	540895.5	-6.52
Movimiento predicho Velocidad inicial Posiciones de los astros Libración lunar	No	11.55	519358.5	-10.24
Movimiento predicho Velocidad inicial Posición del Sol Posición de la Luna Posición de Júpiter	No	9.7	433590.8	-25.06
Movimiento predicho Posiciones de los astros Libración lunar	Sí	0.45	19314206	-22.81
Movimiento predicho Velocidad inicial Posiciones de los astros Libración lunar	Sí	0.46	19404345	-22.45
Movimiento predicho Velocidad inicial Posición del Sol Posición de la Luna Posición de Júpiter	Sí	0.46	19520238	-21.99

Cuadro 8.6: Comparativa de todos los modelos entrenados

Destacar el mejor modelo de todos los de las predicciones sin optimizar, que recibe los parámetros base, la velocidad inicial, las posiciones del Sol, la Luna y Júpiter y el movimiento predicho y que reduce el error medio en un 25 %.

En el caso de las predicciones con los parámetros optimizados, todos los modelos han mejorado la predicción, reduciendo considerablemente el error respecto a las predicciones de SGDP4, sin embargo, como estas predicciones eran peores que las otras, la red no ha podido corregir suficientemente el error como para aproximarse a la posición real.

9. Conclusiones

Este Trabajo Fin de Grado se ha centrado en torno a la propagación de trayectorias de satélites y la dificultad existente para obtener predicciones precisas, especialmente a medio y largo plazo, con un coste computacional admisible.

En este sentido se han desarrollado una serie de algoritmos que permiten optimizar los parámetros orbitales utilizados por los propagadores SGP4 y SDP4, de modo que sus predicciones se ajusten mejor a la órbita en el entorno del punto inicial. Dichos algoritmos fueron también empleados en la elaboración de un conjunto de datos para el satélite Kosmos 2514, que reúne posiciones, velocidades, tanto del satélite como de los astros del Sistema Solar, y predicciones sobre la posición y velocidad del mismo con el propagador SGDP4, tanto con los parámetros optimizados como sin optimizar. Cabe mencionar que dichos datos se han unificado para estar en el mismo marco de referencia.

Una vez creado el conjunto de datos se utilizó para entrenar redes neuronales que predijeran el error cometido con el propagador y así poder corregirlo. Los resultados obtenidos fueron muy prometedores, consiguiendo un modelo que reducía el error medio en un 25 %.

La idea inicial era entrenar también modelos de redes neuronales recurrentes, que pudieran detectar patrones en los vectores de estado orbitales; sin embargo, debido a la complejidad de la materia, surgieron muchos problemas, como el de la optimización de parámetros orbitales que ocupó más tiempo del previsto. A pesar de todo, ha servido para conocer un poco cómo es el mundo de la investigación, en algunos momentos muy frustrante pero, una vez que se logran los objetivos, muy satisfactorio.

Se proponen diversas ideas como trabajo futuro:

1. Entrenar de redes neuronales recurrentes que puedan detectar patrones periódicos en los vectores de estado orbital.
2. Integrar con algún software de visualización avanzado.
3. Recopilar más datos de fenómenos astronómicos que puedan afectar al movimiento de los satélites como, por ejemplo, las tormentas solares.
4. Estudiar en profundidad la optimización de los parámetros orbitales para intentar que no ocurra la explosión del error de la predicción en tiempos lejanos.

A título personal, tengo que destacar mi satisfacción por haber logrado la elaboración del conjunto de datos, con todas las herramientas matemáticas y técnicas que han sido necesarias, y por aportar una solución útil que permita realizar buenas predicciones de trayectorias en un tiempo reducido, aunando todas las competencias adquiridas durante estos cinco años de Ingeniería Informática y Matemáticas.

10. Bibliografía

[1] Roger R. Bate, Donald D. Mueller, and Jerry E. White. *Fundamentals of Astrodynamics*. Dover Publications, 1 edition, 1971.

[2] Celestrak. Datos actualizados de norad en formato tle. URL https://celestrak.org/NORAD/elements/.

[3] Instituto de Estadística y Cartografía. Repositorio de ficheros rinex versión 2 de la junta de andalucía. URL https://www.juntadeandalucia.es/institutodeestadisticaycartografia/rinex/.

[4] Richard L. Burdenan Douglas J. Faires and Annette M. Burden. *Análisis Numérico*. CENGAGE Learning, 10 edition, 2016.

[5] Felix R. Hoods and Ronald L. Roehrich. *SPACETRACK REPORT NO. 3. Models for Propagation of NORAD Element Sets*. U.S. Department of Defensea, 1988.

[6] Richard S. Hujsak. *A Restricted Four Body Solution for Resonating Satellites Without Drag*. Aerospace Defense Command United Space Air Force, 1979.

[7] Zac Yung-Chun Liu, Scott Tarlow, Mohammad Akbar, Quentin Donnellan, and Darek Senkow. Improved orbital propagator integrated with sgp4 and machine learning. In *35th Annual Small Satellite Conference*, 2021. URL https://digitalcommons.usu.edu/cgi/viewcontent.cgi?article=5067&context=smallsat.

[8] John H. Mathews and Kurtis Fink. *Métodos Numéricos con MATLAB*. Prentice Hall, 3 edition, 2000.

[9] Oliver Montenbruck and Eberhard Gill. *Satellite Orbits: Models, Methods and Applications*. Springer, 1 edition, 2000.

[10] Alaa Osama, Mourad Raafat, Ashraf Darwish, Sara Abdelghafar, and Aboul Hassanien. Satellite orbit prediction based on recurrent neural network using two line elements. pages 298–302, 03 2022. doi: 10.1109/ICCI54321.2022.9756063.

[11] H. Peng and X. Bai. Limits of Machine Learning Approach on Improving Orbit Prediction Accuracy using Support Vector Machine. In S. Ryan, editor, *Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*, page 15, January 2017.

[12] Hao Peng and Xiaoli Bai. Artificial neural network–based machine learning approach to improve orbit prediction accuracy. *Journal of Spacecraft and Rockets*, 55:1–13, 06 2018. doi: 10.2514/1.A34171.

[13] Haoli Ren, Xiaolin Chen, Bei Guan, Yongji Wang, Tiantian Liu, and Kongyang Peng. Research on satellite orbit prediction based on neural network algorithm. pages 267–273, 06 2019. ISBN 978-1-4503-7185-8. doi: 10.1145/3341069.3342995.

[14] Naciones Unidas. Índice en línea de objetos lanzados al espacio exterior. URL https://www.unoosa.org/oosa/osoindex/search-ng.jspx.

[15] David A. Vallado. *Fundamentals of Astrodynamics and Applications*. Springer, 4 edition, 2013.