

# Trabajo Fin de Máster

## Máster en Ingeniería Aeronáutica

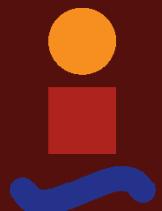
## Predicción de órbitas mediante técnicas de Machine Learning

Autor: César Hernando Tamayo

Tutor: Eduardo Fernández Camacho

Dpto. Ingeniería de Sistemas y Automática  
Escuela Técnica Superior de Ingeniería  
Universidad de Sevilla

Sevilla, 2021





Trabajo Fin de Máster  
Máster en Ingeniería Aeronáutica

# **Predicción de órbitas mediante técnicas de Machine Learning**

Autor:  
César Hernando Tamayo

Tutor:  
Eduardo Fernández Camacho  
Profesor Emérito

Dpto. Ingeniería de Sistemas y Automática  
Escuela Técnica Superior de Ingeniería  
Universidad de Sevilla

Sevilla, 2021



Trabajo Fin de Máster: Predicción de órbitas mediante técnicas de Machine Learning

Autor: César Hernando Tamayo  
Tutor: Eduardo Fernández Camacho

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:



# Agradecimientos

---

**E**n primer lugar me gustaría agradecer el apoyo recibido a todos los familiares y amigos que me han acompañado durante estos últimos años en todas las ciudades por las que he ido pasando: Burgos, León, Perth, Sevilla, Vitoria y Granada. Todos ellos tienen parte del mérito de este trabajo.

Como menciones especiales, debo dar las gracias por la oportunidad que me han brindado mis padres para formarme, y por su incansable apoyo en los días buenos y en los no tan buenos. A mi madre, que tanto me ha insistido en acabar el TFM lo antes posible, y a mi padre y mi hermana que han sabido mediar en el asunto.

También me gustaría agradecer el trabajo a mis compañeros de piso y de equipo, que han logrado que estos meses de arduo trabajo vayan a ser recordados como una etapa feliz de mi vida.

Me gustaría también agradecer a Diego, que tanto me ha ayudado cuando más lo necesitaba.

Por último, me gustaría agradecer este trabajo a Elena, que tanto me ha acompañado y apoyado a lo largo de la realización del trabajo. A ella quiero desearle muchísimo ánimo para el último tramo de estudio para el MIR y muchísima suerte cuando haga este examen.



# Resumen

---

**E**n este trabajo se realiza el estudio de predicción de órbita de un CubeSat mediante el uso de técnicas de Machine Learning, utilizando dos aproximaciones y varios modelos, desde lo más sencillo a más complejos y robustos. Los datos a introducir en estos algoritmos se obtienen a partir de dos modelos: un modelo simplificado de perturbaciones (SGP8), y un modelo dinámico diseñado en el software Simulink. Una vez en disposición de estos datos, la primera aproximación consistirá en utilizar los datos pasados tanto del modelo dinámico como del simplificado para corregir los errores del modelo simplificado en el futuro consecuente. Para esta aproximación se utilizarán los métodos de regresión de Ridge y Lasso.

La segunda aproximación consistirá en propagar los datos del modelo dinámico, considerado el real, para, una vez entrenado el modelo, con la información de una sola revolución, estimar de forma indefinida la posición del satélite en el futuro.

El objetivo del estudio radica en la comparación de las aproximaciones y las diferentes técnicas de aprendizaje automático; y la utilidad que pueden tener estas para la mejora de la estimación de órbitas con respecto a las técnicas actualmente empleadas. En esta aproximación se utilizarán los métodos SVR (Support Vector Regression), GBRT(Gradient Boosting Regression Tree) y XGBoost (eXtreme Gradient Boosting).



# **Abstract**

---

In this project, through the use of Machine Learning techniques, it has been carried out the orbit prediction study of a CubeSat. Two approaches and several algorithms for each approach have been used, from the simplest to more complex and robust ones. The data entered in these algorithms have been obtained from two sources: a simplified perturbations model (SGP8), and a dynamic model designed using Simulink. Once these data are available, the first approach will consist of using the past data of both dynamic and simplified models to correct errors of the simplified model in the consequent future. For this approximation, Ridge and Lasso regression methods will be used.

Second approach will consist of propagating the data of the dynamic model, considered as the real position, in order, once the model has been trained, with the information from a single revolution, indefinitely estimate the position of the satellite in the future. The Machine Learning techniques that will be used are: SVR (Support Vector Regression), GBRT(Gradient Boosting Regression Tree) and XGBoost (eXtreme Gradient Boosting).

The aim of the study lies in the comparison of the approaches and Machine Learning techniques; and the utility they have to improve the ability to forecast orbits compared to the current techniques.



# Índice Abreviado

---

<i>Resumen</i>	III
<i>Abstract</i>	V
<i>Índice Abreviado</i>	VII
<b>1 Introducción</b>	<b>1</b>
1.1 Procedimiento	1
1.2 CubeSat Hatchling Veery "Clay"	2
1.3 MATLAB y Simulink	4
1.4 Python	5
1.5 Machine Learning	6
<b>2 Modelos de Perturbaciones Simplificados</b>	<b>11</b>
2.1 Desarrollo de los modelos de perturbaciones	11
2.2 Elementos orbitales	12
2.3 Código TLE	13
2.4 Proceso de determinación de órbita para el modelo SGP8	14
<b>3 Modelo dinámico de un satélite</b>	<b>21</b>
3.1 Análisis del modelo dinámico	21
<b>4 Comparación de modelos</b>	<b>31</b>
<b>5 Técnicas de Machine Learning</b>	<b>35</b>
5.1 Primera aproximación	35
5.2 Segunda aproximación	36
<b>6 Resultados</b>	<b>41</b>
6.1 Presentación de resultados	41
6.2 Comparación de resultados	68
6.3 Análisis de resultados	70
<b>7 Conclusiones</b>	<b>71</b>
<b>Apéndice A Código de MATLAB</b>	<b>73</b>
A.1 SGP4	73
A.2 SGP8	78

A.3	Test SGP8	83
<b>Apéndice B</b>	<b>Código de Python</b>	<b>87</b>
B.1	Primera aproximación	87
B.2	Segunda aproximación	89
<i>Índice de Figuras</i>		97
<i>Índice de Tablas</i>		99
<i>Índice de Códigos</i>		101
<i>Bibliografía</i>		103
<i>Glosario</i>		105

# Índice

---

<i>Resumen</i>	III
<i>Abstract</i>	V
<i>Índice Abreviado</i>	VII
<b>1 Introducción</b>	<b>1</b>
1.1 Procedimiento	1
1.2 CubeSat Hatchling Veery "Clay"	2
1.2.1 Cubesat	2
1.2.2 Hatchling Veery "Clay"	2
1.3 MATLAB y Simulink	4
1.3.1 MATLAB	4
1.3.2 Simulink	4
1.4 Python	5
1.5 Machine Learning	6
<b>2 Modelos de Perturbaciones Simplificados</b>	<b>11</b>
2.1 Desarrollo de los modelos de perturbaciones	11
2.2 Elementos orbitales	12
2.3 Código TLE	13
2.4 Proceso de determinación de órbita para el modelo SGP8	14
<b>3 Modelo dinámico de un satélite</b>	<b>21</b>
3.1 Análisis del modelo dinámico	21
3.1.1 Entorno	22
3.1.2 Configuración de la misión	24
3.1.3 Dinámica del satélite	24
3.1.4 Módulo de visualización	27
<b>4 Comparación de modelos</b>	<b>31</b>
<b>5 Técnicas de Machine Learning</b>	<b>35</b>
5.1 Primera aproximación	35
5.1.1 Regresión Ridge	35
5.1.2 Regresión Lasso	36
5.2 Segunda aproximación	36
5.2.1 Gradient Boosting Regression Tree (GBRT)	37

5.2.2	XGBoost	39
5.2.3	Support Vector Regression (SVR)	39
<b>6</b>	<b>Resultados</b>	<b>41</b>
6.1	Presentación de resultados	41
6.1.1	Primera aproximación	41
Modelo simplificado de perturbaciones		41
Regresión Ridge		43
Regresión Lasso		45
6.1.2	Segunda aproximación	47
SVR		47
GBRT		51
XGBoost		59
6.2	Comparación de resultados	68
6.2.1	Primera aproximación	68
6.2.2	Segunda aproximación	68
6.2.3	Comparación global	69
6.3	Análisis de resultados	70
<b>7</b>	<b>Conclusiones</b>	<b>71</b>
<b>Apéndice A</b>	<b>Código de MATLAB</b>	<b>73</b>
A.1	SGP4	73
A.2	SGP8	78
A.3	Test SGP8	83
<b>Apéndice B</b>	<b>Código de Python</b>	<b>87</b>
B.1	Primera aproximación	87
B.2	Segunda aproximación	89
B.2.1	Modelo de una coordenada	89
B.2.2	Modelo completo	91
B.2.3	Bloques de técnicas de Machine Learning	94
<i>Índice de Figuras</i>		97
<i>Índice de Tablas</i>		99
<i>Índice de Códigos</i>		101
<i>Bibliografía</i>		103
<i>Glosario</i>		105

# 1 Introducción

---

Tras más de medio siglo desde el lanzamiento del Sputnik, la cantidad de Objetos residentes en el Espacio, RSO a partir de ahora, por sus siglas en inglés (Resident Space Object) ha ido aumentando cada vez a mayor ritmo.

La definición de RSO es un cuerpo natural o artificial que orbita alrededor de un cuerpo; que en este caso será la Tierra. Según la altitud de su órbita se clasifican en LEO (Low Earth Orbit, 160-2.000 km), MEO (Medium Earth Orbit, 2.000- 36.000 km), HEO (High Earth Orbit, más de 36.000 km) o GEO (Geosynchronous Earth Orbit, 35.786 km).

A fecha de 2019 existían más de 21.000 RSO mayores de 10 cm, y alrededor de 500 000 de entre 1 y 10 cm. Esto se traduce en 12,5 colisiones no deliberadas entre objetos al año ([21]). Debido a estos factores, se necesita cada vez más poseer modelos en los que los cuerpos residentes en el espacio estén mejor acotados y se tengan predicciones más precisas de sus localizaciones en espacios temporales más amplios.

Hasta la actualidad, la mayor parte de las predicciones de posiciones se calculan a partir de modelos físicos, pero cada vez se tiende más a utilizar técnicas de Machine Learning para este propósito. En algunos estudios se utiliza el Support Vector Machine (SVM) como la técnica de Machine Learning empleada para este propósito ([22], [23]); en otros se utilizan redes neuronales ([25]). En este estudio se va a realizar un análisis comparativo del uso de distintas técnicas con la intención de concluir con las más adecuadas.

## 1.1 Procedimiento

Se tomará un satélite de referencia, y se usarán diferentes modos de registro de posición y velocidad a lo largo de un tiempo determinado, de los que se obtendrán los datos necesarios para usar las técnicas de Machine Learning.

Las fuentes de datos serán dos: un modelo dinámico de un satélite implementado en Simulink, una herramienta de MATLAB; y un modelo simplificado de perturbaciones (SGP8), una herramienta utilizada para propagar órbitas de satélites a lo largo del tiempo. Se tomará como posición y velocidad real las obtenidas por el modelo dinámico, mientras que se observará como se va alejando de la realidad las medidas del modelo simplificado con el tiempo. A partir de la obtención de estos datos durante un tiempo específico, se realizarán dos aproximaciones, una usando solamente la información del modelo dinámico, y otra combinando ambas.

En la primera aproximación, se utilizarán tanto los datos reales del modelo dinámico como los del modelo simplificado de perturbaciones, propagados como entradas del modelo de Machine Learning. A continuación, se introducirán los datos calculados por el modelo simplificado en un futuro consecuente al introducido en el modelo de Machine Learning, y éste corregirá los errores del modelo simplificado, de manera que se acerque más a la posición real del satélite.

En la segunda aproximación, se recopilarán los datos del modelo dinámico durante una fase llamada de entrenamiento, y estos serán propagados en el futuro, en el que podrán ser comparados con los datos reales del modelo dinámico durante ese marco temporal. En lo único que se podrá emplear el modelo simplificado de perturbaciones será para comparar con los datos del nuevo modelo creado, para comprobar si a través de la técnica de Machine Learning se consigue mejorar la predicción respecto a este.

Finalmente, se modificarán la técnicas de Machine Learning para las dos aproximaciones y se compararán, analizando los resultados de cada una de ellas.

A continuación, se definirán los conceptos, técnicas y programas sobre los que se sostiene este estudio:

## 1.2 CubeSat Hatchling Veery "Clay"

### 1.2.1 Cubesat

El satélite de ejemplo de este proyecto será un CubeSat de 1U llamado Hatchling Veery "Clay".

Para definir lo que es un CubeSat, se debe empezar por la definición de nanosatélite. Un nanosatélite es cualquier satélite con una masa entre 1- 10 kg; aunque existen CubeSats de alrededor de 0.8 kg que también entran en esta categoría. Por debajo de esas masas están los picosatélites, con masas que rondan los 100 g [5].

Un CubeSat es un tipo de nanosatélite definido por la Especificación de Diseño de CubeSat (CubeSat Design Specification, CSD), también llamado de forma no oficial el estándar CubeSat. Este estándar fue creado por la Universidad Politécnica del estado de California, San Luis Obispo y el laboratorio de Desarrollo de Sistemas Espaciales de la Universidad de Stanford en 1999, con el objetivo de facilitar el acceso al espacio a los estudiantes universitarios. Desde entonces, este estándar ha sido adoptado por cientos de organizaciones en todo el mundo. Hoy en día, los desarrolladores de CubeSats no son sólo universidades u otras instituciones educativas, si no también organizaciones gubernamentales y empresas privadas. El estándar CubeSat permite lanzamientos frecuentes y económicos, por lo que está teniendo un gran éxito [24].

Los CubeSats tienen diferentes dimensiones estandarizadas. Algunas de las más frecuentes son:

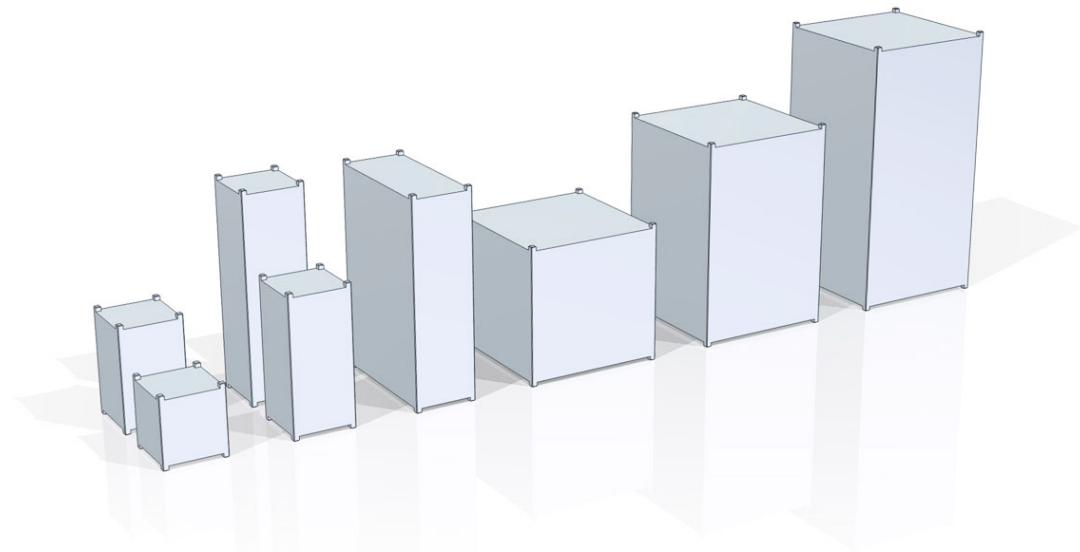
- 1U: 10 cm x 10 cm x 11,35 cm.
- 2U: 10 cm x 10 cm x 22,7 cm.
- 6U: 20 cm x 10 cm x 34,05 cm.
- 12U: 20 cm x 20 cm x 34,05 cm.

A partir de estos ejemplos de dimensiones, se pueden configurar de distintos tamaños, siendo el más pequeño lanzado de 0,25U y próximamente se prevén lanzamientos de 16U y 20U, como los CubeSats de mayor tamaño lanzados.

### 1.2.2 Hatchling Veery "Clay"

El Cubesat Hatchling Veery "Clay" pertenece a la Compañía Care Weather. Esta compañía estadounidense, fundada en 2019, pretende crear una flota de Cubesats de 1U que sean capaces de registrar datos atmosféricos a nivel mundial a tiempo real. Los primeros constarán de un radar para captar la velocidad del viento oceánico cada hora.

Para el satélite de estudio en concreto, se tardaron tan solo 5 meses desde que fue ideado hasta su puesta en órbita. El identificador de este satélite es Veery-RL1, y su fecha de lanzamiento fue el 22 de marzo de 2021.



**Figura 1.1** CubeSats desde 1U hasta 16U.

Este satélite es el primero en utilizar el controlador de vuelo de la compañía. Cuenta con baterías, paneles solares, paneles de drag, giróscopos en 3 ejes y estructura propia. Cuenta con el módulo de propulsión integrado PPT3-1C. El elemento restante, propio de misiones meteorológicas, que no posee, es el sensor meteorológico, que está en desarrollo, puesto que se está reduciendo en un orden de magnitud de 1000 con respecto a uno convencional, y será la pieza revolucionaria.

Algunos de los parámetros más destacados de este satélite son:

- Potencia instantánea: 125 W
- Capacidad de batería: 9 A-h
- Masa: 0,9 kg
- Energía solar: 1 W
- Velocidad de datos: 1 kbps
- Velocidad del procesador: 80 MHz

El encargado del lanzamiento fue Rocket Lab, con un lanzador llamado Electron. Estos lanzadores desechables pueden tener dos o tres etapas, son utilizados para poner en órbita Cubesats y poseen el primer motor alimentado con una bomba eléctrica en cohete.

La misión tiene previsto durar seis meses y la vida orbital del satélite será de tres años y medio [3].

Algunos de los datos identificativos de la órbita de este satélite más relevantes son los siguientes [20]:

- **Apogeo:** 560,7 km.
- **Perigeo:** 545,0 km.
- **Inclinación:** 45°.
- **Período:** 95,6 min.
- **Semieje mayor:** 6923 km.



**Figura 1.2** Imagen del Veery Hatchling "Clay" donde se puede comparar con el tamaño de una mano.

## 1.3 MATLAB y Simulink

### 1.3.1 MATLAB

MATLAB (MATrix LABoratory, Laboratorio de Matrices) es una plataforma de programación y cálculo numérico utilizada ampliamente por matemáticos, científicos e ingenieros, tanto en el contexto educativo como profesional, debido a su gran versatilidad a la hora de analizar datos, desarrollar algoritmos y crear modelos [29].

Algunas de sus características más definitorias son:

- Programación directa de vectores y matrices
- Desarrollo de toolboxes de manera profesional y contrastada
- Desarrollo de aplicaciones interactivas, que permiten ejecutar algoritmos diversos con los datos del usuario.

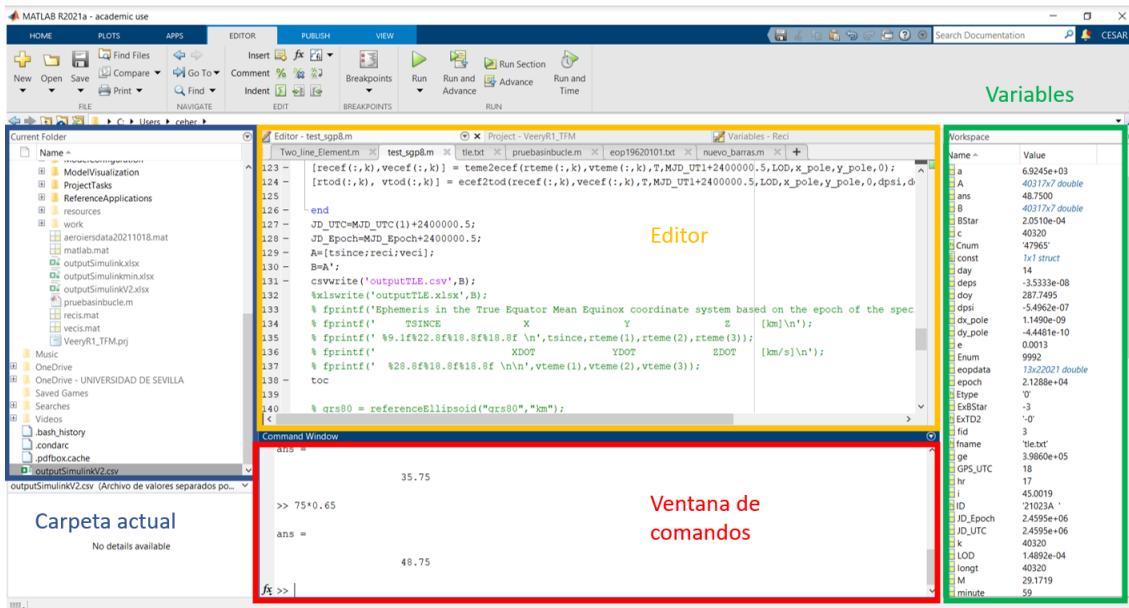
En este proyecto, se va a utilizar este software para la propagación del Modelo Simplificado de Perturbaciones desde el código TLE (Two-Line Element set).

El espacio de trabajo se divide principalmente en:

- Carpeta actual: Localización donde carga y guarda la información.
- Editor: Espacio donde se escribe y visualiza el código a ejecutar.
- Ventana de comandos: Lugar donde se ejecuta el código directamente y donde aparecen los outputs que no están ocultos.
- Variables: Espacio donde aparecen todas las variables cargadas en la memoria.

### 1.3.2 Simulink

Simulink es una herramienta de MATLAB que permite la programación basada en modelos. Es muy útil en ingeniería de sistemas, pues permite diseñar, analizar y optimizar arquitecturas de sistemas ahorrando tiempo y recursos, todo ello antes de aplicarlo al hardware correspondiente.



**Figura 1.3** Entorno de trabajo de MATLAB.

Tiene una biblioteca de bloques predefinidos muy amplia para gran cantidad de sectores. Además, se puede acceder al interior de todos ellos para observar el funcionamiento. También tiene modelos y proyectos predefinidos que se pueden adecuar a las necesidades del usuario [2].

En este proyecto, Simulink se utiliza para el modelo físico que representa el movimiento real del satélite, habiéndose escogido como referencia un proyecto predefinido de CubeSat de 1U, disponible en el Aerospace Blockset.

## 1.4 Python

Python es un lenguaje de programación de alto nivel, es decir, tiene una sintaxis próxima al lenguaje natural. Es uno de los más extendidos en el mundo, debido a su facilidad de aprendizaje, potencia de cálculo, y licencia abierta. Algunos de los campos en los que más se utiliza son: Data analytics, big data, data mining, data science, inteligencia artificial, block chain, machine learning y desarrollo web. La gran mayoría de estos campos implican la gestión de gran cantidad de información para llegar a conclusiones resumidas y concretas [8].

Debido a estas razones, se ha concluido que es el mejor programa para recoger los datos de los modelos de perturbación simplificado y dinámico ampliado; y aplicar las técnicas de Machine Learning. Para ello, las principales librerías utilizadas son:

- Pandas: especializada en la manipulación y visualización de grandes cantidades de datos [18].
- Scikit-learn: herramienta adecuada para el análisis de datos de manera automatizada, como por ejemplo regresiones o clasificaciones. Esta herramienta incluye las técnicas de Machine Learning [17].
- NumPy: es el paquete fundamental para cálculo matricial de Python [1].
- Matplotlib: permite crear gráficas con los resultados obtenidos [7].

## 1.5 Machine Learning

El Machine Learning, también llamado aprendizaje automático, es un método de análisis de datos que automatiza la construcción de modelos analíticos. Es una rama de la inteligencia artificial (AI), que se basa en que los sistemas pueden aprender de los datos, reconocer patrones y tomar decisiones sin apenas intervención humana.

El concepto del aprendizaje automático por parte de los ordenadores no es nuevo, los investigadores en inteligencia artificial llevan iterando algoritmos desde hace muchos años. Sin embargo, con los avances en computación, y la mayor capacidad de almacenamiento de datos (Big Data), esta ciencia ha tomado un gran impulso. La capacidad de aplicar automáticamente cálculos matemáticos complejos al big data de manera rápida y eficaz se usa cada vez más. Algunos ejemplos son: coches autónomos, recomendaciones de plataformas audiovisuales, detección de spam, etc.

Los fundamentos del machine learning consisten en buscar patrones en datos pasados para aplicarlos en situaciones futuras [26].

Para crear un buen sistema de Machine Learning se requiere:

- Recursos de preparación de datos.
- Algoritmos adecuados al problema.
- Automatización y procesos iterativos.
- Escalabilidad.
- Modelado en conjunto.

El proceso lógico a seguir al crear una rutina de Machine Learning es:

1. Importar los datos.
2. Limpiar los datos.
3. Separar los datos entre daos de entrenamiento y de testeo.
4. Crear el modelo.
5. Entrenar el modelo.
6. Hacer predicciones.
7. Evaluar y mejorar.

Este proceso se repite para cualquier técnica de Machine Learning empleada. En el caso de este estudio, los datos a emplear han sido generados por nosotros mismos, y no necesitan ser limpiados, por lo que a la hora de crear el código de cada técnica, después de importar los datos se pasará directamente a separar entre las muestras de entrenamiento y de testeo.

Las técnicas de machine learning que se van a emplear en este proyecto son las siguientes:

- **Regresión Ridge:** Se trata de una estrategia para afrontar un problema de regresión lineal multivariable. Una regresión lineal multivariable no es más que una función que tiene una variable continua y dos o más variables independientes; y esta función se trata de ajustar con una ecuación lineal. El método natural para afrontarlo sería por mínimos cuadrados. Este método no funciona bien cuando las variables están estrechamente correlacionadas, utiliza todos los predictores aunque no sean relevantes y no es capaz de ajustar modelos cuando el número de predictores es superior al de variables.

El método de regresión Ridge fue introducido en [12]. Mientras que el método de mínimos cuadrados simplemente trata de minimizar la suma de los residuos al cuadrado; en el método

Ridge se añade el parámetro  $\alpha$  multiplicado por la pendiente de la función. Por tanto, el método Ridge trata de minimizar:

$$\|Y - X s\|^2 + \alpha \|s\|^2 \quad (1.1)$$

donde el primer término alude a los residuales y el segundo a la penalización.  $s$  representa la pendiente de la función. Mediante el uso de Ridge, se aumenta ligeramente el sesgo pero se reduce en mayor medida la varianza. Así, se consiguen mejores predicciones a largo plazo que con mínimos cuadrados.

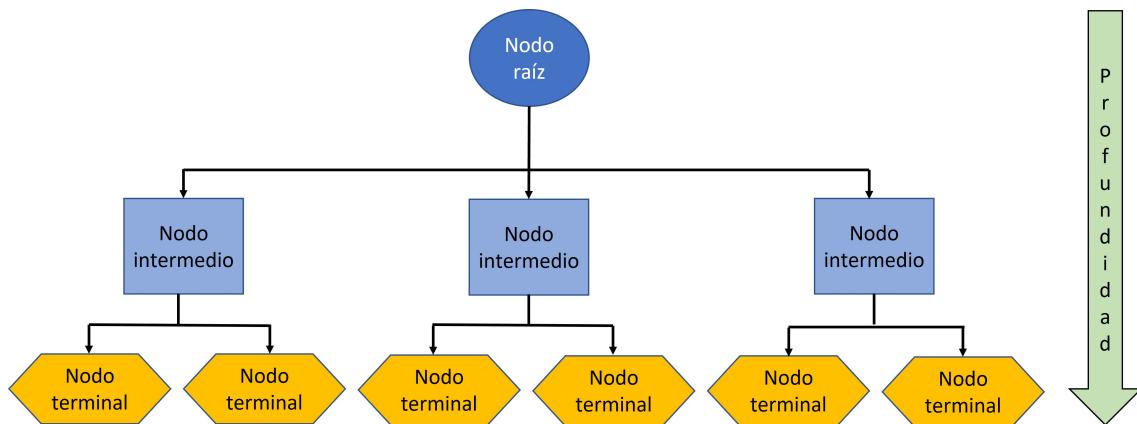
- **Regresión Lasso:** Este método es también una estrategia para afrontar una regresión lineal multivariante. Es un método muy similar a Ridge, sólo cambiando el cuadrado de la pendiente en la penalización por el valor absoluto de ella:

$$\|Y - X s\|^2 + \alpha \|s\| \quad (1.2)$$

De esta manera, también se aumenta ligeramente el sesgo para reducir la varianza en mayor medida. Los resultados de estos dos métodos son muy parecidos. Las grandes diferencias entre ellos es que la pendiente de Ridge solo se puede acercar asintóticamente a 0, mientras que en Lasso puede llegar a 0; y que mientras que Lasso puede dejar coeficientes en 0, Ridge solo puede reducirlos.

- **Árboles de decisión:** Es un método de aprendizaje automático supervisado, que se puede utilizar tanto para clasificación como para regresión. Su propósito es la creación de un modelo capaz de predecir una variable objetivo aprendiendo reglas de decisión simples a partir de características de los datos. Aunque no es el algoritmo más potente, es uno de los más utilizados en el ámbito del Machine Learning.

La estructura de un árbol de decisiones es muy característica. Parte de un nodo raíz, desde el que se divide en nodos intermedios hasta los nodos terminales. Se puede variar el número de divisiones o la profundidad del árbol de decisión.

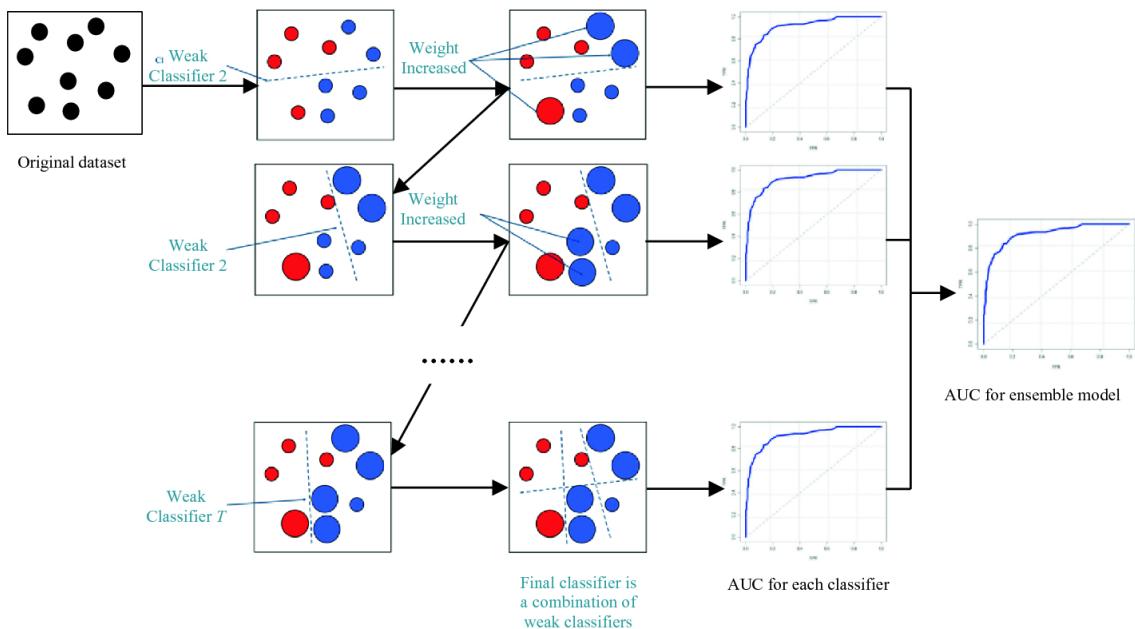


**Figura 1.4** Estructura de un árbol de decisiones.

Este tipo de algoritmo tiene como principales ventajas las siguientes: es muy sencillo, tanto para diseñar, como para interpretar o visualizar; no siempre hace uso de todos los predictores; es capaz de adaptarse a la falta de datos; se puede utilizar tanto para regresiones como para clasificaciones, y responde correctamente a la no linealidad.

También tiene ciertas desventajas, entre las que destaca la tendencia al overfitting, y provoca que en modelos demasiados complejos no se adapte correctamente [9].

- **Gradient Boosting:** Es una técnica de aprendizaje automático que se puede utilizar tanto para clasificación como para regresión. Construye un modelo predictivo a partir de conjuntos débiles, como los árboles de decisiones (técnica que se aplicará en este estudio), para crear un modelo predictivo fuerte. Se basa en la creación de un modelo de manera escalonada, y se encarga de generalizarlos de manera que trata de optimizar una función de pérdida.



**Figura 1.5** Ejemplo gráfico del funcionamiento de una técnica de Gradient Boosting [27].

En la figura 1.5 se puede observar como a través de clasificadores débiles el modelo es capaz de llegar a predicciones fuertes.

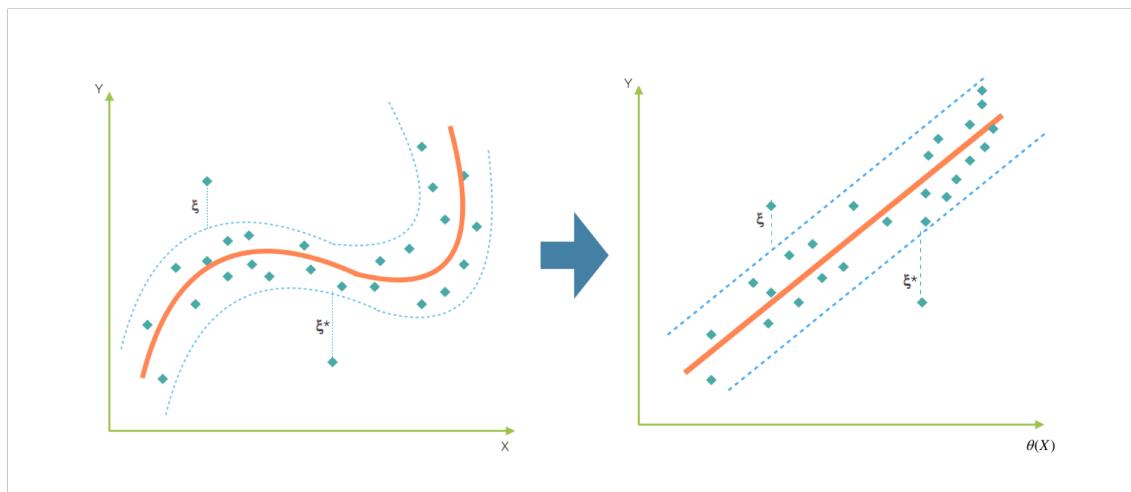
Existe además, un tipo de técnica de Gradient Boosting que será utilizada separadamente. Esta se llama XGBoost (eXtreme Gradient Boosting). Su principal mejora con respecto al Gradient Boosting común es su capacidad para realizar cálculos en paralelo, aprovechando la potencia de computación de los ordenadores actuales. Así, es capaz de obtener resultados de manera muy rápida y precisa, mejorando el rendimiento de la mayoría de algoritmos.

- **Support Vector Regression (SVR):** Se trata de una variante de regresión de Support Vector Machine (SVM). El SVM clasifica muestras separando mediante un plano, denominado hiperplano, una muestra creando dos espacios lo más amplios posibles. Este hiperplano se define añadiendo vectores entre los 2 puntos de las dos clases que se encuentran más cercanos. A estos vectores se le llaman vectores de soporte y dan nombre a la técnica.

Aplicando esta técnica de clasificación a la regresión, se puede intuir que el problema cambia ligeramente, puesto que el resultado pasa a ser un valor real, y la solución tiene infinitas posibilidades. Sabiendo esto, lo que se busca es minimizar el error escogiendo el plano que maximice el margen dado por los vectores de soporte, conociendo el error tolerable. Se puede trabajar con un kernel, que es capaz de convertir datos no lineales en lineales. Los términos que esta técnica trata de minimizar son:

$$\min \frac{1}{2} ||s||^2 + C \sum_{i=1}^N (\xi + \xi^*) \quad (1.3)$$

donde el primer término trata de maximizar el margen y el segundo minimizar el error de entrenamiento. En esta ecuación  $s$  es la magnitud del hiperplano (pendiente del mismo),  $C$  es una constante que determina el equilibrio entre la tolerabilidad de desviaciones del vector de soporte y la regularidad de la función;  $\xi$  y  $\xi^*$  controlan el error cuando se aproxima la función a las bandas.



**Figura 1.6** Representación gráfica del funcionamiento de la técnica SVR [15].

En la figura 1.6 se puede observar como linealiza la función el kernel. También se observa el hiperplano en color naranja y los vectores de soporte en azul con línea discontinua.



## 2 Modelos de Perturbaciones Simplificados

---

Los modelos de perturbaciones simplificados (Simplified General Perturbationss, SGP) son cinco modelos matemáticos (SGP, SGP4, SDP4, SGP8 y SDP8) empleados para calcular vectores de estado orbital de Objetos Residentes en el Espacio (RSOs), basados en los ejes ECI (Earth-Centered Inertial coordinates, coordenadas de inercia centradas en la Tierra). El modelo utilizado en este estudio es el SGP8. A lo largo de este capítulo se mencionarán todos los modelos y se concluirá la razón por la que el SGP8 es el empleado.

Los modelos simplificados de perturbaciones comenzaron a desarrollarse en los años 60, y comenzaron a estar operativos a principios de los 70. La intención era conseguir un modelo simple del comportamiento orbital de un satélite.

En un principio se puede calcular como la suma de los vectores de fuerza que actúan sobre el satélite en cada instante. Esto se calcula mediante integración numérica. Por debajo de los 5000 km y los 225 minutos de período, las fuerzas que intervienen son, además de las gravitatorias:

- **Frenado atmosférico:** rozamiento entre el remanente de atmósfera y el satélite.
- **Presión solar:** presión acumulativa debida a la capacidad reflectiva de la superficie donde impacta la radiación solar.

Este modelo, para ser preciso, necesita valores muy exactos en cada instante de todas las fuerzas, lo que lo vuelve muy complejo y requiere grandes recursos, tanto físicos como computacionales.

Por esta razón, se desarrollaron modelos que tienen en cuenta las perturbaciones existentes en las órbitas. Mediante este sistema, se realizan ciertas simplificaciones, que permiten una mayor agilidad de cálculo en detrimento de la precisión de los resultados de estos. Se consideran perturbaciones a los fenómenos físicos que modifican la posición orbital de un satélite. De esta manera, las dos fuerzas definidas anteriormente, el frenado atmosférico y la presión solar, pasan a ser consideradas como perturbaciones. Este modelo no requiere integración, permite hallar posiciones y velocidades en cualquier marco temporal de forma determinista. Las principales simplificaciones de este modelo son considerar despreciable la masa del satélite con respecto de la Tierra y considerar "baja" la excentricidad de la órbita.

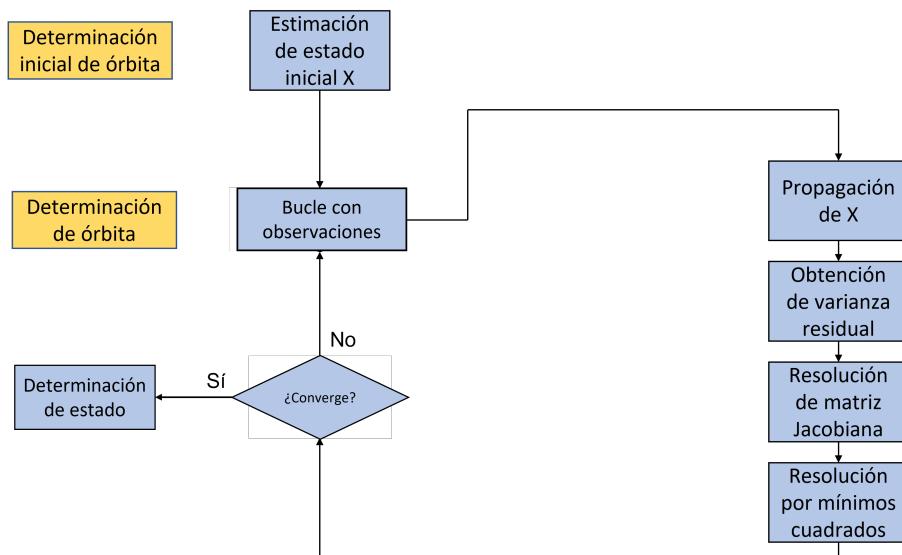
### 2.1 Desarrollo de los modelos de perturbaciones

El modelo SGP fue desarrollado en 1966 por Hilton y Kuhlman [11], basado en investigaciones de Kozai en 1959 [16]. Estaba pensado para satélites con períodos de órbita inferiores a 225 minutos. En este modelo se asume también una baja excentricidad de órbita y que la altitud del perigeo se mantiene constante.

El siguiente modelo, el SGP4 fue desarrollado en 1970. Fue una mejora del anterior modelo, puesto que tenía que cubrir las necesidades del creciente número de satélites orbitando alrededor de la Tierra. Este modelo también está diseñado para satélites con órbitas de período inferior a 225 minutos.

El SDP4, fue desarrollado en 1979 por Hujasak [14]. Se trataba de un modelo a partir del SGP4 adaptado para satélites con período orbital superior a 225 minutos. En este modelo se añaden las perturbaciones gravitatorias inducidas por los sistemas Tierra-Sol y Tierra-Luna. Además también se tienen en cuenta efectos de resonancia de períodos de 12 y 24 horas.

Estos modelos aun siendo más simples que los de fuerzas, tienen gran complejidad por los cálculos que implican. A través de este diagrama de flujo se puede intuir la complejidad de los modelos:



**Figura 2.1** Diagrama de flujo del proceso de propagación de órbita para los modelos SGP4 y SDP4 [30].

Se puede observar el código de MATLAB del modelo SGP4 en el apéndice A.1.

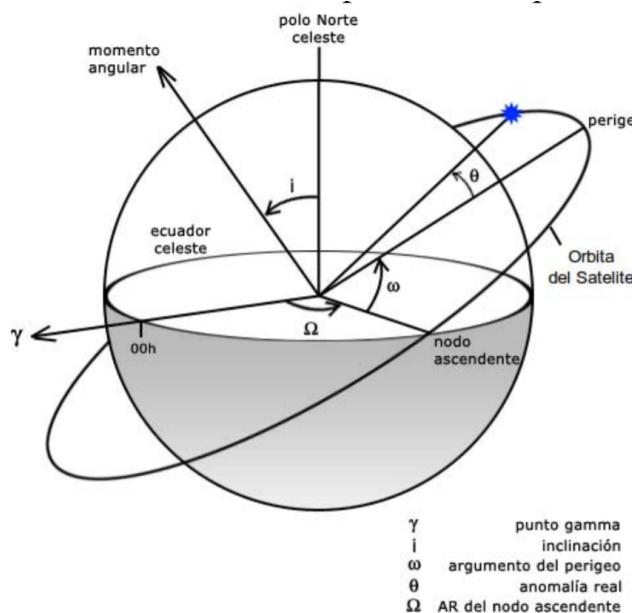
El siguiente modelo es el SGP8, cuyos resultados son ligeramente mejores que los del SGP4 pero hallados por otro método más simple. Este método se utiliza para órbitas bajas de menos de 225 minutos de período. Este modelo es explicado completamente en el apartado A.2.

El último de los 5 modelos se llama SDP8 y es la versión del SGP8 ampliada para satélites con períodos orbitales mayores de 225 minutos.

## 2.2 Elementos orbitales

Para describir una órbita, se utilizan seis elementos, de entre los llamados "Elementos Keplerianos". A continuación, se muestra una representación de los elementos keplerianos para un satélite en órbita (fig. 2.2):

- **Excentricidad (e):** determina la forma de la elipse. Su valor oscila entre 0 y 1, siendo 1 una circunferencia y 0 un segmento.
- **Movimiento medio:** equivalente a velocidad media, medido en revoluciones por día.
- **Inclinación (i):** ángulo entre el plano de órbita y el plano de referencia, medido en el nodo ascendente.



**Figura 2.2** Elementos keplerianos para un satélite en órbita.

- **Ascensión Recta (AR) de nodo ascendente  $\Omega$ :** orientación horizontal del nodo ascendente.
- **Argumento del perigeo ( $\omega$ ):** orientación de la elipse en el plano orbital, desde el nodo ascendente hacia el perigeo (punto más cercano del satélite con respecto a la Tierra):
- **Anomalía real ( $\Theta$ ):** Ángulo en el plano de la elipse entre el perigeo y la posición del satélite en un instante. A ese instante se le denomina "época".
- **Anomalía media:** ángulo que varía linealmente con el tiempo. Es más usado que la anomalía real. Se puede transformar en anomalía real aunque no se corresponde con ella.

Este modelo acumula error con el tiempo. Para controlar este error, se necesitan puntos de referencia del objeto residente en el espacio. Estos puntos de referencia se pueden obtener por observación directa o por radar, y se computan en un formato estándar denominado TLE (Two-Line Element, Elemento de Dos Líneas). Es por esto que el error va creciendo entre observación y observación, y las predicciones dejan de tener cualquier atisbo de rigor para períodos de alrededor de una semana.

## 2.3 Código TLE

El código TLE se genera tras una observación de un cuerpo al que se quiere predecir su trayectoria. En estas dos líneas se contiene toda la información necesaria para el modelo simplificado de perturbaciones. A continuación, se presenta el código TLE utilizado en este estudio y los datos que se pueden extraer de él:

A partir de este código TLE, se ha propagado durante cuatro semanas, minuto a minuto la posición y velocidad del satélite. Con este código se puede visualizar en varios sistemas de coordenadas, pero el que se ha fijado para ser utilizado más tarde es el ECI. De esta manera, se obtendrán tres coordenadas de posición y tres de velocidad, completando los 6 grados de libertad de la misma manera que con los parámetros keplerianos [6].

El código utilizado se encuentra en el apéndice A.2.

Nombre del satélite (11 caracteres)	Designador Internacional	Año de la Época / Fracción del día juliano	1 <sup>a</sup> derivada del movimiento medio (coef. balístico)	2 <sup>a</sup> derivada del movimiento medio	Coeficiente de frenado o Coeficiente de presión solar	Tipo de efemérides	Número de elemento y Checksum
Veerry-RL1							
1 47965U 21023A 21287.74951745 .000002855 00000-0 20510-3 0 9992	47965	21023A	21287.74951745	.000002855	00000-0	20510-3 0	9992
2 47965 45.0019 95.4755 0012600 330.8452 29.1719 15.06679096 31036			95.4755	0012600	330.8452	29.1719	15.06679096 31036
	Número de satélite	Inclinación	AR del nodo Ascendente	Excentricidad	Argumento del perigeo	Anomalía media	Movimiento medio
							Número de revolución para época dada y Checksum

Figura 2.3 Código TLE de estudio con la explicación de sus cifras.

## 2.4 Proceso de determinación de órbita para el modelo SGP8

A continuación, se describe el desarrollo matemático que permite la propagación de las órbitas con el tiempo [13].

Primeramente, se pueden sacar las siguientes variables utilizando: los datos del código TLE de la excentricidad original  $e_0$ , movimiento medio  $n_0$  e inclinación  $i_0$ ; y las constantes  $k_e = \sqrt{GM}$ , donde  $G$  es la constante gravitatoria universal y  $M$  la masa de la Tierra; y  $k_2 = 1/2J_2a_E^2$ , donde  $J_2$  es el segundo armónico gravitacional de la Tierra y  $a_E$  es el radio ecuatorial de la Tierra. El subíndice 0, se refiere a los datos para la época.

$$a_1 = \left( \frac{k_e}{n_0} \right)^{2/3} \quad (2.1)$$

$$\delta_1 = \frac{3 k_2}{2 a_1^2} \frac{3 \cos^2 i_0 - 1}{(1 - e_0^2)^{3/2}} \quad (2.2)$$

$$a_0 = a_1 \left( 1 - \frac{1}{3} \delta_1 - \delta_1^2 - \frac{134}{81} \delta_1^3 \right) \quad (2.3)$$

$$\delta_0 = \frac{3 k_2}{2 a_0^2} \frac{3 \cos^2 i_0 - 1}{(1 - e_0^2)^{3/2}} \quad (2.4)$$

$$n_0'' = \frac{n_0}{1 + \delta_0} \quad (2.5)$$

$$a_0'' = \frac{a_0}{1 - \delta_0} \quad (2.6)$$

El coeficiente balístico  $B$  se calcula sobre el término de la resistencia aerodinámica  $B^*$ :

$$B = 2B^*/\rho_0 \quad (2.7)$$

donde  $\rho_0$  es un valor de referencia de la densidad atmosférica.

A continuación, se calculan las siguientes constantes. Para ello, se extraen del código TLE: la anomalía media  $M_0$ , el argumento medio del perigeo  $\omega_0$  y la longitud media del nodo ascendente  $\Omega_0$ ; y las constantes siguientes: parámetro para la función de densidad en los modelos de SGP4 y SGP8  $s$  y  $A_{3,0} = J_3 a_E^3$ , donde  $J_3$  es el tercer armónico gravitacional de la Tierra.

$$\beta^2 = 1 - e^2 \quad (2.8)$$

$$\theta = \cos i \quad (2.9)$$

$$\dot{M}_1 = -\frac{3}{2} \frac{n'' k_2}{a''^2 \beta^3} (1 - 3\theta^2) \quad (2.10)$$

$$\dot{\omega}_1 = -\frac{3}{2} \frac{n'' k_2}{a''^2 \beta^4} (1 - 5\theta^2) \quad (2.11)$$

$$\dot{\Omega}_1 = -3 \frac{n'' k_2}{a''^2 \beta^4} \theta \quad (2.12)$$

$$\dot{M}_2 = \frac{3}{16} \frac{n'' k_2^2}{a''^4 \beta^7} (13 - 78\theta^2 + 137\theta^4) \quad (2.13)$$

$$\dot{\omega}_2 = \frac{3}{16} \frac{n'' k_2^2}{a''^4 \beta^8} (7 - 1140\theta^2 + 3950\theta^4) + \frac{5}{4} \frac{n'' k_4}{a''^4 \beta^8} (3 - 36\theta^2 + 49\theta^4) \quad (2.14)$$

$$\dot{\Omega}_2 = \frac{3}{2} \frac{n'' k_2^2}{a''^4 \beta^8} (4 - 19\theta^2) + \frac{5}{2} \frac{n'' k_4}{a''^4 \beta^8} \theta (3 - 7\theta^2) \quad (2.15)$$

$$\dot{l} = n'' + \dot{M}_1 + \dot{M}_2 \quad (2.16)$$

$$\dot{\omega} = \dot{\omega}_1 + \dot{\omega}_2 \quad (2.17)$$

$$\dot{\Omega} = \dot{\Omega}_1 + \dot{\Omega}_2 \quad (2.18)$$

$$\xi = \frac{1}{a'' + \beta^2 - s} \quad (2.19)$$

$$\eta = e s \xi \quad (2.20)$$

$$\psi = \sqrt{1 - \eta^2} \quad (2.21)$$

$$\alpha^2 = 1 + e^2 \quad (2.22)$$

$$C_0 = \frac{1}{2} B \rho_0 (q_0 - s)^4 n'' a'' \xi^4 \alpha - 1 \psi^{-7} \quad (2.23)$$

$$C_1 = \frac{3}{2} n'' \alpha^4 C_0 \quad (2.24)$$

$$D_1 = \xi p s i^{-2} / a'' \beta^2 \quad (2.25)$$

$$D_2 = 12 + 36\eta^2 9 \frac{9}{2} \eta^4 \quad (2.26)$$

$$D_3 = 15\eta^2 + \frac{5}{2} \eta^4 \quad (2.27)$$

$$D_4 = 5\eta + \frac{15}{4} \eta^3 \quad (2.28)$$

$$D_5 = \xi p s i^{-2} \quad (2.29)$$

$$B_1 = -k_2 (1 - 3\theta^2) \quad (2.30)$$

$$B_2 = -k_2 (1 - \theta^2) \quad (2.31)$$

$$B_3 = \frac{A3,0}{k_2} \sin i \quad (2.32)$$

$$C_2 = D_1 D_5 B_3 \quad (2.33)$$

$$\dot{n}_0 = C_1 \left( 2 + 3\eta^2 + 20e\eta + ee\eta^3 + \frac{17}{2}e^2 + 34e^2\eta^2 + D_1 D_2 B_1 + C_2 \cos 2\omega + C_3 \sin \omega \right) \quad (2.34)$$

$$C_4 = D_1 D_7 B_2 \quad (2.35)$$

$$(2.36)$$

$$C_5 = D_5 D_8 B_3 \quad (2.37)$$

$$D_6 = 30\eta + \frac{45}{2}\eta^3 \quad (2.38)$$

$$D_7 = 5\eta + \frac{25}{2}\eta^3 \quad (2.39)$$

$$D_8 = 1 + \frac{27}{4}\eta^2 + \eta^4 \quad (2.40)$$

$$\dot{e}_0 = -C_0 \left( 4\eta + \eta^3 + 5e + 15e\eta^2 + \frac{31}{2}e^2\eta + 7e^2\eta^3 + D_1 D_6 B_1 + C_4 \cos 2\omega + C_5 \sin \omega \right) \quad (2.41)$$

$$\frac{\dot{\alpha}}{\alpha} = e\dot{e}\alpha^{-2} \quad (2.42)$$

$$C_6 = \frac{1}{3} \frac{\dot{n}}{n''} \quad (2.43)$$

$$\frac{\dot{\xi}}{\xi} = 2a''\psi (C_6\beta^2 + e\dot{e}) \quad (2.44)$$

$$\dot{\eta} = \left( \dot{e} + e\dot{\xi}/\xi \right) s\xi \quad (2.45)$$

$$\frac{\dot{psi}}{psi} = -\eta \dot{\eta} \psi^{-2} \quad (2.46)$$

$$\frac{\dot{C}_0}{C_0} = C_6 + 4\dot{\eta}/\eta - \dot{\alpha}/\alpha - 7\psi/\psi \quad (2.47)$$

$$\frac{\dot{C}_1}{C_1} = \dot{n}/n'' - 4\dot{\alpha}/\alpha + \dot{C}_0/C_0 \quad (2.48)$$

$$D_9 = 6\eta + 20e + 15e\eta^2 + 68e^2\eta \quad (2.49)$$

$$D_{10} = 20\eta + 5\eta^3 + 17e + 68e\eta^2 \quad (2.50)$$

$$D_{11} = 72\eta + 18\eta^3 \quad (2.51)$$

$$D_{12} = 30\eta + 10\eta^3 \quad (2.52)$$

$$D_{13} = 5 + \frac{45}{4}\eta^2 \quad (2.53)$$

$$D_{14} = \dot{\xi}/\xi - 2\psi/\psi \quad (2.54)$$

$$D_{15} = 2(C_6 + e\dot{e}\beta^{-2}) \quad (2.55)$$

$$\dot{D}_1 = D_1(D_{14} + D_{15}) \quad (2.56)$$

$$\dot{D}_2 = \dot{\eta}D_{11} \quad (2.57)$$

$$\dot{D}_3 = \dot{\eta}D_{12} \quad (2.58)$$

$$\dot{D}_4 = \dot{\eta}D_{13} \quad (2.59)$$

$$\dot{D}_5 = D_5 D_{14} \quad (2.60)$$

$$\dot{C}_2 = B_2(\dot{D}_1 D_3 + D_1 \dot{D}_3) \quad (2.61)$$

$$\dot{C}_3 = B_3(\dot{D}_5 D_4 + D_5 \dot{D}_4) \quad (2.62)$$

$$\dot{\omega} = -\frac{3}{2} \frac{n''k_2}{a''^2\beta^4} (1 - 5\theta^2) \quad (2.63)$$

$$D_{16} = D_9 \dot{\eta} + D_{10} \dot{e} + B_1(\dot{D}_1 D_2 + D_1 \dot{D}_2) + \dot{C}_2 \cos 2\omega + \dot{C}_3 \sin \omega + (C_3 \cos \omega - 2C_2 \sin 2\omega) \quad (2.64)$$

$$\ddot{n}_0 = \dot{n}\dot{C}_1/C_1 + C_1 D_{16} \quad (2.65)$$

$$\begin{aligned} \ddot{e}_0 = & \dot{e}\dot{C}_0/C_0 - C_0 \left\{ \left( 4 + 3\eta^2 + 30e\eta + \frac{31}{2}e^2 + 21e^2\eta^2 \right) \dot{\eta} + (5 + 15\eta^2 + 31e\eta + 14e\eta^3)\dot{e} \right. \\ & + B_1 \left[ \dot{D}_1 D_6 + D_1 \dot{\eta} \left( 30 + \frac{135}{2}\eta^2 \right) \right] + B_2 \left[ \dot{D}_1 D_7 + D_1 \dot{\eta} \left( 5 + \frac{75}{2}\eta^2 \right) \right] \cos \omega \\ & \left. + B_3 \left[ \dot{D}_5 D_8 + D_5 \eta \dot{\eta} \left( \frac{27}{2} + 4\eta^2 \right) \right] \sin \omega + \dot{\omega} (C_5 \cos \omega - 2C_4 \sin 2\omega) \right\} \end{aligned} \quad (2.66)$$

$$D_{17} = \ddot{n}/n'' - (\dot{n}/n'')^2 \quad (2.67)$$

$$\ddot{\xi} = 2 \left( \dot{\xi}/\xi - C_6 \right) \dot{\xi}/\xi + 2a''\xi \left( \frac{1}{3}D_{17}\beta^2 - 2C_6e\dot{e} + \dot{e}^2 + e\ddot{e} \right) \quad (2.68)$$

$$\ddot{\eta} = \left( \ddot{e} + 2\dot{e}\dot{\xi}/\xi \right) s\xi + \eta \ddot{\xi}/\xi \quad (2.69)$$

$$D_{18} = \ddot{\xi}/\xi - \left( \dot{\xi}/\xi \right)^2 \quad (2.70)$$

$$D_{19} = -(\psi/\psi)^2 (1 + \eta - 2) - \eta \ddot{\eta} \psi^{-2} \quad (2.71)$$

$$\ddot{D}_1 = \dot{D}_1 (D_{14} + D_{15}) + D_1 \left( D_{18} - 2D_{19} + \frac{2}{3}D_{17} + 2\alpha^2\dot{e}^2\beta - 4 + 2e\ddot{e}\beta^{-2} \right) \quad (2.72)$$

$$\ddot{n}_0 = \dot{n} \left[ \frac{4}{3}D_{17} + 3\dot{e}^2\alpha^{-2} - 6(\dot{\alpha}/\alpha)^2 + 4D_{18} - 7D_{19} \right]$$

$$\begin{aligned} & + \dot{n}\dot{C}_1/C_1 + C_1 \{ D_{16}\dot{C}_1/C_1 + D_9\ddot{\eta} + D_{10}\ddot{e} + \dot{\eta}^2 (6 + 30e\eta + 68e^2) \} \\ & + \dot{\eta}\dot{e} (40 + 30\eta^2 + 272e\eta) + \dot{e}^2 (17 + 68\eta^2) \\ & + B_1 [\ddot{D}_1 D_2 + 2\dot{D}_1 \dot{2} + D_1 (\ddot{\eta} D_{11} + \dot{\eta}^2 (72 + 54\eta^2))] \end{aligned} \quad (2.73)$$

$$+ B_2 [\ddot{D}_1 D_3 + 2\dot{D}_1 \dot{3} + D_1 (\ddot{\eta} D_{12} + \dot{\eta}^2 (30 + 30\eta^2))] \cos 2\omega$$

$$+ B_3 \left[ (\dot{D}_5 D_{14} + D_5 (D_{18} - 2D_{19})) D_4 + 2\dot{D}_4 \dot{D}_5 + D_5 \left( \ddot{\eta} D_{13} + \frac{45}{2}\eta \dot{\eta}^2 \right) \right] \sin \omega$$

$$\dot{\omega} [(7C_6 + 4e\dot{e}\beta^{-2}) (C_3 \cos \omega - 2C_2 \sin 2\omega) 2C_3 \cos \omega$$

$$- 4C_2 \sin 2\omega - \dot{\omega} (C_3 \sin \omega + 4C_2 \cos 2\omega)$$

$$p = \frac{2\ddot{n}_0^2 - \dot{n}_0 \ddot{n}_0}{\dot{n}_0^2 - \dot{n}_0 \ddot{n}_0} \quad (2.74)$$

$$\gamma = -\frac{\ddot{n}_0}{\dot{n}_0} \frac{1}{(p-2)} \quad (2.75)$$

$$n_D = \frac{\dot{n}_0}{p\gamma} \quad (2.76)$$

$$q = 1 - \frac{\ddot{e}_0}{\dot{e}_0 \gamma} \quad (2.77)$$

$$e_D = \frac{\dot{e}_0}{q\gamma} \quad (2.78)$$

Los efectos de la resistencia aerodinámica atmosférica y la gravedad están incluidos en:

$$n = n_0'' + n_D [1 - (1 - \gamma(t - t_0))^p] \quad (2.79)$$

$$e = e_0 + e_D [1 - (1 - \gamma(t - t_0))^q] \quad (2.80)$$

$$\omega = \omega_0 + \dot{\omega}_1 \left[ (t - t_0) + \frac{7}{3} \frac{1}{n_0''} Z_1 \right] + \dot{\omega}_2 (t - t_0) \quad (2.81)$$

$$\Omega = \Omega_0'' + \dot{\Omega}_1 \left[ (t - t_0) + \frac{7}{3} \frac{1}{n_0''} Z_1 \right] + \dot{\Omega}_2 (t - t_0) \quad (2.82)$$

$$M = M_0 + n_0''(t - t_0) + Z_1 + \dot{M}_1 \left[ (t - t_0) + \frac{7}{3} \frac{1}{n_0''} Z_1 \right] + \dot{M}_2 (t - t_0) \quad (2.83)$$

Hay que tener en cuenta que  $t - t_0$  es el tiempo desde la época y donde:

$$\dot{e} = -\frac{2}{3} \frac{\dot{n}_0}{n_0''} (1 - e_0) \quad (2.84)$$

A continuación, se resuelve la ecuación de Kepler iterando lo siguiente:

$$E_{i+1} = E_i + \Delta E_i \quad (2.85)$$

donde:

$$\Delta E_i = \frac{M + e \sin E_i - E_i}{1 - e \cos E_i} \quad (2.86)$$

y

$$E_1 = M + e \sin M + \frac{1}{2} e^2 \sin 2M \quad (2.87)$$

Mediante las siguientes ecuaciones se pueden calcular la para el corto período:

$$a = \left( \frac{k_e}{n} \right)^{\frac{2}{3}} \quad (2.88)$$

$$\beta = (1 - e^2)^{\frac{1}{2}} \quad (2.89)$$

$$\sin f = \frac{\beta \sin E}{1 - e \cos E} \quad (2.90)$$

$$\cos f = \frac{\cos E - e}{1 - e \cos E} \quad (2.91)$$

$$u = f + \omega \quad (2.92)$$

$$r'' = \frac{a\beta^2}{a + e \cos f} \quad (2.93)$$

$$\dot{r}'' = \frac{nae}{\beta} \sin f \quad (2.94)$$

$$(r\dot{f})'' = \frac{na^2\beta}{r} \quad (2.95)$$

$$\delta r = \frac{1}{2} \frac{k_2}{a\beta^2} [(1 - \theta^2) \cos 2u + 3(1 - 3\theta^2)] - \frac{1}{4} \frac{A_{3,0}}{k_2} \sin i_0 \sin u \quad (2.96)$$

$$\delta \dot{r} = -n \left( \frac{a}{r} \right)^2 \left[ \frac{k_2}{a\beta^2} (1 - \theta^2) \sin 2u + \frac{1}{4} \frac{A_{3,0}}{k_2} \sin i_0 \cos u \right] \quad (2.97)$$

$$\delta I = \theta \left[ \frac{3}{2} \frac{k_2}{a^2\beta^4} \sin i_0 \cos wu - \frac{1}{4} \frac{A_{3,0}}{k_2 a \beta^2} e \sin \omega \right] \quad (2.98)$$

$$\begin{aligned} \delta u &= \frac{1}{2} \frac{k_2}{a^2\beta^4} \left[ \frac{1}{2} (1 - 7\theta^2) \sin 2u - 3(1 - 5\theta^2)(f - M + e \sin f) \right] \\ &\quad - \frac{1}{4} \frac{k_2}{a^2\beta^4} \left[ \sin i_0 \cos u (2 + e \cos f) + \frac{1}{2} + \frac{\theta^2}{\sin i_0/2 \cos i_0/2} e \cos \omega \right] \end{aligned} \quad (2.99)$$

$$\delta \lambda = \frac{1}{2} \frac{k_2}{a^2\beta^4} \left[ \frac{1}{2} (1 + 6\theta - 7\theta^2) \sin 2u - 3(1 + 2\theta - 5\theta^2)(f - M + e \sin f) \right] \quad (2.100)$$

$$+ \frac{A_{3,0}}{k_2 a \beta^2} \sin i_0 \left[ \frac{e\theta}{1 + \theta} \cos \omega - (2 + e \cos f) \cos u \right]$$

Los términos del corto período son añadidos para hallar los valores oscilantes:

$$r = r'' + \delta r \quad (2.101)$$

$$\dot{r} = \dot{r}'' + \delta \dot{r} \quad (2.102)$$

$$r\dot{f} = (r\dot{f})'' + \delta(r\dot{f}) \quad (2.103)$$

$$y_4 = \sin \frac{i_0}{2} \sin u + \cos u \sin \frac{i_0}{2} \delta u + \frac{1}{2} \sin u \cos \frac{i_0}{2} \delta I \quad (2.104)$$

$$y_5 = \sin \frac{i_0}{2} \cos u + \sin u \sin \frac{i_0}{2} \delta u + \frac{1}{2} \cos u \cos \frac{i_0}{2} \delta I \quad (2.105)$$

$$\lambda = u + \Omega + \delta \lambda \quad (2.106)$$

Se calculan los vectores de posición y velocidad:

$$U_x = 2y_4(y_5 \sin \lambda - y_4 \cos \lambda) + \cos \lambda \quad (2.107)$$

$$U_y = -2y_4(y_5 \cos \lambda + y_4 \sin \lambda) + \sin \lambda \quad (2.108)$$

$$U_z = 2y_4 \cos \frac{I}{2} \quad (2.109)$$

$$V_x = 2y_5(y_5 \sin \lambda - y_4 \cos \lambda) - \sin \lambda \quad (2.110)$$

$$V_y = -2y_5(y_5 \cos \lambda + y_4 \sin \lambda) + \cos \lambda \quad (2.111)$$

$$U_z = 2y_5 \cos \frac{I}{2} \quad (2.112)$$

$$(2.113)$$

donde

$$\cos \frac{I}{2} = \sqrt{1 - y_4^2 - y_5^2} \quad (2.114)$$

Y la posición y velocidad se dan de la siguiente manera:

$$\mathbf{r} = r\mathbf{U} \quad (2.115)$$

$$\dot{\mathbf{r}} = \dot{r}\mathbf{U} + r\dot{f}\mathbf{V} \quad (2.116)$$

# 3 Modelo dinámico de un satélite

---

El modelo dinámico escogido calcula la órbita de manera mucho más exacta, a partir de fuerzas y momentos e integrando para calcular posiciones y velocidades. Se incluyen en este modelo interacciones con el sol y las posiciones de los cuerpos celestes en el momento a simular. Este modelo dinámico se obtiene a partir de un proyecto de Simulink que se encuentra en [28]. Este proyecto está diseñado mediante programación por bloques, que es el tipo empleado en Simulink, que permite crear modelos muy complejos con realimentaciones de forma visual e intuitiva. La principal librería que utiliza es *Aerospace Blockset*, la cual contiene numerosos bloques de transformación de unidades o tipo de coordenadas, así como ecuaciones dinámicas para vehículos tales como aviones, helicópteros, vehículos multirrotores, cohetes o satélites. Este proyecto de Simulink sirve como base para simular un Cubesat de 1U de hasta 1,33 kg de masa, por lo que se adecúa perfectamente al Veery Hatchling "Clay", satélite de estudio.

Las modificaciones que se han realizado sobre el modelo original han sido las siguientes:

- Se ha fijado la masa de 0,9 kg.
- Se ha fijado la misma fecha que en el modelo simplificado de perturbaciones. En este caso se introduce la fecha juliana: 2459502,2502, que se corresponde al 14 de octubre de 2021 a las 18.00 y 17,28 segundos.
- Se ha introducido la posición y velocidad del satélite en el momento indicado del inicio de la simulación. Esta posición y velocidad se ha dado en coordenadas ECI en km y km/s, respectivamente:

$$\mathbf{r} = [-945,140,0711 \quad 6,844,261,9514 \quad 324,147,2724] [m]$$
$$\dot{\mathbf{r}} = [-5,280,9284 \quad -984,6273 \quad 5,374,6366] [m/s]$$

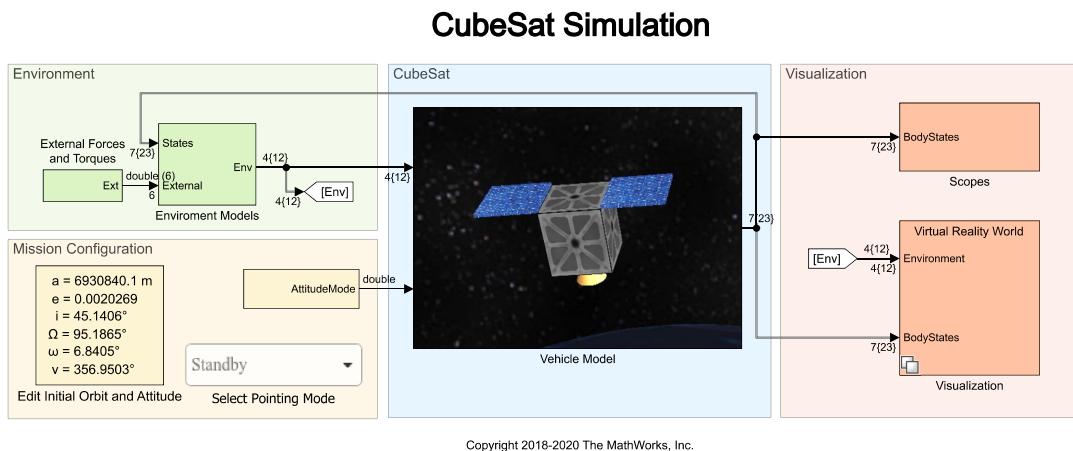
- Se ha fijado el tiempo de simulación en cuatro semanas.
- Se han incorporado en el bloque de salidas dos bloques que guardan en un archivo .mat la posición y velocidad en ejes ECI de cada minuto simulado.

## 3.1 Análisis del modelo dinámico

A primera vista, al entrar en el proyecto de Simulink aparece la capa general del modelo. Esta se divide en 4 grupos:

- **Entorno:** Bloque de color verde en el que se introducen las fuerzas y momentos externos que actúan sobre el satélite.

- **Configuración de la misión:** Bloque de color amarillo, en el que se incluyen los parámetros como el tiempo, la posición o la velocidad del satélite.
- **Dinámica del satélite:** Bloque de color azul, en el que se incluyen todas las ecuaciones y transformaciones de la dinámica de movimiento del satélite y del que salen las posiciones y velocidades en cada instante.
- **Visualización:** Bloque de color salmón donde se incluyen las gráficas de posiciones y velocidades en distintos ejes, una animación en 3 dimensiones a tiempo real de simulación del movimiento del satélite y los bloques de los que se extraen y guardan en la memoria las posiciones y velocidades a lo largo del tiempo.



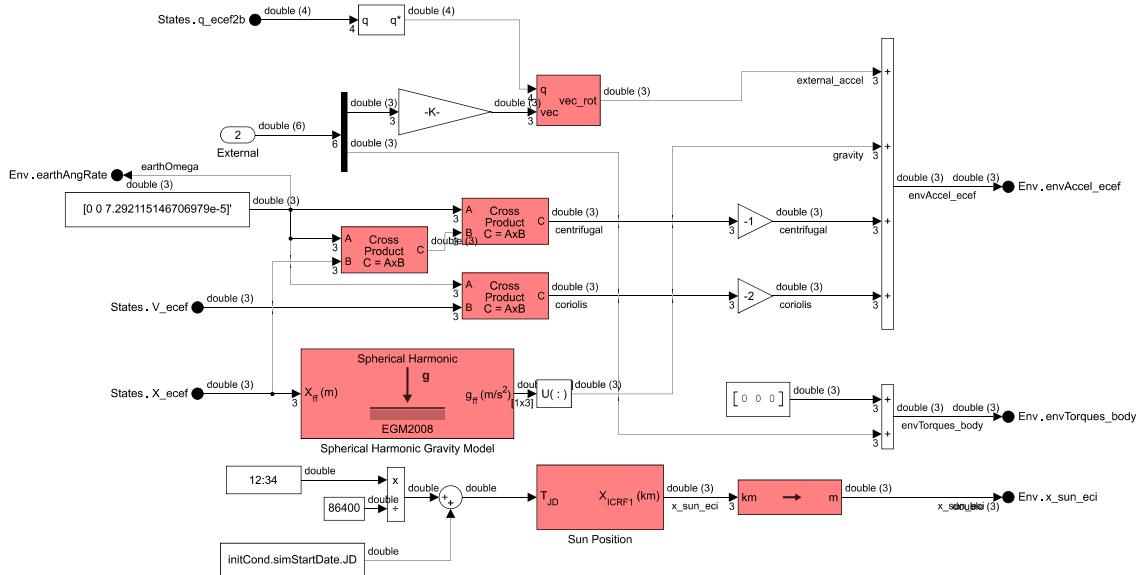
**Figura 3.1** Capa superior del proyecto de Simulink del modelo dinámico.

### 3.1.1 Entorno

En el bloque más a la izquierda (figura 3.1), se introducen dos vectores nulos en fuerzas y momentos, ya que se considera que el sistema no va a sufrir perturbaciones externas ajenas a las propias que influyen en la órbita del satélites (por ejemplo gravedad, resistencia aerodinámica o presión solar).

En el siguiente bloque del entorno figura 3.1) se introducen las fuerzas y momentos externos (aunque van a ser nulos) procedentes del bloque anteriormente mencionado, y la realimentación de los estados anteriores del entorno, y se extraen las aceleraciones, momentos y posición solar del entorno en ejes ECEF (Earth-Centered Earth Fixed coordinates). A continuación se muestra la visualización del interior de este bloque, y se procede a su explicación:

Empezando por la parte superior del esquema (figura 3.2), se toman la velocidades de rotación del entorno y se transforman a cuaterniones. Este cuaternion se rota con las aceleraciones externas (nulas en este caso), que se han hallado dividiendo las fuerzas externas entre la masa del vehículo; para dar como resultado las aceleraciones externas.



**Figura 3.2** Bloque principal de representación del entorno del satélite.

A continuación, tomando la realimentación del movimiento angular de la Tierra, la velocidad angular de rotación de la Tierra (vector con única componente no nula en el tercer parámetro, correspondiente a 7,292 rad/s), y los estados realimentados de velocidad y posición en ejes ECEF; mediante productos cruzados, se hallan las aceleraciones de Coriolis y centrífuga. Estas aceleraciones vienen descritas por las ecuaciones:

$$a_{co} = 2\omega \times v \quad (3.1)$$

$$a_{ce} = \omega^2 r \quad (3.2)$$

donde  $\omega$  es la velocidad de rotación de la Tierra,  $v$  es la velocidad no inercial del satélite y  $r$  es el radio de la órbita en ese momento.

La última aceleración en ser calculada es la de la gravedad, que viene definida por el modelo EGM2008, que se trata de un modelo que trata la Tierra como un geoide, por lo que se puede calcular la aceleración de la gravedad de manera muy precisa.

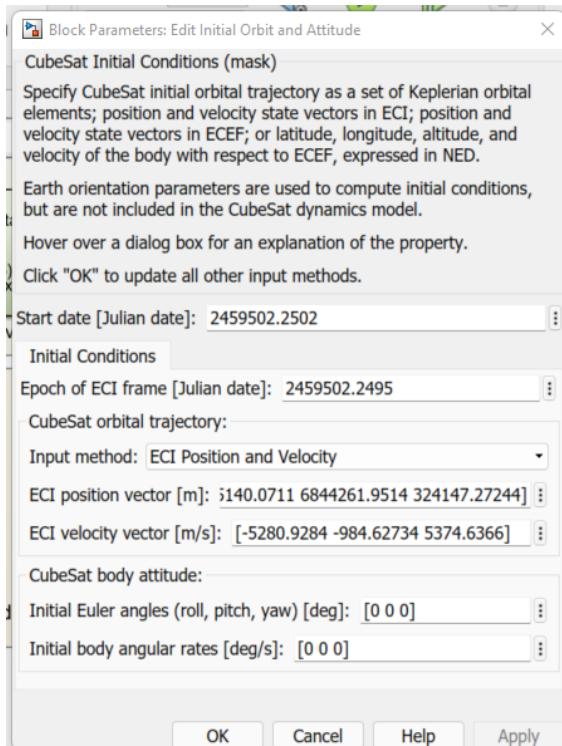
Por último, estas aceleraciones se suman para tener las aceleraciones que ejerce el entorno sobre el satélite en coordenadas ECEF.

A continuación, hay una suma de dos vectores nulos para calcular los momentos, por lo que el momento del entorno sobre el cuerpo es nulo.

Finalmente, utilizando un bloque que representa un reloj y fijando la fecha inicial, se coloca un bloque que establece la posición del Sol en coordenadas ECI (importante tener en cuenta esto para posteriores operaciones).

### 3.1.2 Configuración de la misión

Este sistema es quizá el más simple, pero a su vez tiene un vital importancia para el correcto funcionamiento global. En él se introducen los datos del momento exacto de inicio de la simulación y la posición y velocidad del objeto. Esto se introduce haciendo doble click en el bloque izquierdo del sistema de configuración de la misión de la figura 3.1. Se puede observar que en este cuadro aparece la designación de actitud completa en forma de parámetros de Kepler. Esto siempre aparecerá así en esta capa, aunque se introduzca mediante coordenadas ECI o ECEF. El cuadro de diálogo que aparece se puede ver en la figura 3.3.



**Figura 3.3** Cuadro de diálogo para escoger los parámetros de posicionamiento del satélite e instante de inicio de la simulación.

El instante de inicio de la simulación se introduce en modo fecha juliana. También hay que determinar la época, y los 6 parámetros necesarios, que se pueden introducir en coordenadas ECI, coordenadas ECEF, parámetros de Kepler o coordenadas geodéticas.

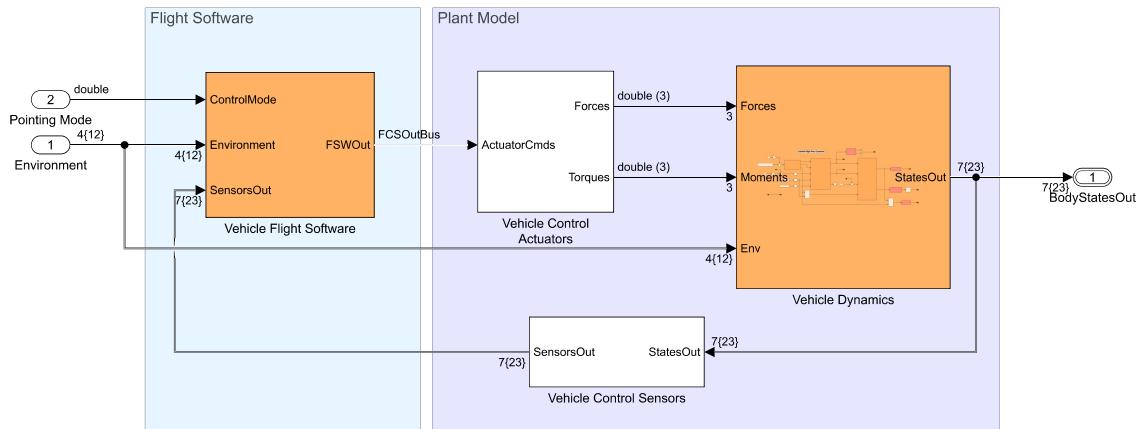
Además, se puede introducir una actitud de satélite en forma de ángulos de cabeceo, balance y guiñada; o velocidades angulares, que no son necesarias para este estudio.

En el bloque de la derecha, sólo se introduce si se requiere que el satélite se oriente hacia algún astro, lo que tampoco es necesario, por lo que en el desplegable se deja en *StandBy*.

### 3.1.3 Dinámica del satélite

En este sistema solo existe un bloque, en el que aparece una representación gráfica de un Cubesat (figura 3.1). En este bloque se encuentra la parte más importante del proyecto, y quizás la más compleja.

Entrando dentro de este sistema, se pueden observar dos sistemas diferenciados. Los inputs del modelo dinámico son los datos de entorno y el astro al que apunta (ninguno en el caso de estudio).



**Figura 3.4** Interior del bloque del modelo del satélite.

En el bloque de la izquierda de la figura 3.4 se encuentra el software de vuelo, en el que entran los datos de entorno, la información sobre el astro al que se apunta, y la realimentación de los sensores del satélite.

En el interior del bloque del software de vuelo se puede observar el sistema de control de actitud (figura 3.5), en el que a partir de los datos de entrada genera las señales de error de actitud, momentos y control de altitud.

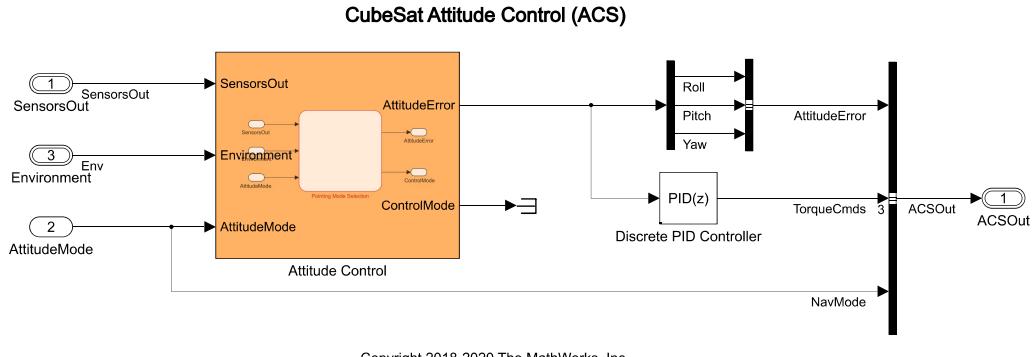
A la derecha, en el sistema morado de la figura 3.4 encontramos tres bloques. El bloque superior izquierdo se encarga de traducir las órdenes de los actuadores a fuerzas y momentos resultantes, que entran a la dinámica del vehículo. El sistema inferior traduce las variables de salida del modelo dinámico a impulsos de sensores, para entrar en el software del vehículo como realimentación.

En la parte superior a la derecha, nos encontramos el bloque que incluye la dinámica completa del vehículo, con entradas de fuerzas, momentos y entorno (fuerzas y momentos del entorno).

Entrando en el interior del bloque podemos encontrar (figura 3.6):

Empezando por la esquina superior izquierda se puede observar que a partir del instante de inicio de simulación se calcula el tiempo, del que se obtienen las posiciones y velocidades en sistema ICRF (International Celestial Reference Frame, Marco de Referencia Celeste Internacional), a partir del modelo ITRF (International Terrestrial Reference Frame, Marco de Referencia Terrestre Internacional). En las salidas se tienen posiciones, velocidades y matriz de cosenos directores (DCM) de este sistema.

Debajo de lo anterior, se toman las fuerzas, y dividiéndolas entre la masa del satélite se obtienen las aceleraciones. Estas se suman a las aceleraciones del entorno. El sistema ITRF de posiciones y velocidades, las aceleraciones y las condiciones iniciales del posiciones y velocidades en coordenadas



Copyright 2018-2020 The MathWorks, Inc.

**Figura 3.5** Sistema de control de actitud.

ECI se introducen en el sistema de dinámica translacional. Este bloque viene predeterminado en el *Aerospace Blockset* y se basa en la ecuación:

$$F_{ext} - M \cdot a_p = M \cdot a_{cm} \quad (3.3)$$

donde  $M$  es la masa del sistema completo,  $a$  es la aceleración,  $p$  es el punto de observación y  $cm$  es el centro de masas del sistema completo.

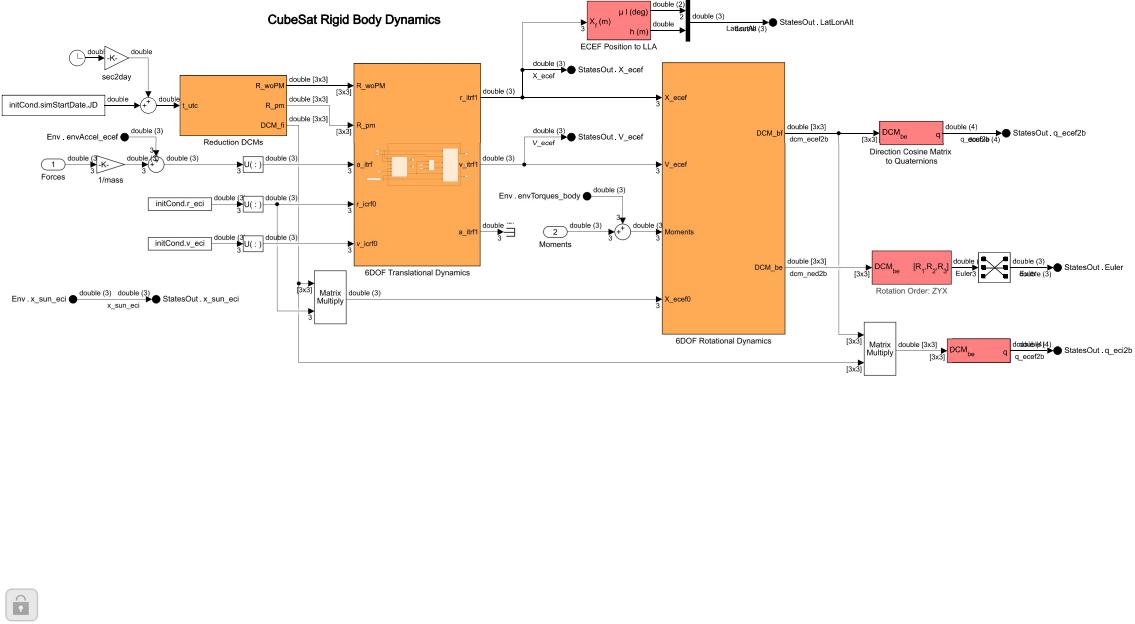
De este bloque salen las posiciones, velocidades y aceleraciones en marco terrestre, coordenadas ECEF.

Debajo de lo analizado hasta ahora se encuentra un sistema que simplemente traslada la posición del Sol a la salida.

A su derecha se puede encontrar un bloque que multiplica la matriz de cosenos directores con la posición inicial en coordenadas ECI, para obtener las coordenadas de posición iniciales ECEF.

Ligeramente a la derecha, en la parte superior, se produce una transformación de posición de coordenadas ECEF a latitud, longitud y altitud, que se lleva directamente a las salidas.

El siguiente bloque a la derecha es el de la dinámica rotacional del vehículo. Las entradas son las posiciones y velocidades en coordenadas ECEF extraídas de la dinámica translacional, los momentos a partir de los momentos del entorno y los del vehículo, y la posición inicial en coordenadas ECEF. Este también se trata de un bloque que se encuentra por defecto en el *Aerospace Blockset*, y cuyas ecuaciones que encierra son:



**Figura 3.6** Sistema de dinámica del vehículo.

$$\vec{L}_P = M\vec{r}_{CM,P} \times \vec{v}_{CM,P} + \vec{L}_{CM} \quad (3.4)$$

$$\dot{\vec{L}}_P = \vec{\tau}_{P,I} - Mr_{CM,P} \times \ddot{\vec{r}}_P = M\vec{r}_{CM,P} \times \vec{a}_{CM,P} + \vec{L}_{CM} \quad (3.5)$$

$$\vec{\tau}_{CM,B} = I_B \dot{\vec{\omega}}_{BIB} + \vec{\omega}_{BIB} \times (I_B \vec{\omega}_{BIB}) \quad (3.6)$$

donde  $\vec{L}_P$  es el momento angular del cuerpo en un punto  $P$ ,  $M$  es la masa del cuerpo,  $\vec{r}_{CM,P}$  es el vector posición del centro de masas con respecto al punto  $P$ ,  $\vec{v}_{CM,P}$  es el vector velocidad del centro de masas con respecto al punto  $P$ ,  $\vec{a}_{CM,P}$  es el vector aceleración del centro de masas con respecto al punto  $P$ ,  $\vec{L}_{CM}$  es el momento angular del cuerpo en el centro de masas,  $\vec{\tau}_{P,I}$  es el momento en el cuerpo con respecto del punto  $P$  en un marco inercial,  $\vec{\tau}_{CM,B}$  es el momento en el cuerpo en ejes cuerpo respectivo del centro de masas,  $I_B$  es la matriz de momentos de inercia en ejes cuerpo y  $\vec{\omega}_{BIB}$  es la velocidad angular en ejes cuerpo con respecto a un marco inercial en ejes cuerpo.

De este bloque se extraen las matrices de cosenos directores, que se transforman en cuaterniones en coordenadas ECI y ECEF y en ángulos de Euler.

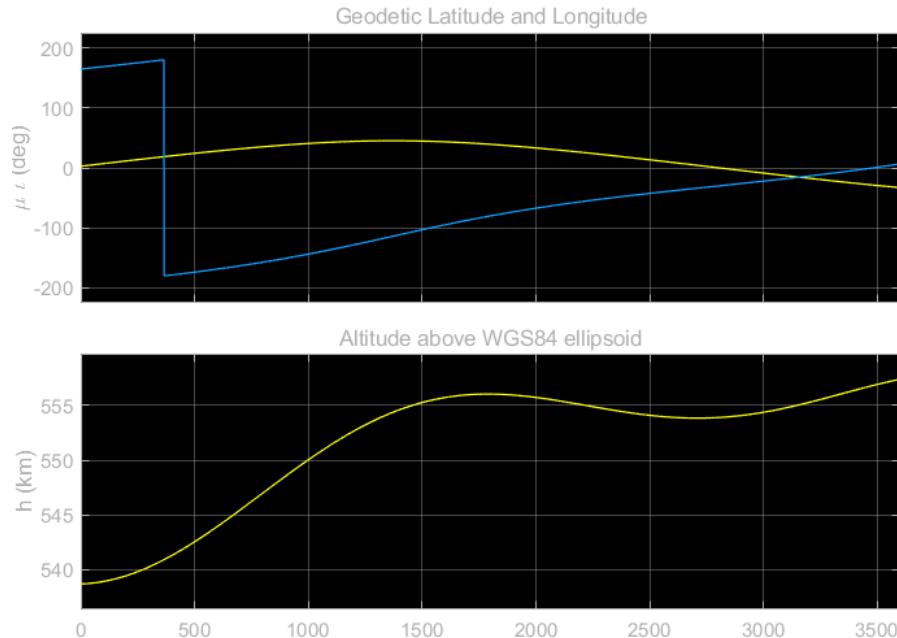
De esta manera, ya se tendría toda la información del estado del satélite en diferentes ejes actualizándose a lo largo del tiempo. Esta información puede introducirse a continuación en el módulo de visualización.

### 3.1.4 Módulo de visualización

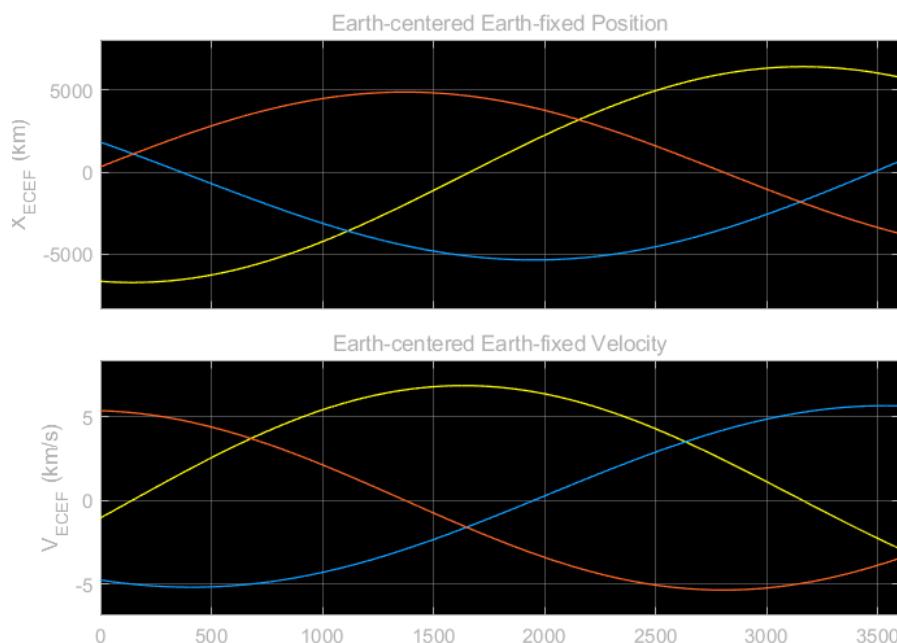
Por último, en este módulo se recogen los resultados y se representan de diversas formas para entender el funcionamiento del vehículo en órbita con mayor calidad y para dar el uso requerido

a la información obtenida. En la figura 3.1, podemos observar que este módulo se compone de dos bloques. En el bloque superior se encuentran las representaciones gráficas de los resultados obtenidos en tres formas: coordenadas geodéticas (latitud, longitud, altitud), posición y velocidad en ejes ECEF; y posición y velocidad en ejes ECI.

A continuación, se muestra la evolución de los parámetros de localización en los tres tipos de coordenadas a lo largo de una hora (figuras 3.7, 3.9 y 3.8).

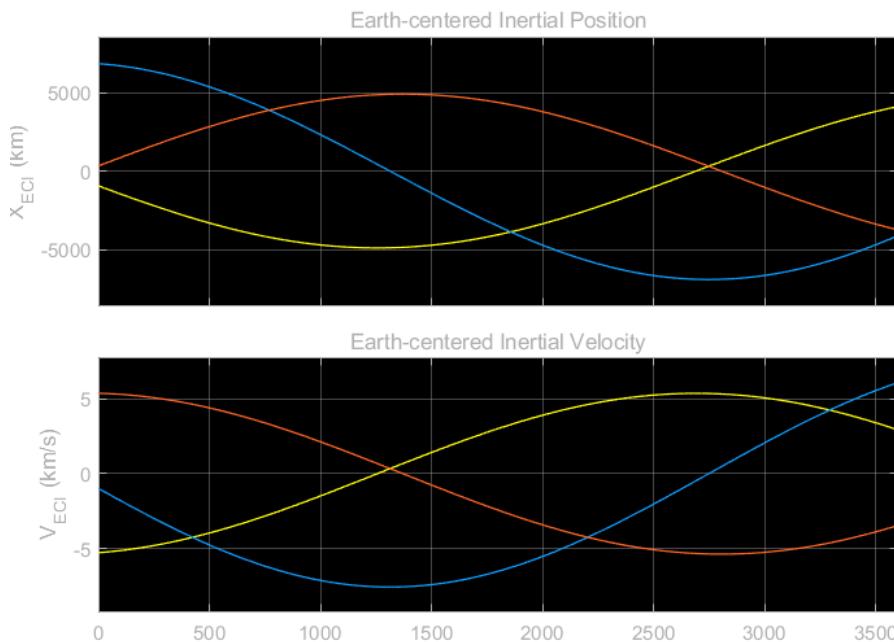


**Figura 3.7** Evolución de coordenadas geodéticas durante una hora de simulación.

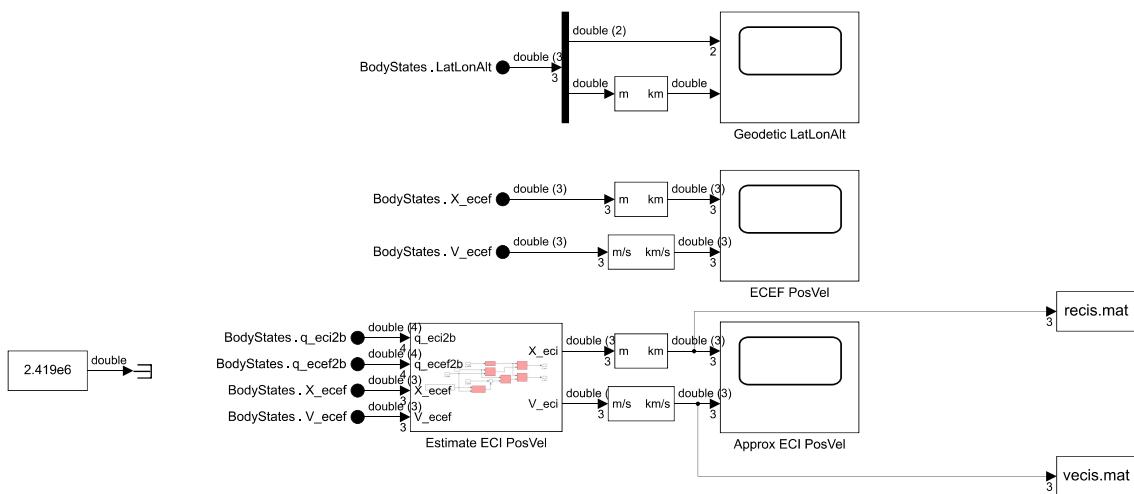


**Figura 3.8** Evolución de coordenadas ECEF durante una hora de simulación.

Además, en este bloque se han incluido dos sistemas que almacenan minuto a minuto las coordenadas de la posición y la velocidad en coordenadas ECI (figura 3.10):

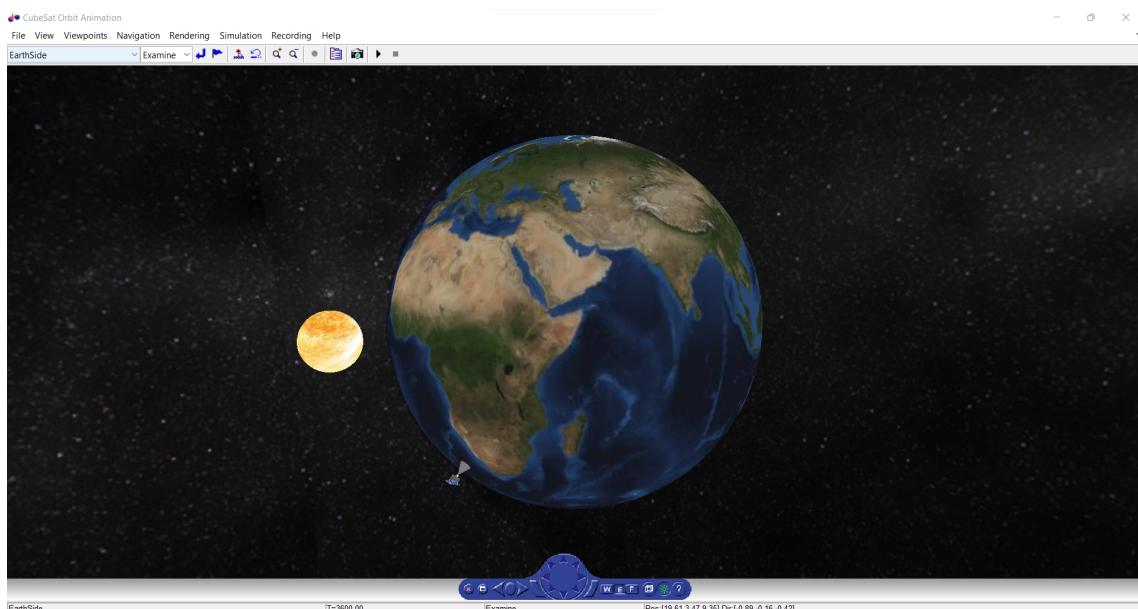


**Figura 3.9** Evolución de coordenadas ECI durante una hora de simulación.



**Figura 3.10** Bloque de gráficas y sistema de almacenamiento de información.

El bloque inferior, de este módulo (figura 3.1) se encarga de crear una animación en 3 dimensiones en tiempo real del movimiento de una representación gráfica del CubeSat alrededor de una representación gráfica de la Tierra (figura 3.11).



**Figura 3.11** Representación gráfica del satélite orbitando alrededor de la Tierra.

## 4 Comparación de modelos

---

Una vez descritos ambos modelos, se puede realizar una breve comparación entre ellos. De cara a la aplicación de las técnicas de machine learning, se va a tomar el modelo dinámico como el comportamiento real del satélite, mientras que se van a observar cómo se va desviando la trayectoria del modelo de perturbaciones simplificado a medida que aumenta el tiempo.

Para comenzar con esta comparación, podemos observar sendas capturas de las dos trayectorias durante una hora, en dos momentos clave: al principio y al final de la simulación:



**Figura 4.1** Trayectorias durante la primera hora de simulación del modelo de perturbaciones simplificado y el modelo dinámico.

De estas figuras se pueden extraer una conclusión: las distancias entre órbitas no son lo suficientemente grandes como para apreciarse en comparación con el tamaño del planeta Tierra.

También se puede observar como ha cambiado la órbita del cuerpo durante 4 semanas de simulación, puesto que en un momento inicial la órbita avanza de manera paralela a la costa oeste de Estados Unidos, mientras que al final de la simulación, ésta es prácticamente perpendicular al inicio y divide América del Sur en dos de este a oeste.

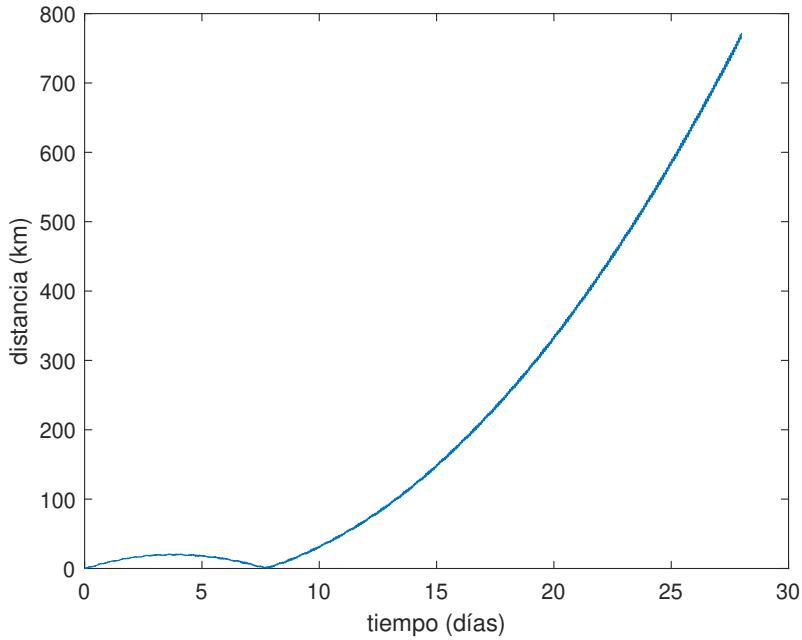
Por tanto, si queremos observar de manera gráfica las diferencias de órbitas entre el satélite predicho por el modelo simplificado de perturbaciones y el modelo dinámico del satélite, convendrá realizar una gráfica con el aumento de la distancia a lo largo del tiempo, y su equivalente en velocidad, que, podría asemejarse a la diferencia entre los módulos de las velocidades.



**Figura 4.2** Trayectorias durante la última hora de simulación del modelo de perturbaciones simplificado y el modelo dinámico.

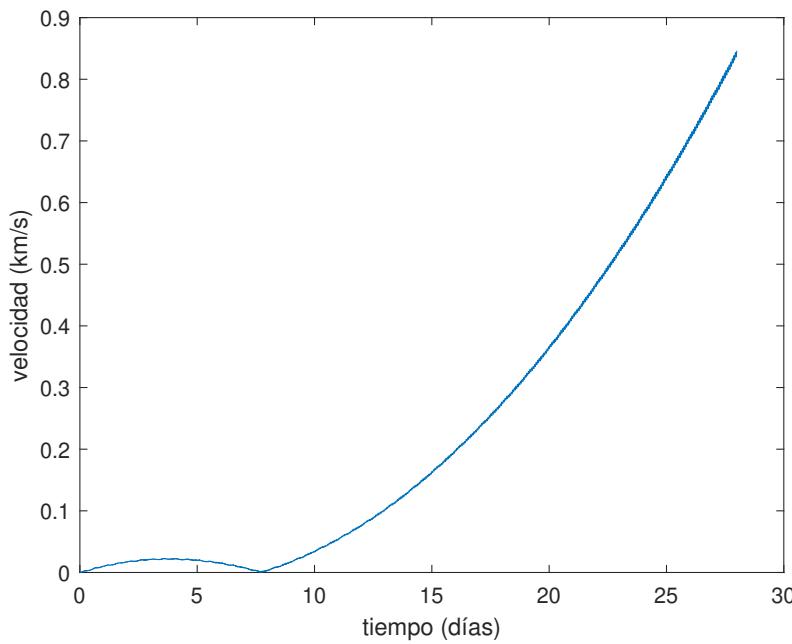
Para hallar esta distancia se aplicará esta sencilla fórmula, tanto en el caso de las posiciones como de las velocidades:

$$d(P_1, P_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \quad (4.1)$$



**Figura 4.3** Distancia entre la trayectoria de los dos modelos a lo largo del tiempo.

Se puede observar que el modelo simplificado de perturbaciones funciona de manera notable durante un período de alrededor de una semana. Sin embargo, a partir de ahí el error de posición y de velocidad aumenta en gran medida. Calculando el módulo de la velocidad en el último instante de simulación



**Figura 4.4** Diferencia entre los módulos de la velocidad de los dos modelos a lo largo del tiempo.

$$|\vec{U}| = \sqrt{u_1^2 + u_2^2 + u_3^2} \quad (4.2)$$

El módulo de velocidad en el último minuto, se obtiene un valor de 7,6 km/s, por lo que el error en la velocidad es de grandes dimensiones.

A partir de este momento, podemos comenzar a aplicar los métodos de Machine Learning para conseguir mejores predicciones y más longevas que las ofrecidas por el modelo de perturbaciones simplificado.



# 5 Técnicas de Machine Learning

---

En este capítulo, se van a describir los algoritmos de las técnicas de Machine Learning empleadas para obtener la predicción de órbitas con mayor detalle al introducido en el apartado 1.5. Se empezará por técnicas simples que, a priori, deberían ofrecer peores resultados y gradualmente se utilizarán algoritmos más complejos para llegar al mejor modelo posible para una predicción longeva. Para ello, se tomarán dos aproximaciones, descritas en el capítulo 1.1.

Es importante tener en cuenta que además de predecir las posiciones en el futuro, también se predecirán las velocidades, si bien esta última información no es tan relevante a la hora de predecir colisiones entre objetos residentes en el espacio, por lo que se considerarán buenas predicciones aquellas que menor error absoluto tengan con respecto a la posición real del satélite dada por el modelo dinámico.

## 5.1 Primera aproximación

En la primera aproximación, se utilizarán tanto los datos reales del modelo dinámico como los del modelo simplificado de perturbaciones, propagados como entradas del modelo de Machine Learning. A continuación, se introducirán los datos calculados por el modelo simplificado en un futuro consecuente al introducido en el modelo de Machine Learning, y éste corregirá los errores del modelo simplificado, de manera que se acerque más a la posición real del satélite.

En esta aproximación, se opta por utilizar modelos de regresión simples, que sean capaces de mejorar la información presentada por el modelo de perturbaciones, sobre todo en la segunda semana.

### 5.1.1 Regresión Ridge

Es uno de los modelos más simples a la hora de implementar. Conociendo la expresión a minimizar:

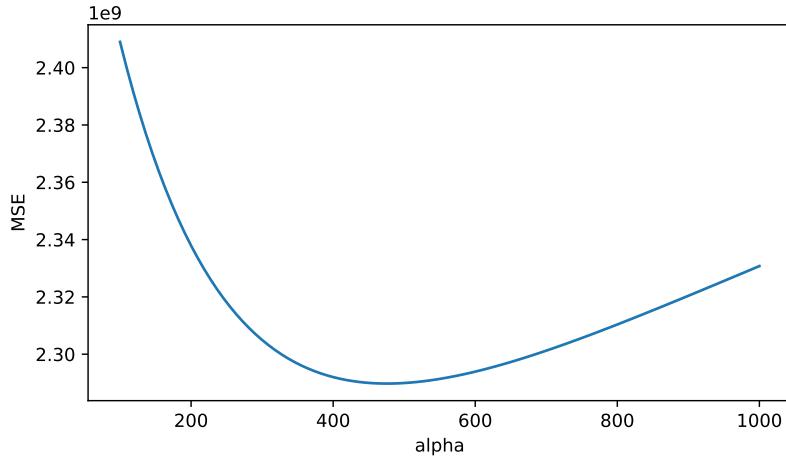
$$\|Y - X s\|^2 + \alpha \|s\|^2 \quad (5.1)$$

Se buscará el valor de  $\alpha$  para el cual el error sea mínimo. Se va a utilizar el error absoluto (Root Mean Squared Error, RMSE) como elemento para medir este error:

$$RMSE = \sqrt{(x_{real} - x_{estimada})^2 + (y_{real} - y_{estimada})^2 + (z_{real} - z_{estimada})^2} \quad (5.2)$$

Se utiliza el valor del RMSE acumulado durante toda la simulación:

Se observa un mínimo en la gráfica 5.1, que se corresponde con el valor de  $\alpha = 475$ , que es el que se fijará para hallar el modelo.



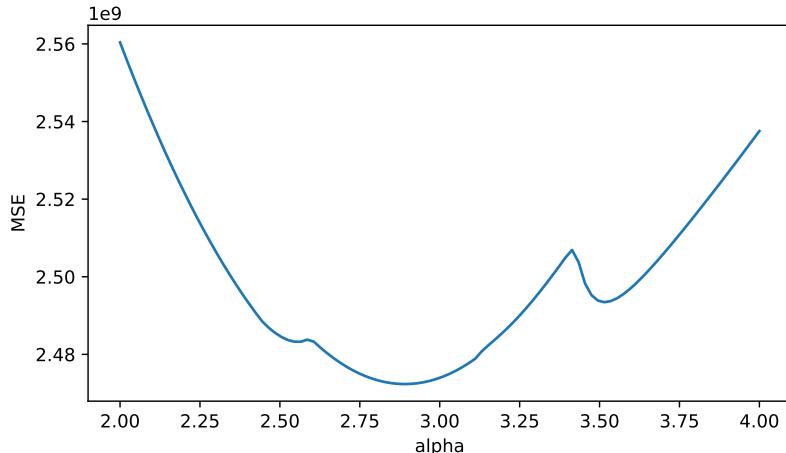
**Figura 5.1** Error cuadrático medio con respecto al parámetro  $\alpha$  para el modelo de regresión Ridge.

### 5.1.2 Regresión Lasso

Este modelo, como se ha explicado en 1.5 es muy similar a la regresión Ridge, solo que en este caso la función a minimizar es la siguiente:

$$\|y - X s\|^2 + \alpha \|s\| \quad (5.3)$$

De esta manera, se crea un barrido de factor  $\alpha$  y se encuentra el que produce un MSE mínimo:



**Figura 5.2** Error cuadrático medio con respecto al parámetro  $\alpha$  para el modelo de regresión Lasso.

Se observa un mínimo en la gráfica 5.2, que se corresponde con el valor de  $\alpha = 2.9$ , que es el que se fijará para hallar el modelo.

## 5.2 Segunda aproximación

En esta aproximación, se utilizarán solamente los datos tomados como reales del modelo dinámico. Se creará el modelo utilizando datos de posición de 8 semanas y dividiendo estos datos aleatoriamente la muestra entre entrenamiento y test. Esto creará un modelo, que será capaz de propagar los datos de manera indefinida a partir de una sola rotación, en este caso de alrededor de 96 minutos.

### 5.2.1 Gradient Boosting Regression Tree (GBRT)

El algoritmo de GBRT está incluido en una función de Python, en *ScikitLearn*, que se llama *GradientBoostingRegressor*. Sin embargo, se va a explicar el funcionamiento de este algoritmo, para una mejor comprensión del mismo:

- **Paso 1:** Se comienza con una hoja (analogía con un árbol) en la que se sitúa la media de la variable que se quiere predecir. Se fijará como base para hallar una solución en pasos posteriores.
- **Paso 2:** Se calculan los residuales de cada una de las muestras. Estos residuales son la diferencia entre el valor calculado (en este caso, la media de la variable que se quiere predecir) y el valor que tiene la variable en ese caso.
- **Paso 3:** Se construye el árbol de decisiones. En este árbol, los nodos intermedios se corresponden a clasificaciones del tipo (valor>cifra) (para un problema de clasificación en vez de regresión, esta elección no es para valores continuos). De esta manera, se llega a los nodos terminales u hojas con los valores de los residuales posibles después de todas las elecciones. Si existen más valores de residuales que nodos terminales, se pueden agrupar haciendo la media con otro valor de residuos.
- **Paso 4:** El conjunto total se separa en muestras, y todas ellas se pasan por el árbol de decisión. Los residuos en cada caso se van utilizando para predecir la variable objetivo.

Para lograr una varianza más baja, se introduce un parámetro llamado tasa de aprendizaje (learning rate). Cada residuo se multiplicará por esa tasa de aprendizaje (valor menor a la unidad). Cuanto más pequeña sea la tasa de aprendizaje, menores son los cambios entre una iteración y otra, lo que permite evitar mínimos locales y llegar a una solución adecuada.

- **Paso 5:** Cada vez que un conjunto pasa a través del árbol de decisión, se genera un nuevo valor de la variable predicha, lo cual genera un nuevo valor de residuos, que son los que se van introduciendo en cada nueva iteración. Por lo tanto, como podemos observar se trata de un proceso que se realiza en serie, no se puede computar de forma paralela, hay que tener el valor de los cálculos anteriores.
- **Paso 6:** Se itera entre los pasos 3 y 5 hasta que entre una iteración y la siguiente no se mejore la predicción.
- **Paso 7:** Finalmente, se utilizarán los valores de todos los árboles para llegar a la predicción final sobre la variable objetivo.

Esto se realizará utilizando la siguiente fórmula: media del primer paso sumada a todos los residuales de todos los árboles multiplicados por la tasa de aprendizaje.

A continuación, se detallan las fórmulas matemáticas que permiten llegar a la función que aproxima el algoritmo GBRT [10]:

Las entradas al sistema son: el conjunto de entrenamiento  $[(x_i, y_i)]_{i=1}^n$ , una función de pérdida  $L(y, F(x))$ , el número de iteraciones  $M$  y la tasa de aprendizaje  $\alpha$ .

Los pasos a seguir del algoritmo son:

- 1. Se inicializa la función con una constante:

$$F_0(X) = \arg \min_{\lambda} \sum_{i=1}^n L(y_i, \lambda) \quad (5.4)$$

- 2. Desde  $m = 1$  a el número de iteraciones,  $M$ :

- a. Se calculan los residuales:

$$r_{im} = - \left[ \frac{\delta L(y, F(X_i))}{\delta F(X_i)} \right]_{F(X) = F_{m-1}(X)} \quad (5.5)$$

para  $i = 1, \dots, n$

- b. Se selecciona el modelo débil, en este caso los árboles de decisión, que se representan como  $h_m(x)$ , y se entrena hallando los residuales  $[(X_i, r_{im})]_{i=1}^n$
- c. Mediante el método de optimización unidimensional, se calcula el factor  $\lambda_m$

$$\lambda_m = \arg \min_{\lambda} \sum_{i=1}^n L(Y_i, F_{m-1}(X_i) + \lambda_m h_m(x_i)) \quad (5.6)$$

$$F_m(X) = \alpha \lambda_m(X). \quad (5.7)$$

- d. Por último se actualiza el modelo:

$$F_m(X) = F_{m-1}(X) + \lambda_m h_m(X) \quad (5.8)$$

- 3. Se obtiene la función de salida  $F_M(X)$ .

$$F_M(X) = \sum_{m=0}^M F_m(X) \quad (5.9)$$

El código para aplicar esta técnica está formado por cuatro programas. Cada uno de los tres primeros se encarga de entrenar la posición en los 3 ejes por separado. Estos además, guardan los respectivos modelos, que son cargados en el cuarto programa, donde se realiza la propagación y se calcula el error.

A continuación, se explican brevemente las entradas en el código:

- **Valores a propagar:** Número de valores en el tiempo que se utilizan para propagar el siguiente minuto. En este caso se han escogido 96.
- **Número de árboles:** cantidad de modelos débiles a implementar. En este caso se utilizan 1500.
- **Profundidad del árbol:** Número de niveles del árbol contando los de nodos terminales, nodos intermedios y nodo raíz. Se han introducido 15.
- **Iteraciones sin cambio:** Iteraciones en las que un modelo no mejora para que se considere adecuado el modelo. 10 en este caso.
- **Fracción de validación:** Fracción de los datos utilizados para la comprobación del modelo. Un 20% en este caso.
- **Tasa de aprendizaje:** Se ha establecido en 0,01.
- **Tolerancia:** Valor a partir del cual se considera despreciable el cambio entre iteración e iteración. Se ha establecido en  $1e-7$ .
- **Función de pérdida:** el modelo empleado es el de error al cuadrado.

Cuanto más grandes sean estos parámetros, más preciso puede llegar a ser el modelo. Se han escogido estos en base a la capacidad de computación del PC usado para el desarrollo de este proyecto. El siguiente modelo empleado puede ayudar a conseguir mejores resultados para la misma capacidad computacional.

### 5.2.2 XGBoost

El algoritmo del método XGBoost hace básicamente lo mismo que el de GBRT sólo que en vez de hacer cálculos en serie únicamente, realiza estos cálculos también en paralelo. Por lo tanto, se prevén unos resultados similares que en el anterior modelo. La ventaja de este modelo es que puede realizar los cálculos de manera más rápida. Por tanto, con resultados similares, se preferiría este método por su eficiencia.

Al igual que en el caso de GBRT, en este caso también se han creado cuatro programas, uno para cada coordenada de posición y el cuarto donde se evalúa el funcionamiento en conjunto de los tres modelos creados, uno para cada coordenada.

A continuación, se explican las variables de entrada en este programa, muchas de ellas similares al anterior caso:

- **Objetivo:** Elemento en el que se va a fijar el modelo a la hora del aprendizaje. En este caso, el error al cuadrado.
- **Valores a propagar:** Número de valores en el tiempo que se utilizan para propagar el siguiente minuto. En este caso se han escogido 96.
- **Número de árboles:** cantidad de modelos débiles a implementar. En este caso se utilizan 10000.
- **Profundidad del árbol:** Número de niveles del árbol contando los de nodos terminales, nodos intermedios y nodo raíz. Se han introducido 25.
- **Fracción de muestra de entrenamiento:** Fracción de los datos utilizados para entrenar el modelo. Un 60% en este caso.
- **Tasa de aprendizaje:** Se ha establecido en 0,01.

Para esta técnica, se han podido fijar unos hiperparámetros que profundicen más en el aprendizaje, debido a la eficacia computacional de este modelo, por lo que se esperan mejores resultados que para el XGBRT.

### 5.2.3 Support Vector Regression (SVR)

Para la aplicación de este método se ha utilizado una función de *SciKitLearn* llamada *SVR*, en la que se juega con los parámetros  $\varepsilon$  y  $C$ , que aparecen en la fórmula de términos a minimizar y las restricciones:

$$\min \frac{1}{2} \|s\|^2 + C \sum_{i=1}^N (\xi_i + \xi_i^*) \quad (5.10)$$

$$|Y_i - s_i X_i| \leq \varepsilon + |\xi_i| \quad (5.11)$$

La  $s$  indica la pendiente del hiperparámetro, la  $C$  es la constante de equilibrio entre la tolerabilidad de desviaciones del vector de soporte y la regularidad de la función,  $\xi$  y  $\xi^*$  controlan el error al aproximar la función a las bandas y  $\varepsilon$  controla la distancia de los vectores de soporte al hiperplano. Se puede observar una representación gráfica de los parámetros en la figura 1.6.

Este algoritmo es muy simple. El principal problema que surge, es casi nunca se trata de un problema unidimensional o lineal. Para conseguir que este sencillo algoritmo funcione se le debe aplicar una función kernel, que es la que adecúa el entorno a la aplicación del SVR. Se puede observar la representación gráfica también en la figura 1.6.

En el caso de estudio, se ha observado que los parámetros  $C$  y  $\varepsilon$  son muy poco sensibles. Variando en gran cantidad sus valores no se aprecian mejoras en el funcionamiento de la técnica. Finalmente, se escogieron:  $C = 1$  y  $\varepsilon = 0,2$ .

En lo relativo a la función kernel, se ha escogido la *Radial Basis Function* (RBF). La rutina SVR de *SciKitLearn* lo ejecuta automáticamente, así que se va a proceder a explicar algo más de esta función. Es el kernel más difundido y empleado debido a su similitud con una distribución gaussiana.

A continuación, se presenta la función para dos puntos  $x_1$  y  $x_2$ , donde se calcula la similitud entre ambos o su cercanía:

$$K(x_1, x_2) = \exp -\frac{\|x_1 - x_2\|^2}{2\sigma^2} \quad (5.12)$$

donde:  $\sigma$  es la varianza y se utiliza como hiperparámetro; y  $\|x_1 - x_2\|$  es la distancia euclídea entre los puntos.

Debido a esto, el valor máximo que puede alcanzar la función kernel es 1, y se da cuando la distancia entre puntos es nula. A medida que estos puntos se alejan, la función kernel se va acercando a 0. El hiperparámetro regula la evolución entre el 1 y el 0.

# 6 Resultados

---

Una vez que se ha entendido el funcionamiento de cada una de las técnicas empleadas, se procederá a la exposición de los resultados obtenidos y posterior análisis, buscando aquella técnica que mejor se adapta para predecir una órbita de un CubeSat.

## 6.1 Presentación de resultados

En primer lugar, se van a presentar los valores obtenidos de los errores a lo largo del tiempo para ambas aproximaciones y para cada una de las técnicas empleadas en cada aproximación. El objetivo, además de la introducción a los resultados es, la apreciación de la evolución de las técnicas empleadas hacia modelos más complejos y de los que se esperan mejores resultados.

### 6.1.1 Primera aproximación

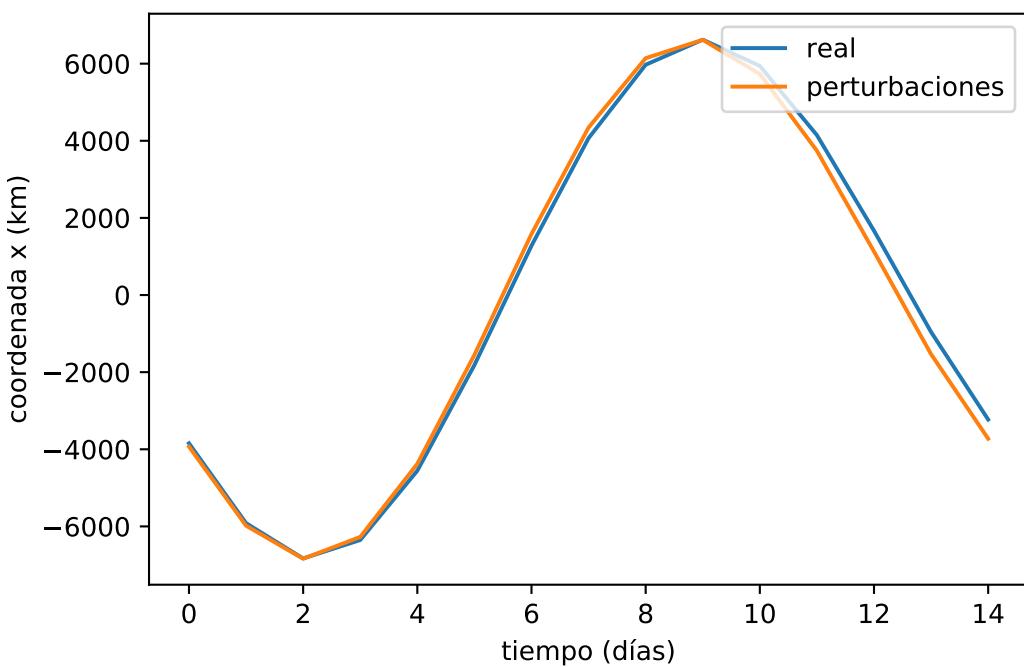
De los siguientes modelos, se van a ofrecer los datos de los errores absolutos (RMSE) de los modelos con respecto a los errores que comete el modelo simplificado de perturbaciones:

#### Modelo simplificado de perturbaciones

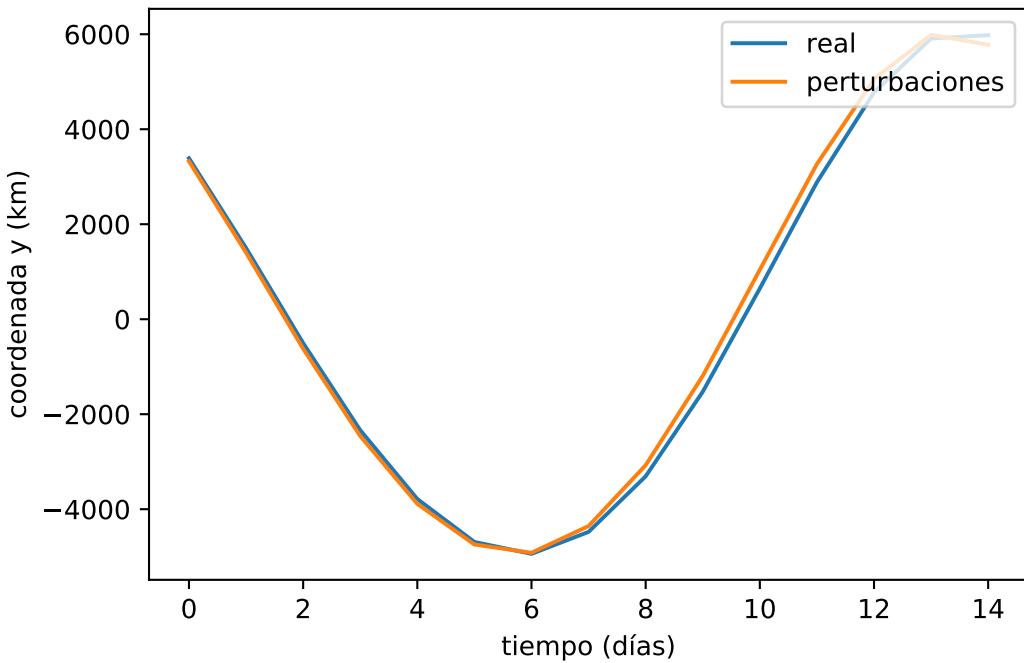
En esta sección se van a presentar también los resultados del modelo simplificado de perturbaciones, para contemplar las mejoras de los modelos de regresión.

Primero, se mostrará la evolución de la posición día a día durante dos semanas:

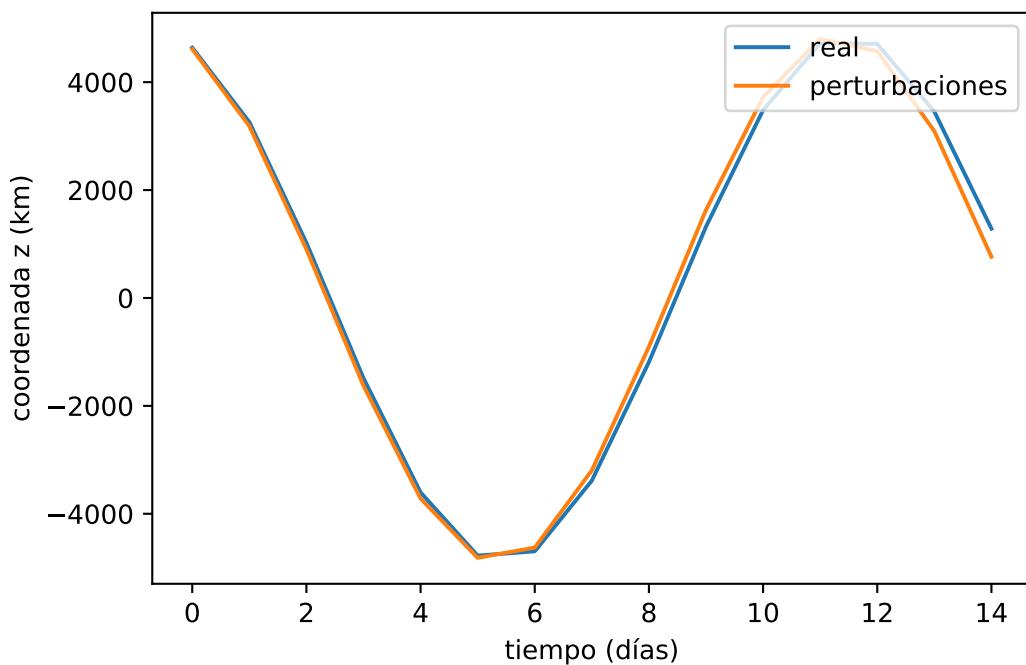
También se incluye una tabla con el error diario durante dos semanas.



**Figura 6.1** Evolución de la órbita a lo largo del tiempo, eje x.



**Figura 6.2** Evolución de la órbita a lo largo del tiempo, eje y.



**Figura 6.3** Evolución de la órbita a lo largo del tiempo, eje z.

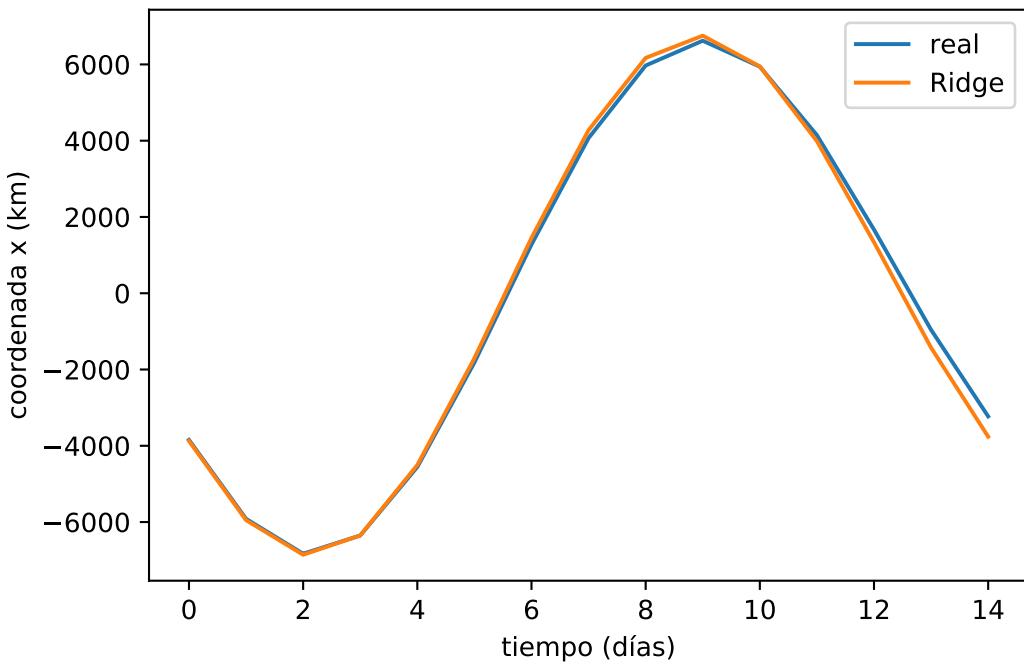
**Tabla 6.1** Error absoluto cada día (SGP8).

Día	RMSSE (km)
1	64.1715
2	80.1896
3	97.5433
4	116.007
5	135.789
6	157.094
7	180.412
8	205.922
9	233.371
10	262.609
11	293.448
12	325.931
13	359.887
14	395.151
15	431.278

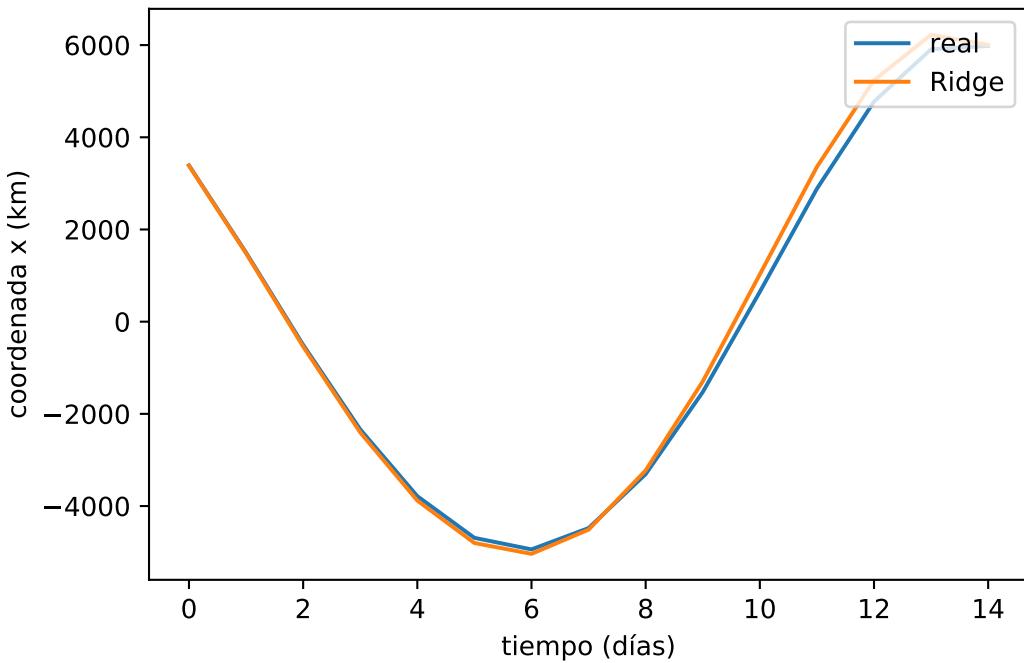
### Regresión Ridge

A continuación, se muestra la evolución de las posiciones diarias durante dos semanas:

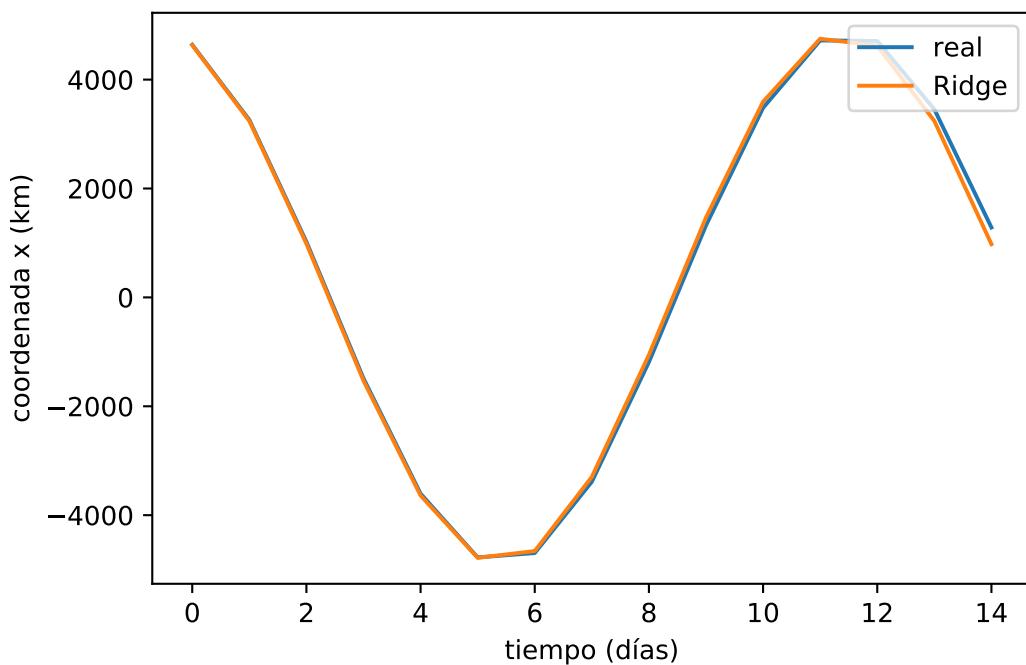
También se muestra la tabla con el error absoluto diario durante dos semanas.



**Figura 6.4** Evolución de la órbita a lo largo del tiempo, eje x.



**Figura 6.5** Evolución de la órbita a lo largo del tiempo, eje y.



**Figura 6.6** Evolución de la órbita a lo largo del tiempo, eje z.

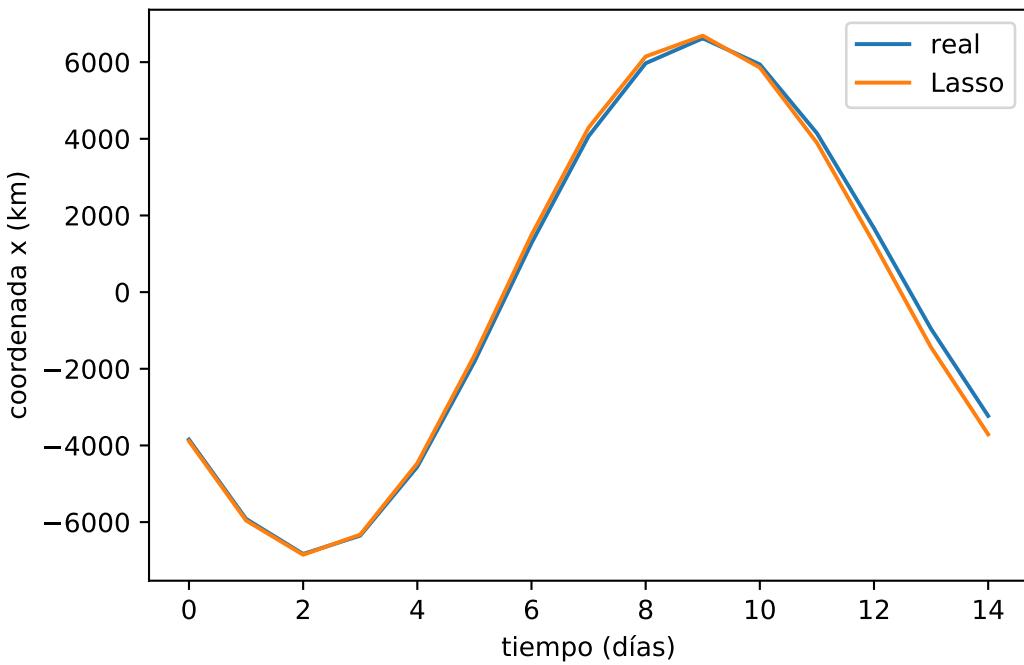
**Tabla 6.2** Error absoluto cada día (Ridge).

Día	RMSSE (km)
1	16.9352
2	25.0142
3	33.3147
4	45.3964
5	65.7511
6	91.8261
7	115.864
8	132.078
9	145.311
10	174.096
11	227.790
12	289.114
13	332.876
14	348.207
15	354.531

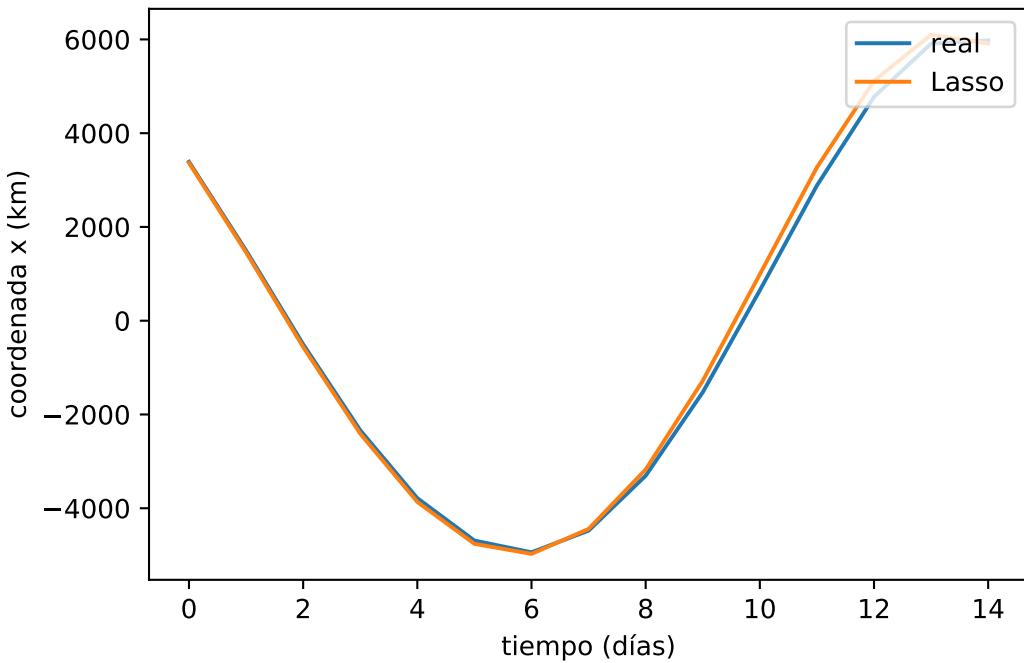
### Regresión Lasso

Se muestra la evolución de la posición diariamente durante dos semanas.

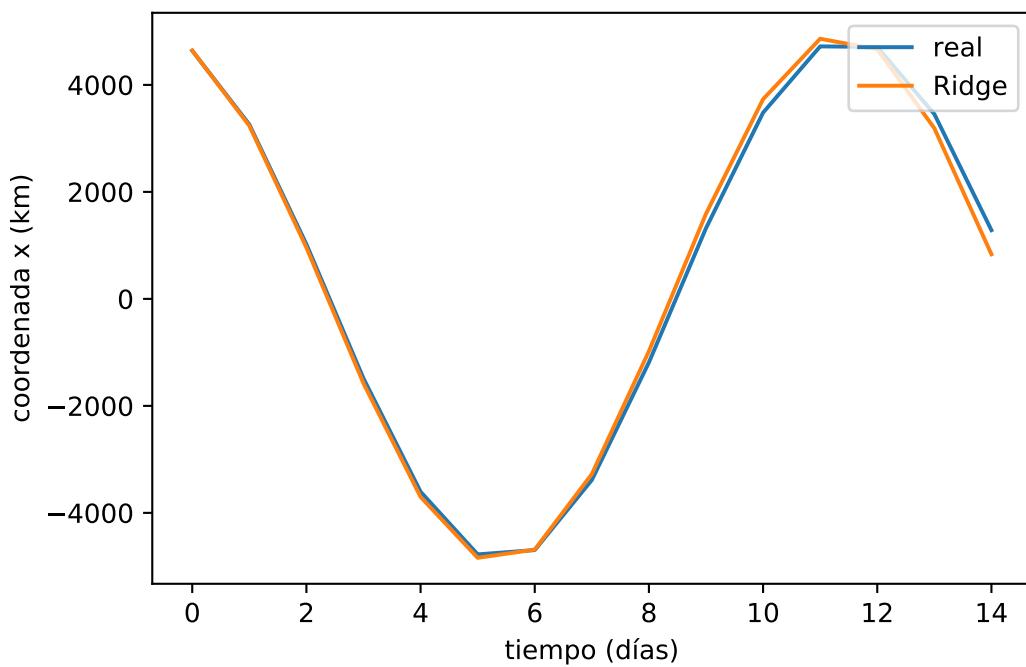
También se muestra la tabla con el error absoluto diario durante dos semanas.



**Figura 6.7** Evolución de la órbita a lo largo del tiempo, eje x.



**Figura 6.8** Evolución de la órbita a lo largo del tiempo, eje y.



**Figura 6.9** Evolución de la órbita a lo largo del tiempo, eje z.

**Tabla 6.3** Error absoluto cada día (Lasso).

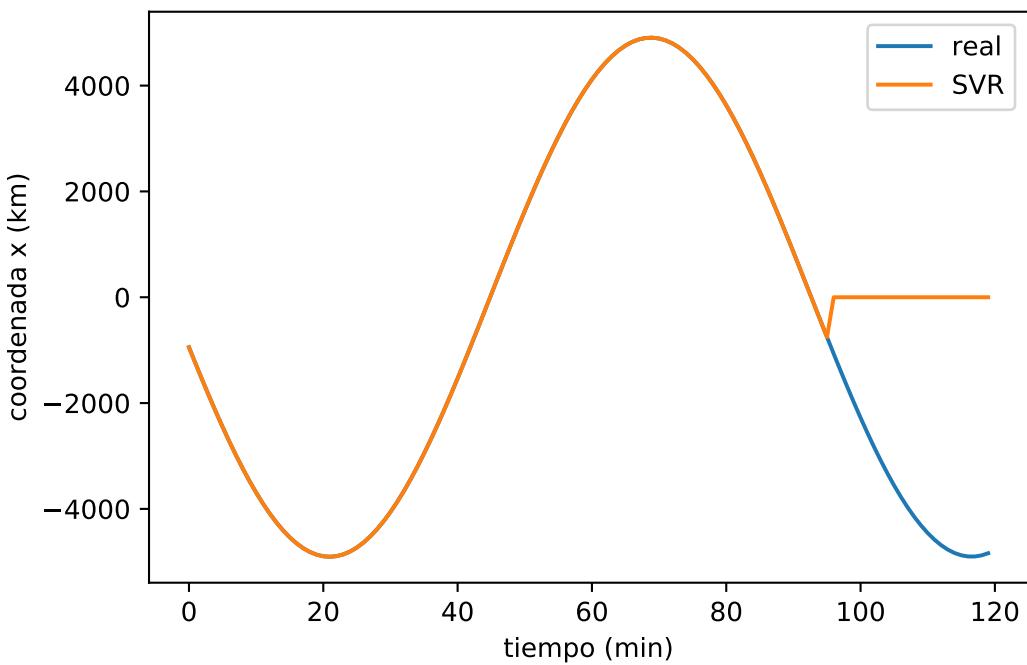
Día	RMSE (km)
1	28.0126
2	36.7609
3	50.3572
4	69.5906
5	90.2431
6	108.058
7	123.982
8	144.772
9	175.859
10	213.869
11	249.845
12	278.428
13	303.397
14	336.006
15	382.987

### 6.1.2 Segunda aproximación

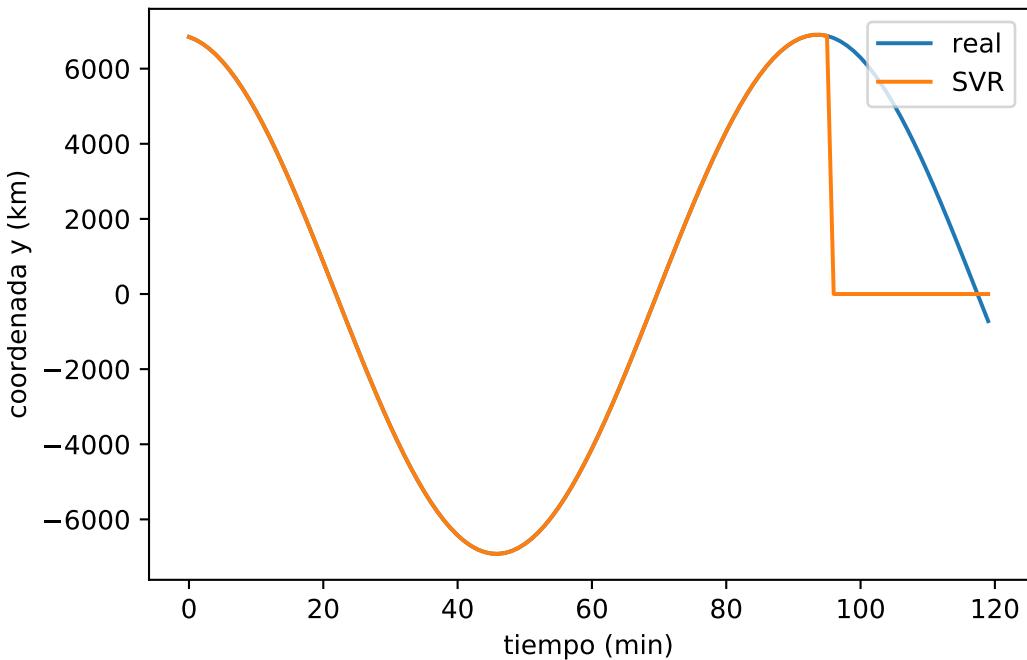
De los siguientes modelos, se expondrán la evolución de las órbitas y los errores que acumulan cada día:

#### SVR

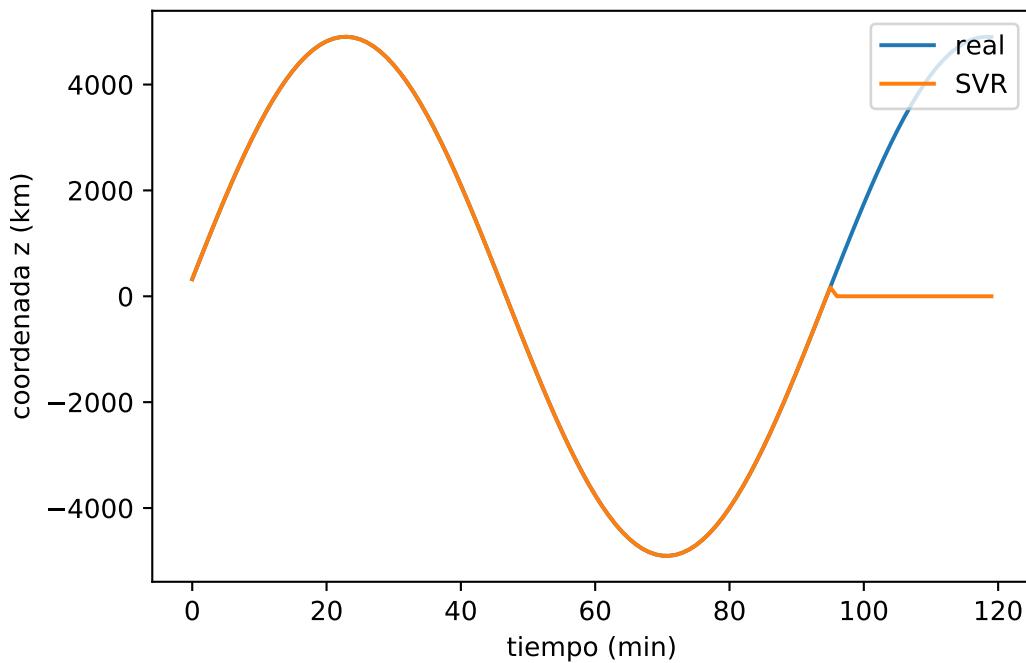
Este método es el más simple de todos, y es capaz de resolver resultados de manera muy rápida:



**Figura 6.10** Evolución de la órbita a lo largo del tiempo, eje x.



**Figura 6.11** Evolución de la órbita a lo largo del tiempo, eje y.



**Figura 6.12** Evolución de la órbita a lo largo del tiempo, eje z.

Se puede observar muy claramente que, después de superar los 96 minutos de ciclo que se necesitan para comenzar a propagar, el modelo no sigue en absoluto la órbita real, por lo que este modelo no tiene ninguna validez.

Esto se debe a que se usa una función kernel genérica, que no tiene gran utilidad en este caso. La solución ideal para mejorar el funcionamiento sería diseñar una función kernel propia que se adapte a la forma elíptica de la órbita.

A pesar de los malos resultados, a continuación se representará el error absoluto que se produce cada día durante cuatro semanas.

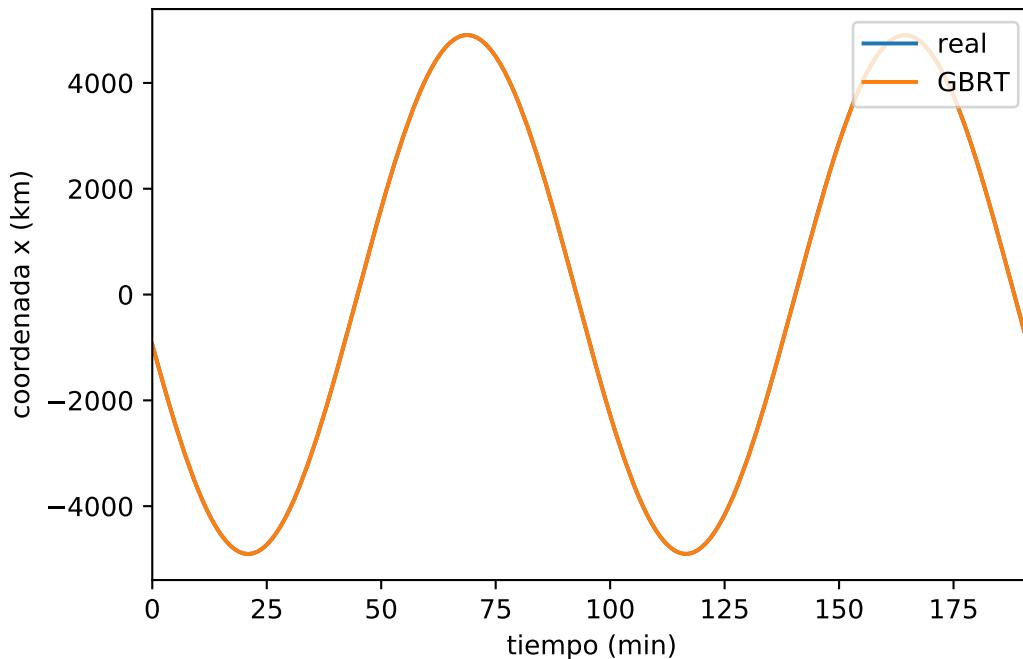
**Tabla 6.4** Error absoluto cada día (SVR).

Día	RMSE (km)	Día	RMSE (km)
1	3993.3444	15	3989.1854
2	3992.6199	16	3990.0348
3	3993.5156	17	3992.5017
4	3995.3224	18	3995.6656
5	3997.7227	19	3998.6247
6	4000.0370	20	4000.9603
7	4001.7325	21	4002.6698
8	4002.6153	22	4003.7156
9	4002.6122	23	4003.7836
10	4001.6265	24	4002.4024
11	3999.5996	25	3999.5082
12	3996.5983	26	3995.7349
13	3993.1502	27	3992.2069
14	3990.3466	28	3990.0392

Se puede observar que los valores diarios de error absoluto son tan altos que el análisis de los resultados de este modelo carezcan de sentido.

### GBRT

El siguiente método es más complejo y tarda varias horas en generarse. Cuanta más profundidad se le da a los árboles y más árboles se generen, mejora sus predicciones, a costa de tardar más tiempo en generarse.

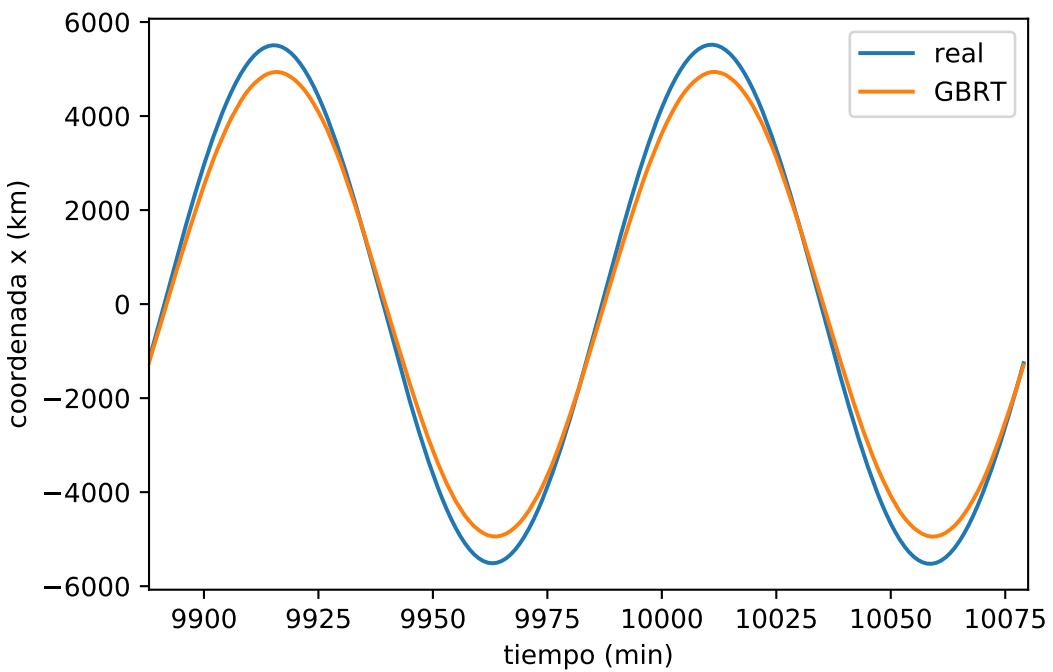


**Figura 6.13** Evolución de la órbita durante las dos primeras rotaciones, eje x.

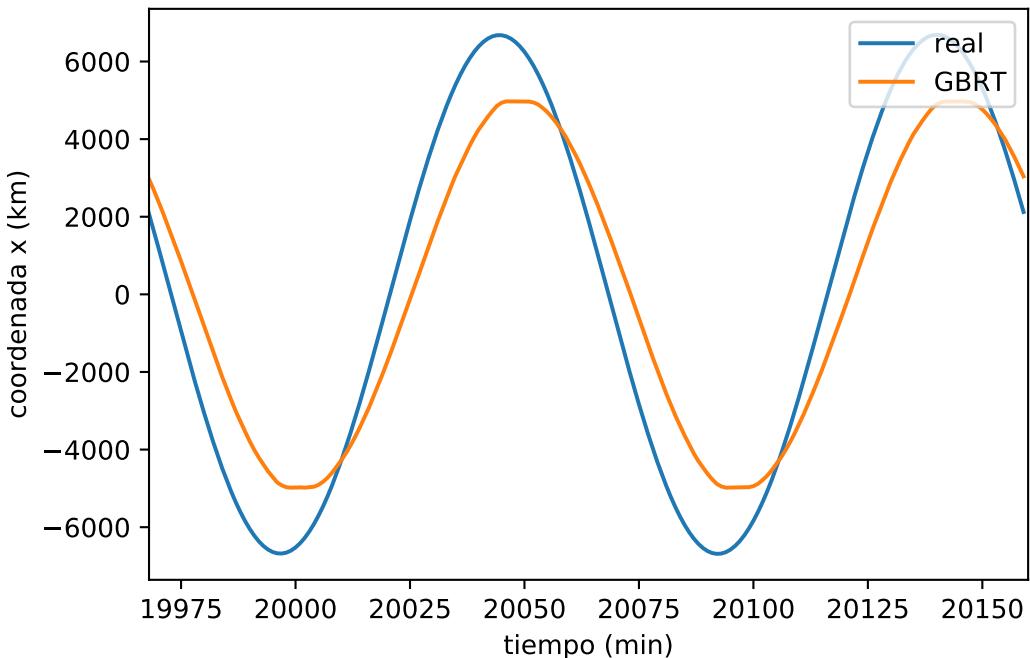
Se puede observar que este modelo funciona bastante mejor, pero aun así va acumulando error poco a poco con el tiempo.

A continuación se presenta la tabla con el error absoluto que se produce cada día durante 4 semanas:

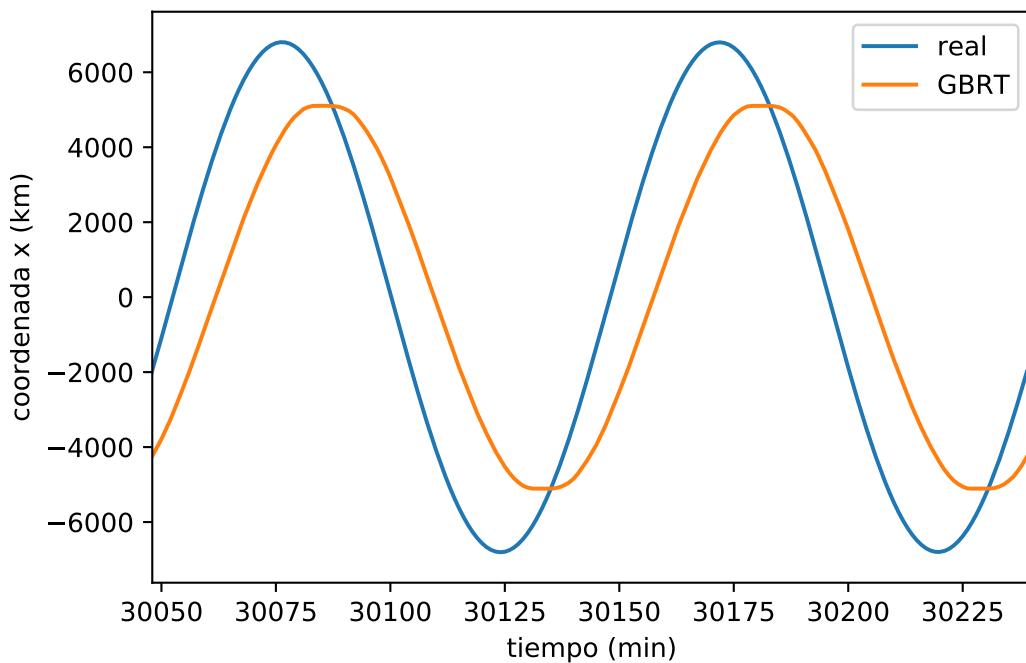
Se observa una degradación del modelo a lo largo del tiempo, funcionando muy bien durante la primera semana.



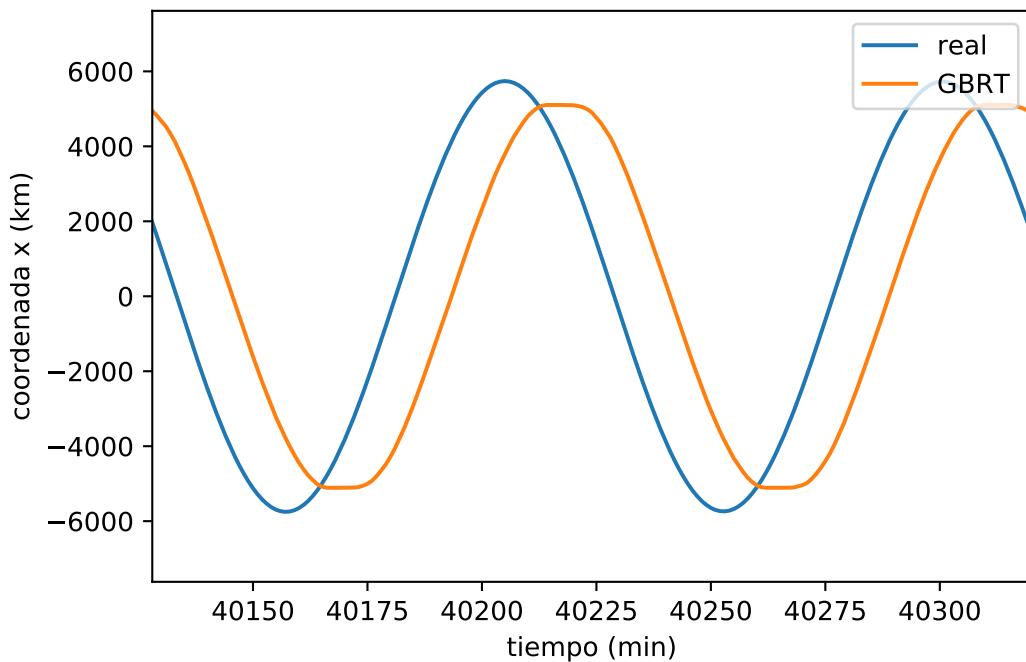
**Figura 6.14** Evolución de la órbita tras una semana, eje x.



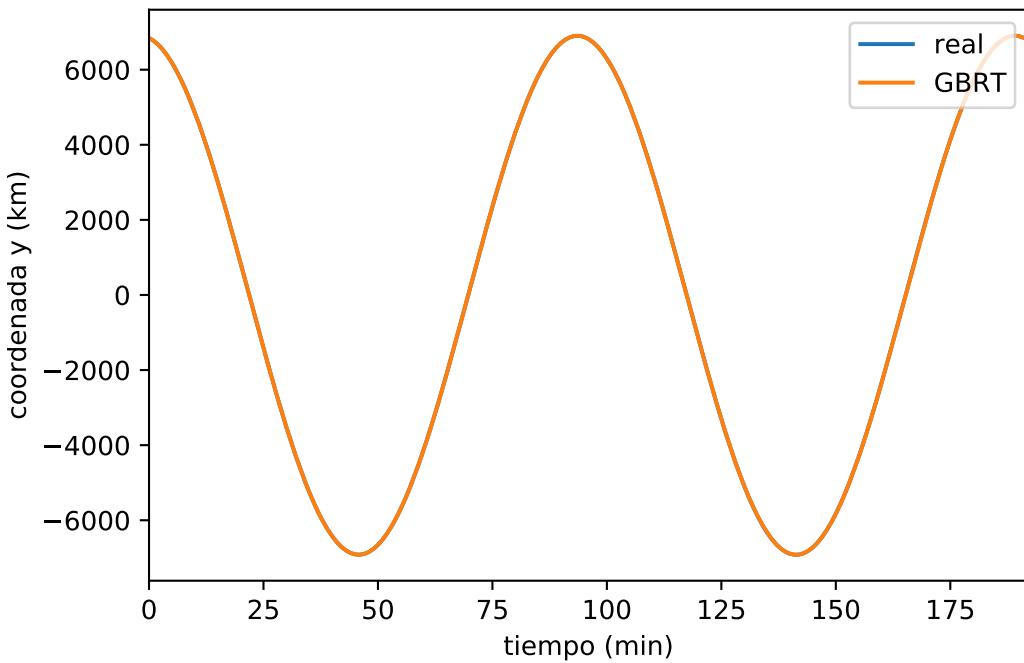
**Figura 6.15** Evolución de la órbita tras dos semanas, eje x.



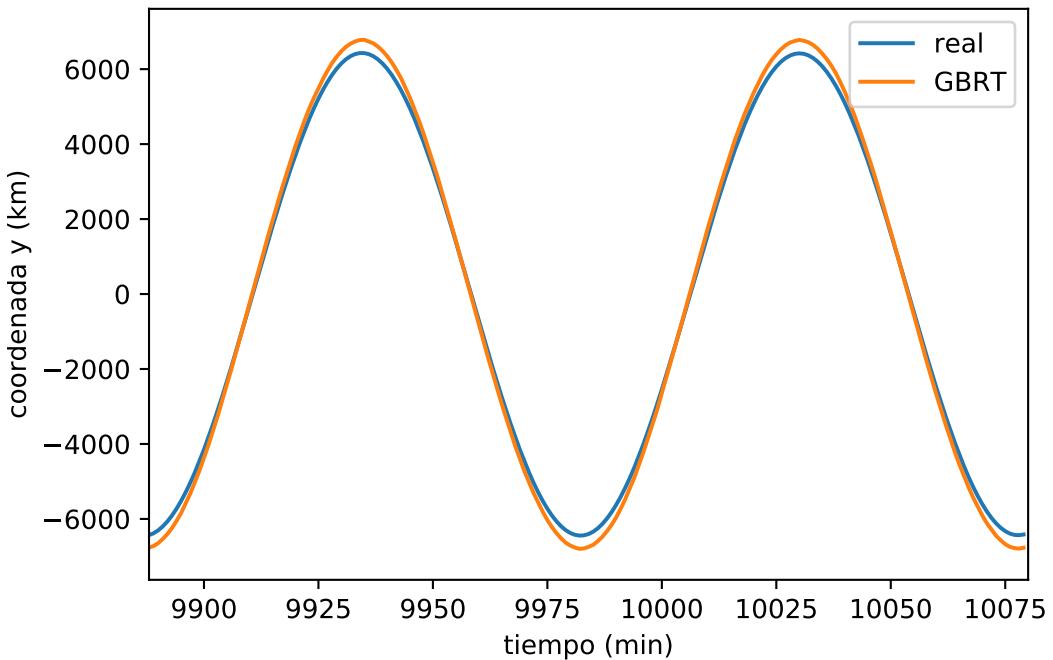
**Figura 6.16** Evolución de la órbita tras tres semanas, eje x.



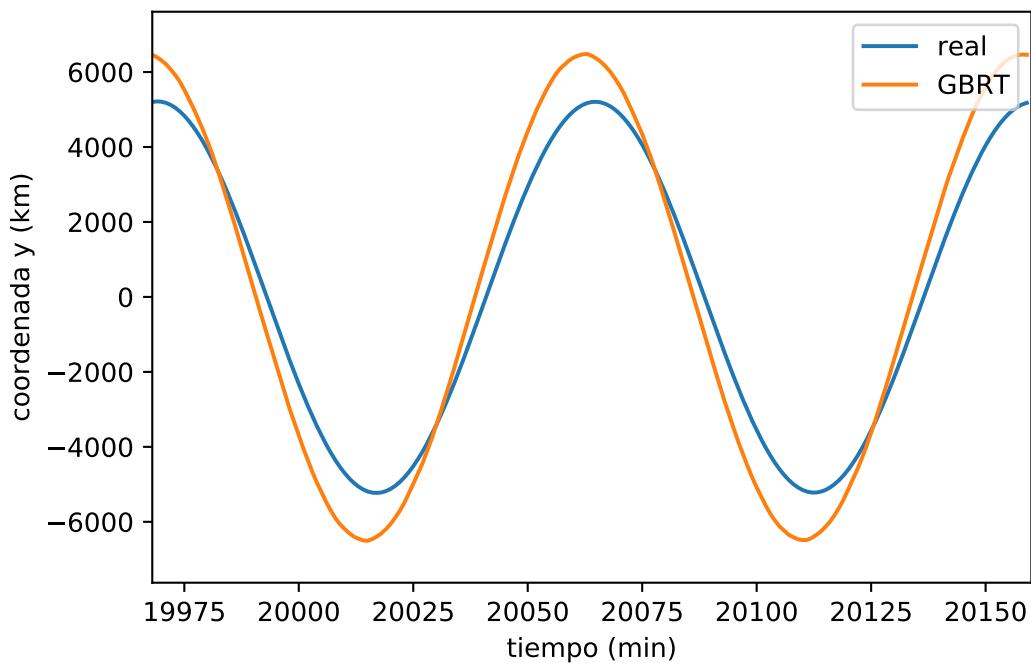
**Figura 6.17** Evolución de la órbita tras cuatro semanas, eje x.



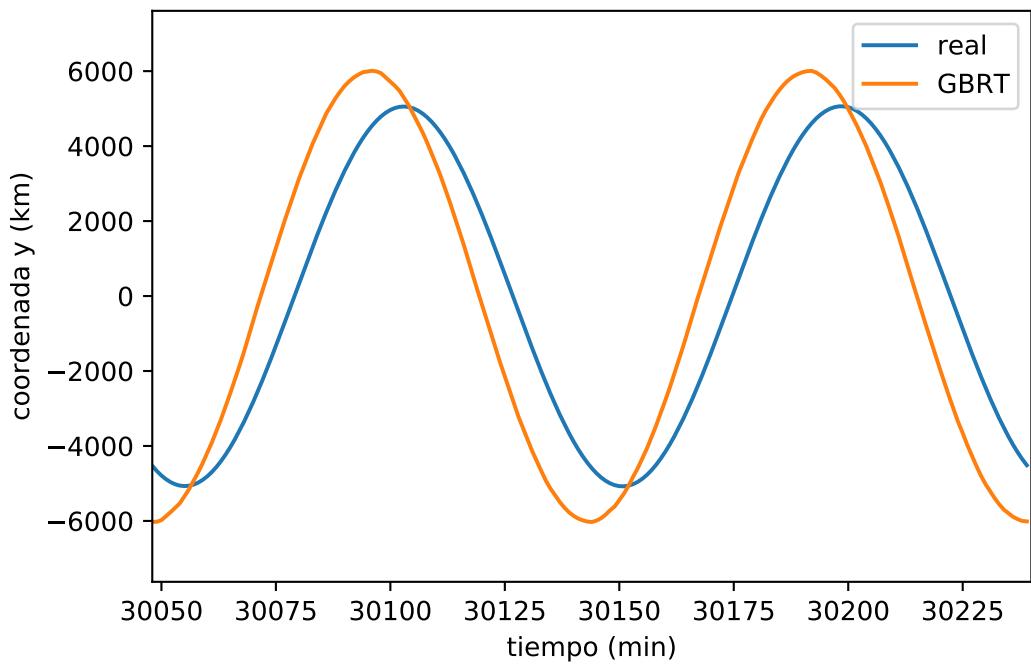
**Figura 6.18** Evolución de la órbita durante las dos primeras rotaciones, eje y.



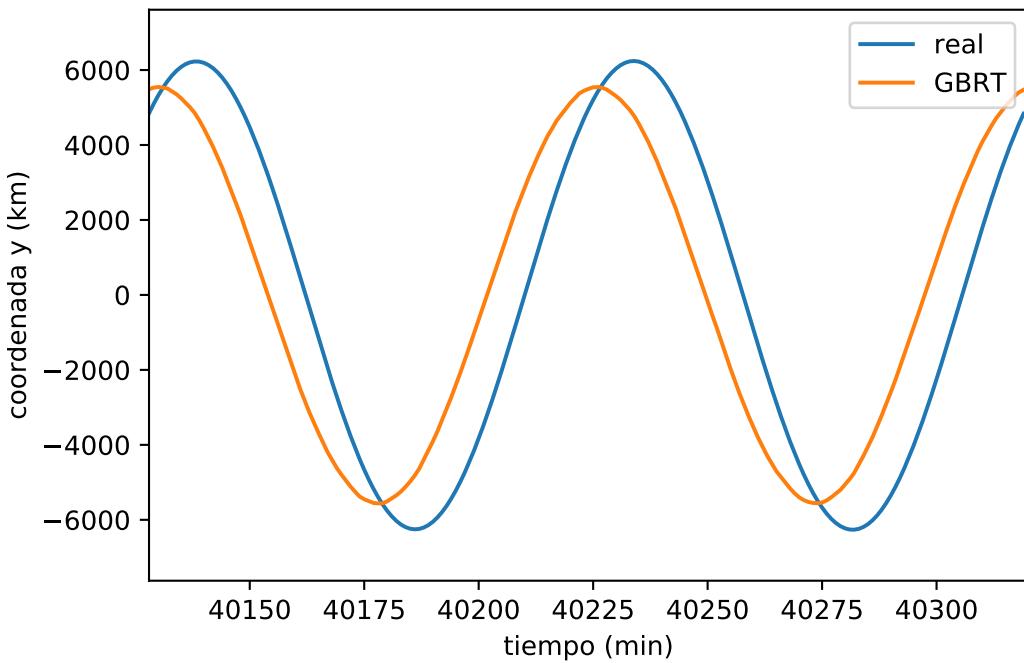
**Figura 6.19** Evolución de la órbita tras una semana, eje y.



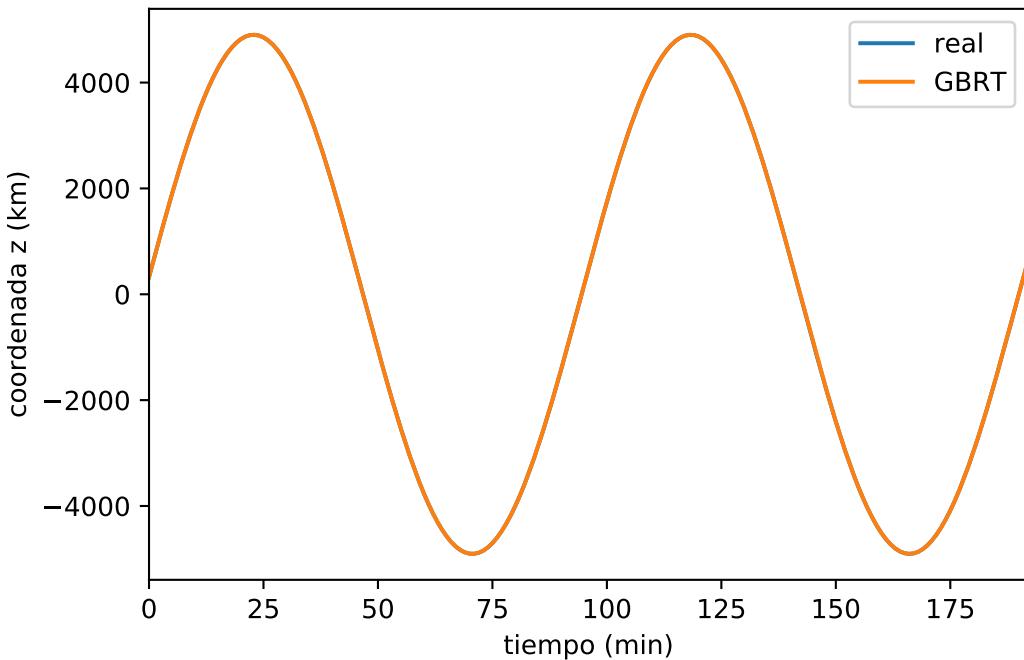
**Figura 6.20** Evolución de la órbita tras dos semanas, eje y.



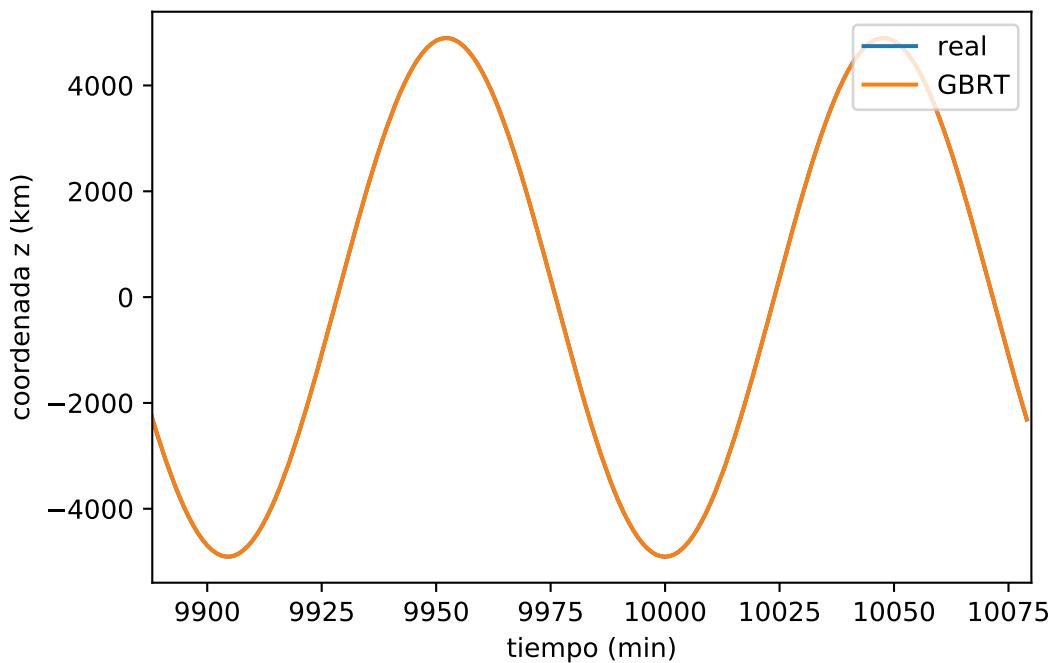
**Figura 6.21** Evolución de la órbita tras tres semanas, eje y.



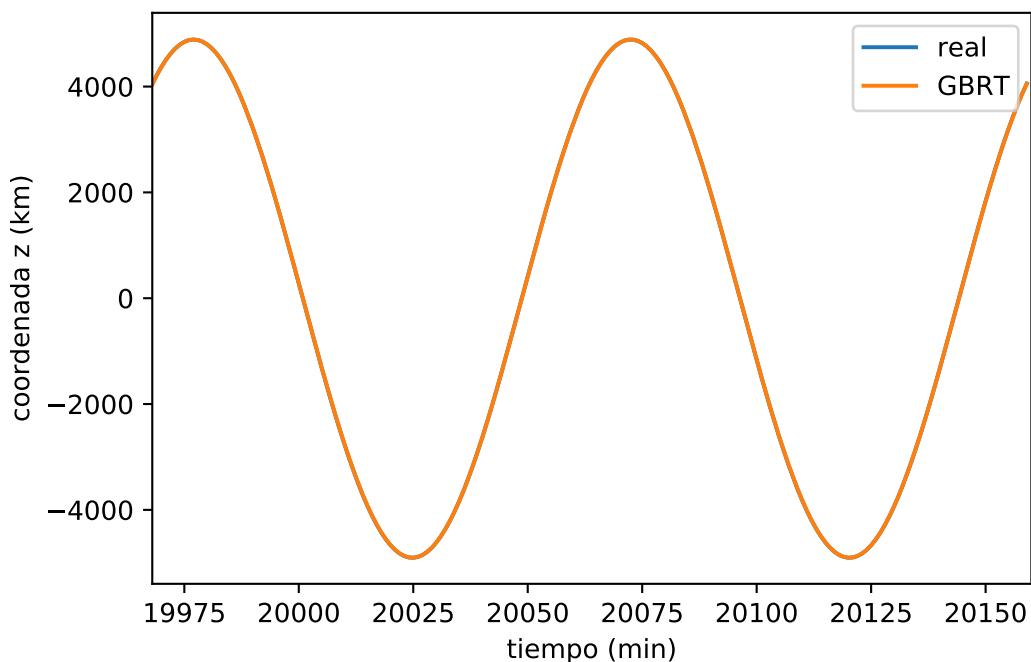
**Figura 6.22** Evolución de la órbita tras cuatro semanas, eje y.



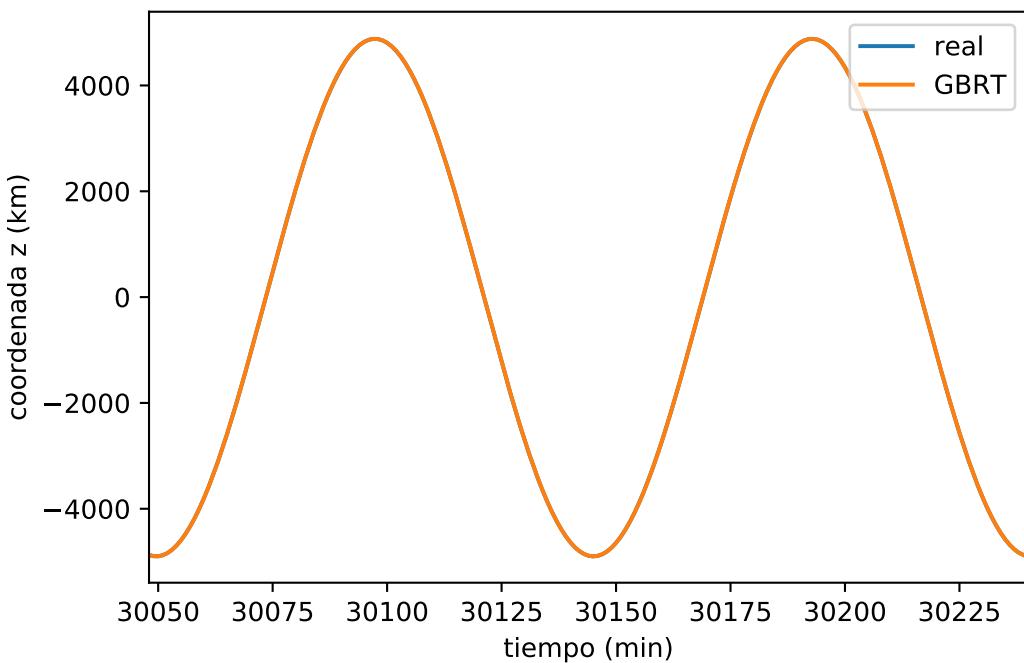
**Figura 6.23** Evolución de la órbita durante las dos primeras rotaciones, eje z.



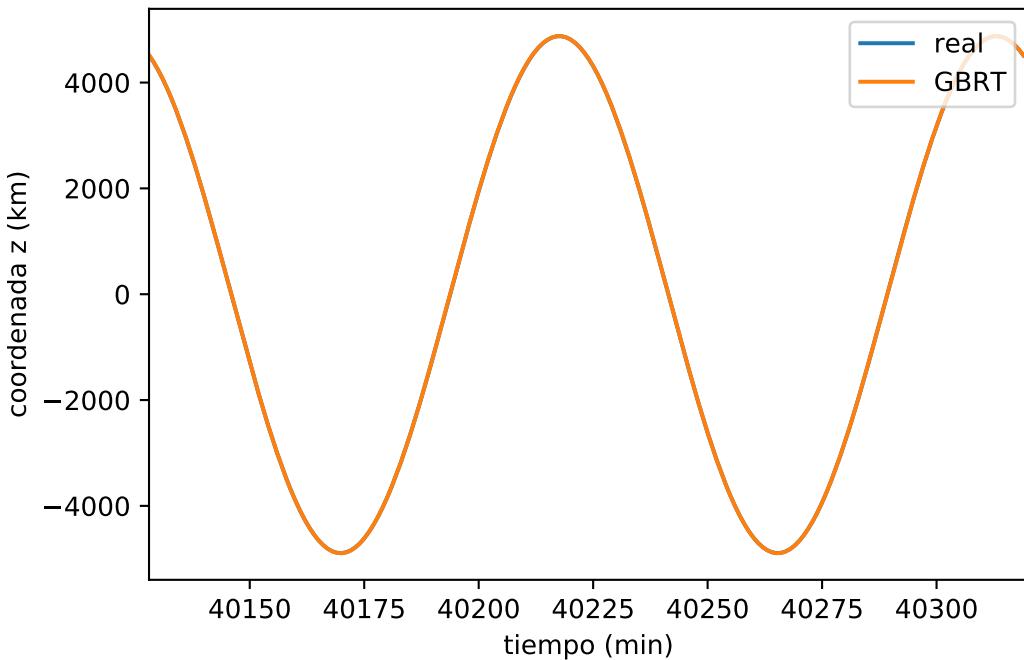
**Figura 6.24** Evolución de la órbita tras una semana, eje z.



**Figura 6.25** Evolución de la órbita tras dos semanas, eje z.



**Figura 6.26** Evolución de la órbita tras tres semanas, eje z.



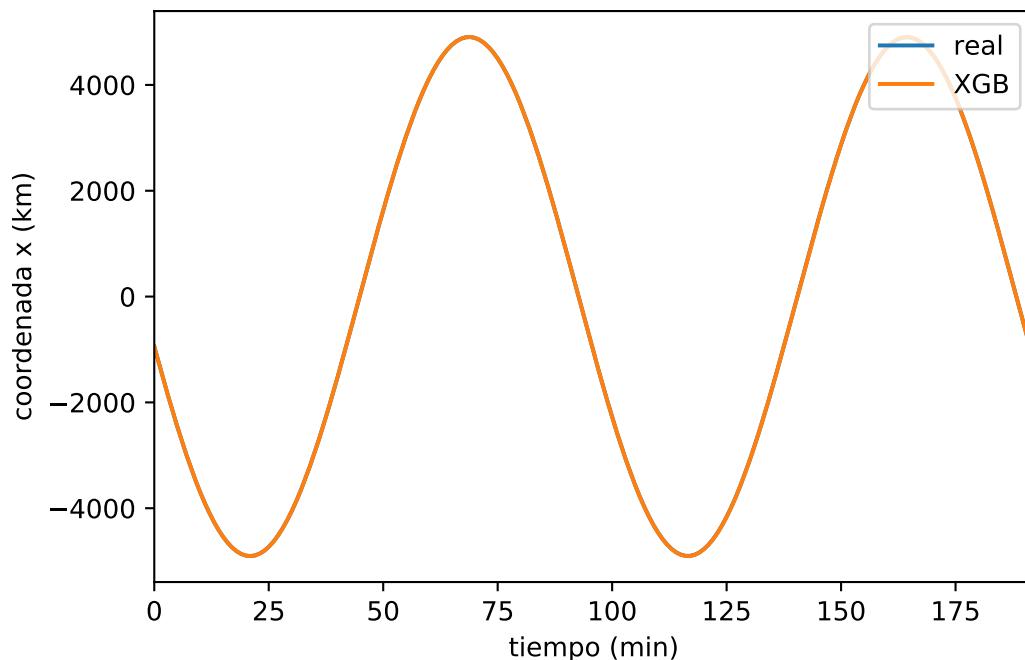
**Figura 6.27** Evolución de la órbita tras cuatro semanas, eje z.

**Tabla 6.5** Error absoluto cada día (GBRT).

Día	RMSE (km)	Día	RMSE (km)
1	0.0026	15	943.7823
2	12.4287	16	1351.4257
3	19.7593	17	1690.6028
4	23.6616	18	1871.3572
5	83.2105	19	1807.4193
6	116.1197	20	1574.1829
7	145.7990	21	1332.2199
8	217.0867	22	1677.7145
9	378.2825	23	2218.6501
10	584.3710	24	2593.3384
11	783.3635	25	2633.8441
12	859.8711	26	2273.4668
13	836.9982	27	1674.2484
14	777.9398	28	1291.5208

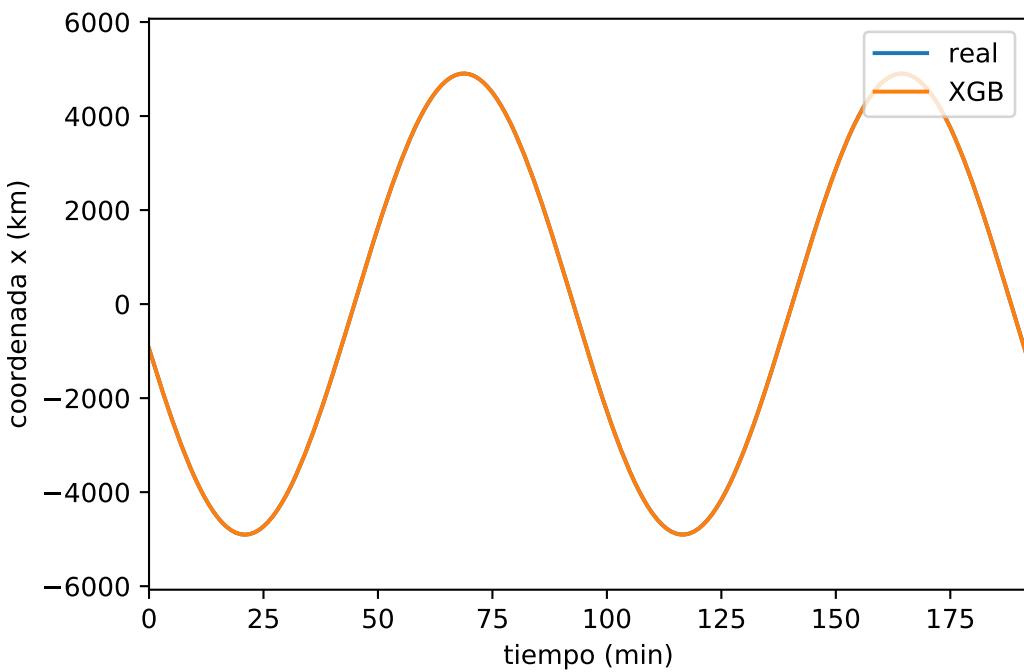
### XGBoost

El siguiente modelo es una versión mejorada del GBRT, que permite obtener modelos con mayor número de estimadores y con mayor profundidad en menor tiempo.

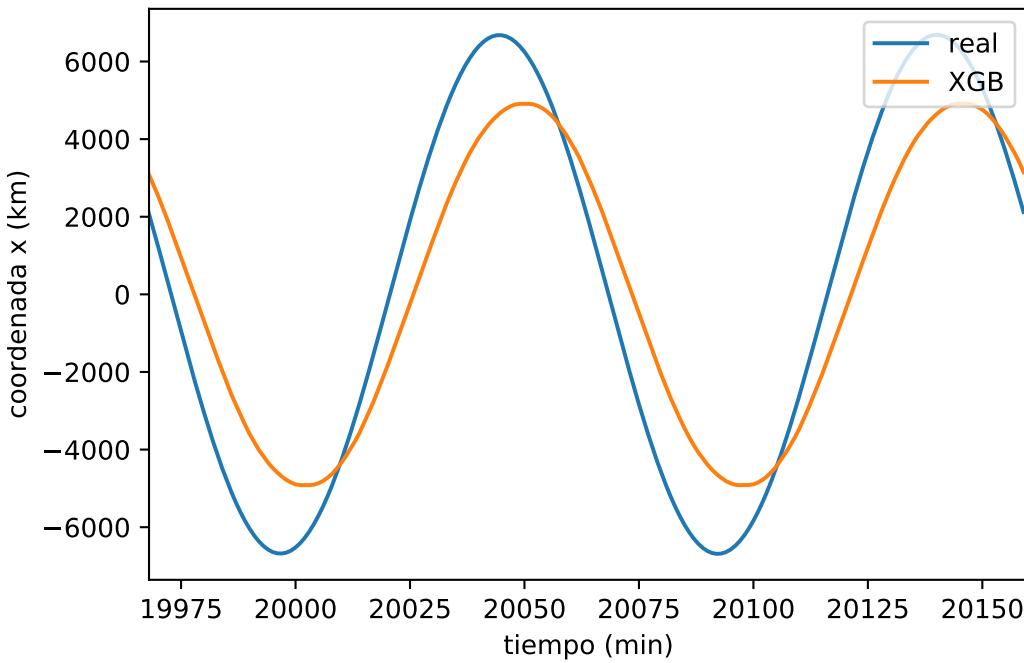
**Figura 6.28** Evolución de la órbita durante las dos primeras rotaciones, eje x.

Se observa una buena predicción la primera semana. Después comienza a empeorar, pero manteniendo resultados aceptables.

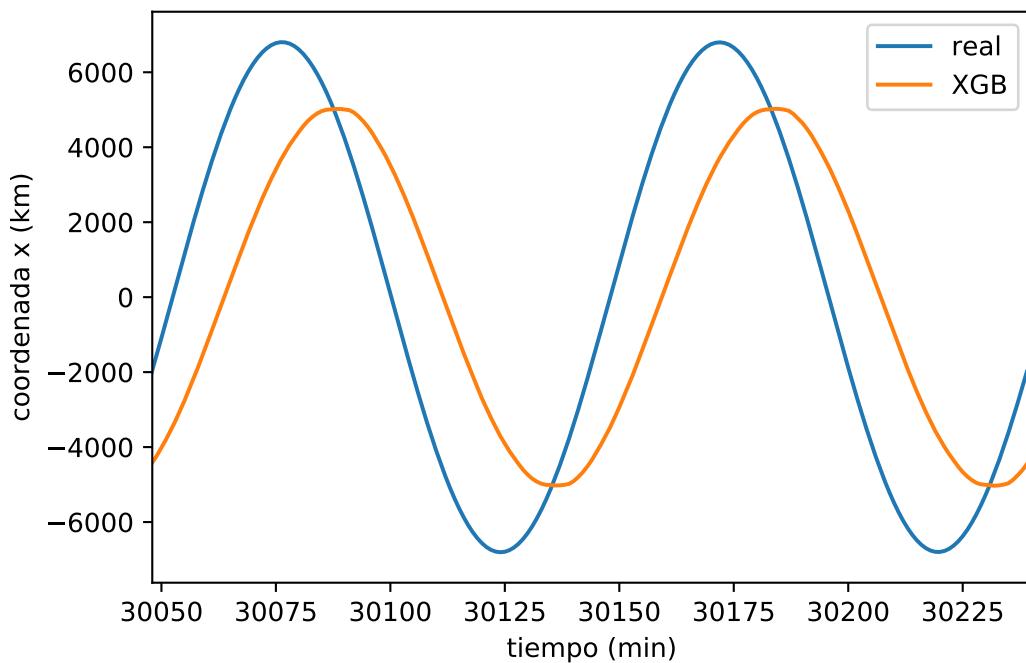
También se presenta la tabla con el error absoluto cada día durante cuatro semanas.



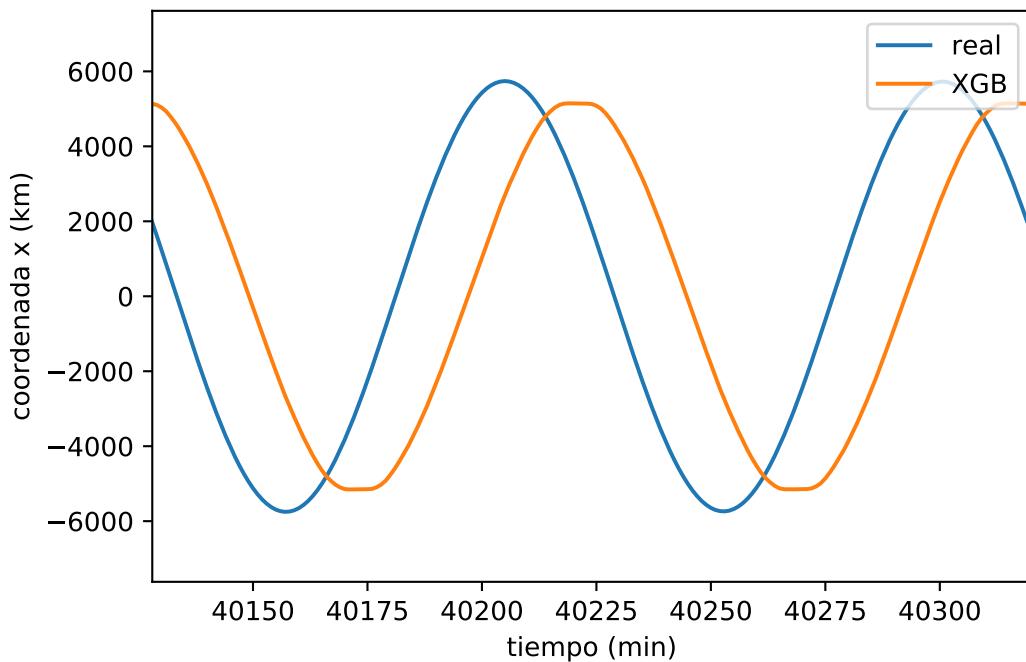
**Figura 6.29** Evolución de la órbita tras una semana, eje x.



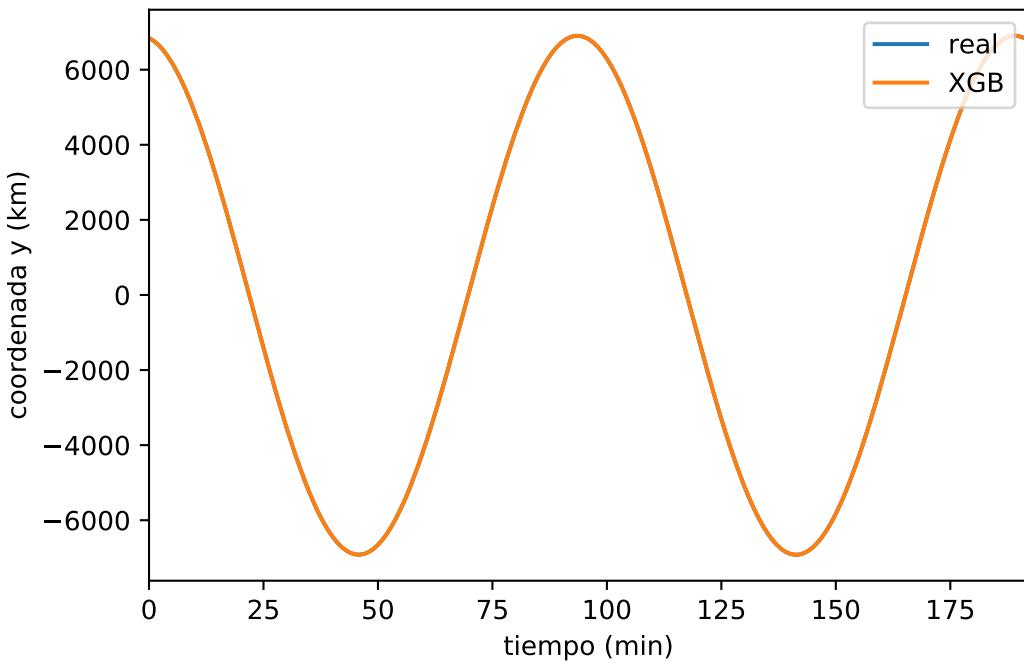
**Figura 6.30** Evolución de la órbita tras dos semanas, eje x.



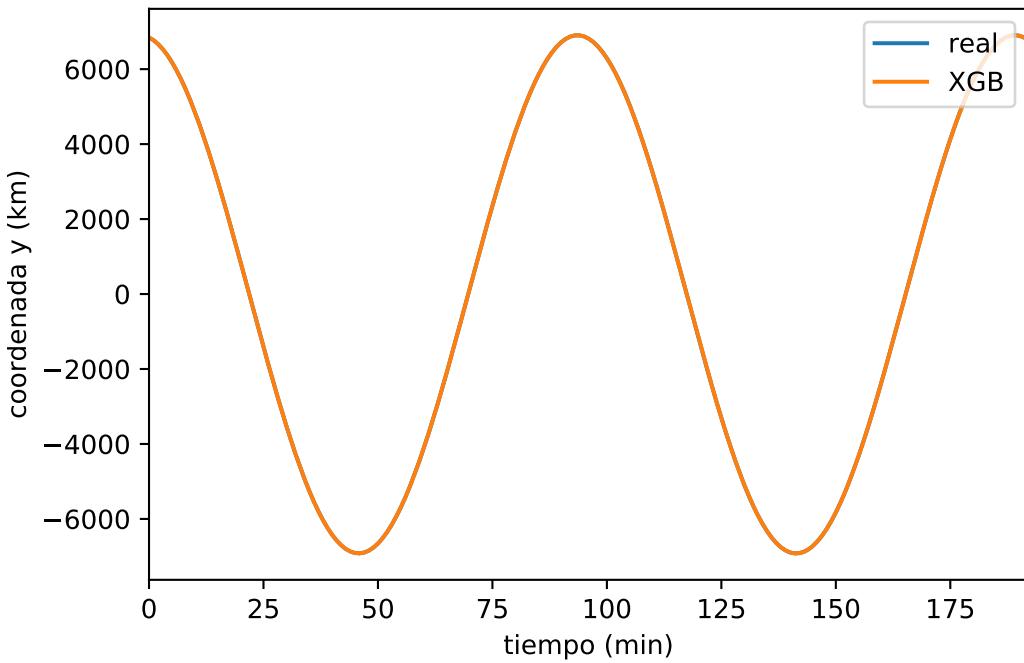
**Figura 6.31** Evolución de la órbita tras tres semanas, eje x.



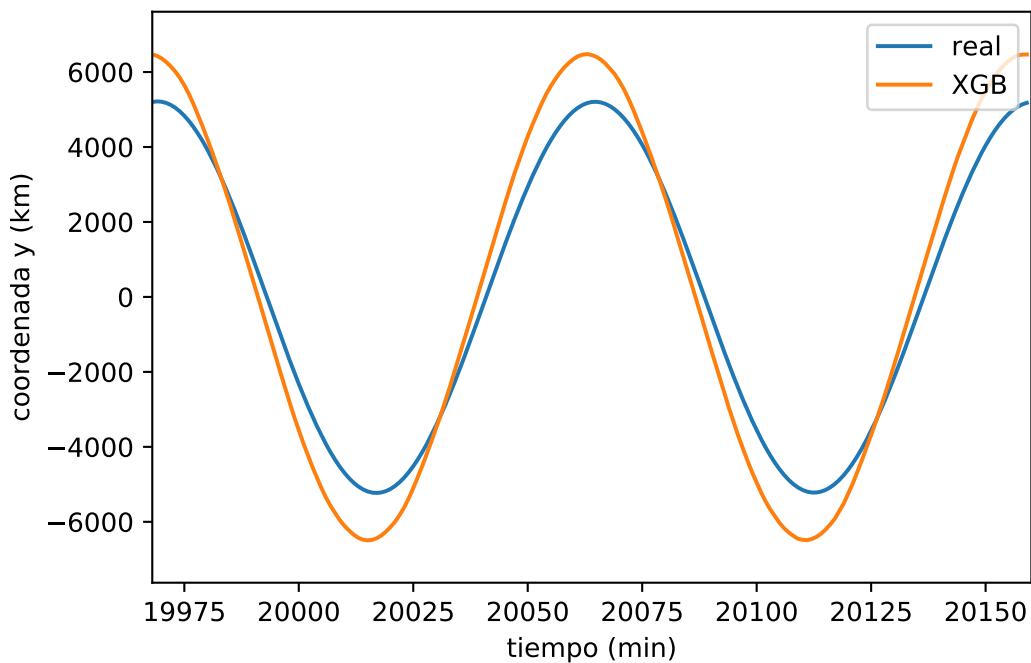
**Figura 6.32** Evolución de la órbita tras cuatro semanas, eje x.



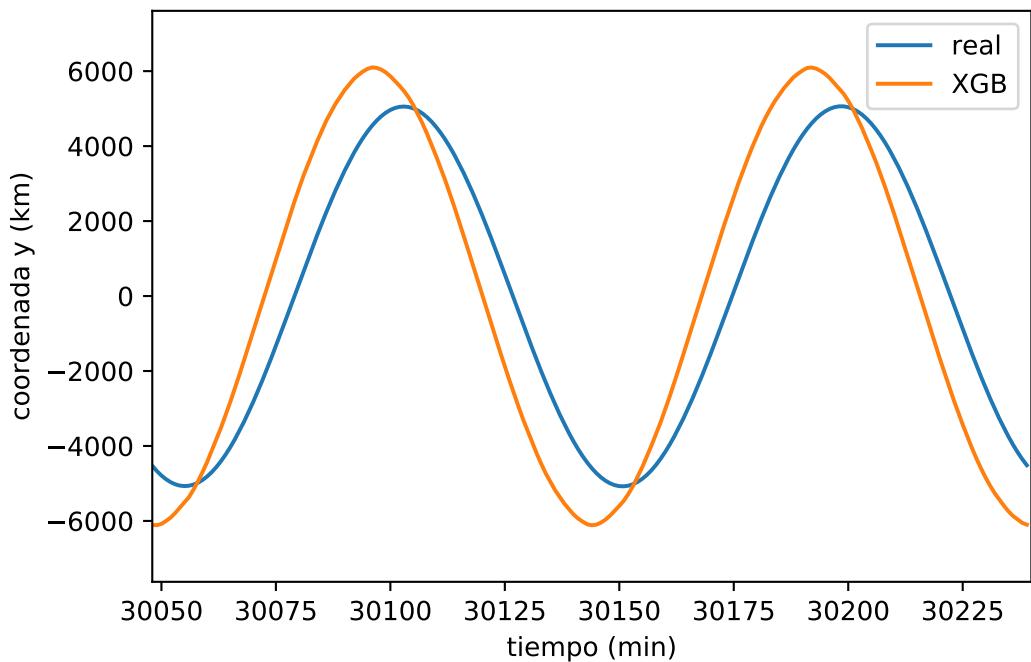
**Figura 6.33** Evolución de la órbita durante las dos primeras rotaciones, eje y.



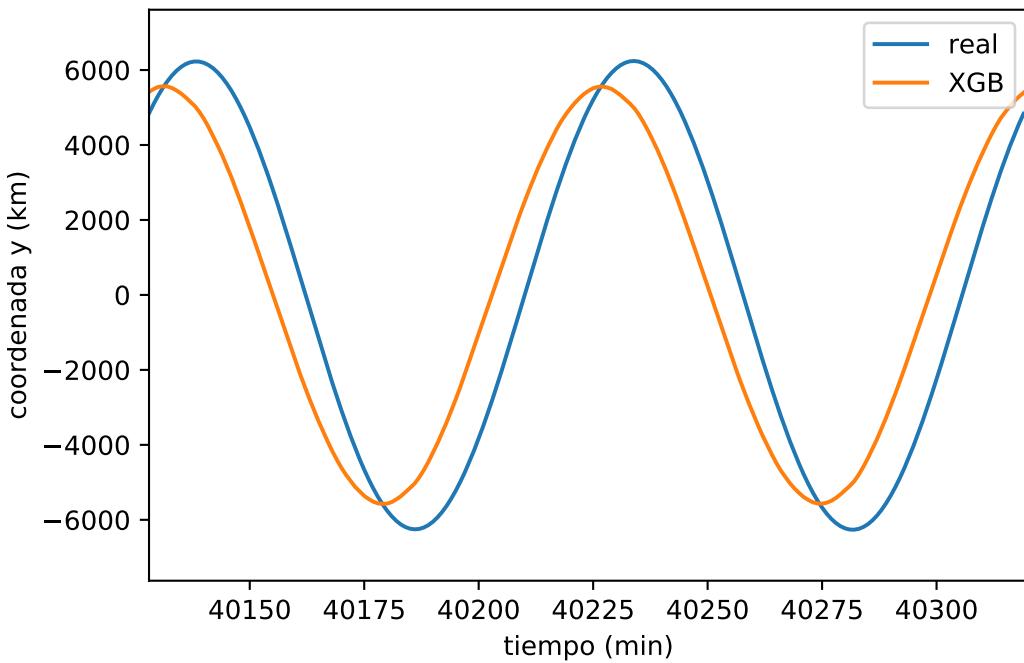
**Figura 6.34** Evolución de la órbita tras una semana, eje y.



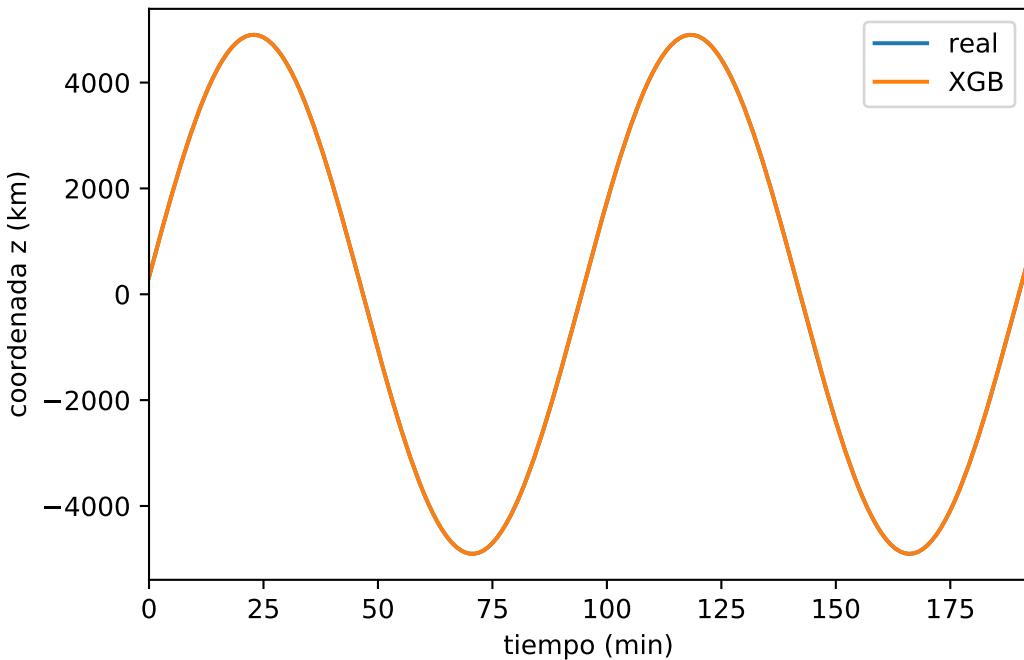
**Figura 6.35** Evolución de la órbita tras dos semanas, eje y.



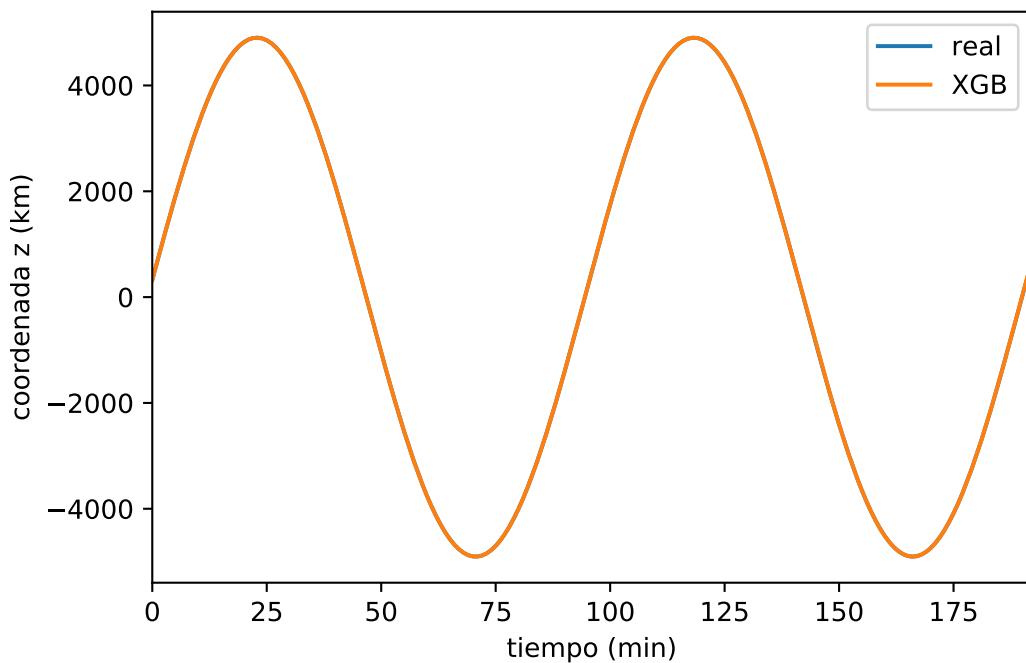
**Figura 6.36** Evolución de la órbita tras tres semanas, eje y.



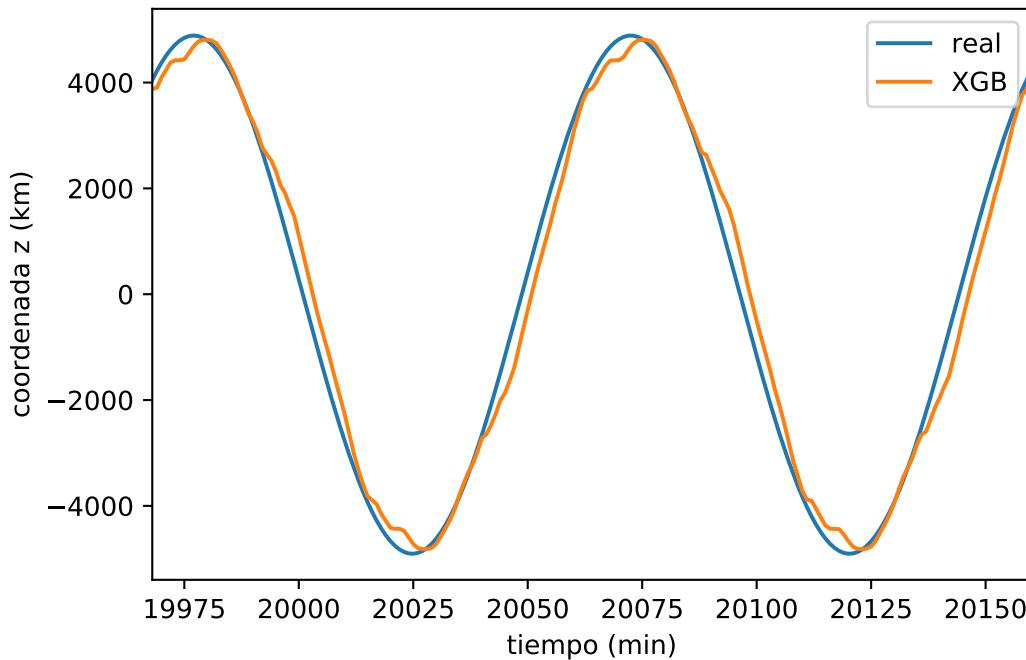
**Figura 6.37** Evolución de la órbita tras cuatro semanas, eje y.



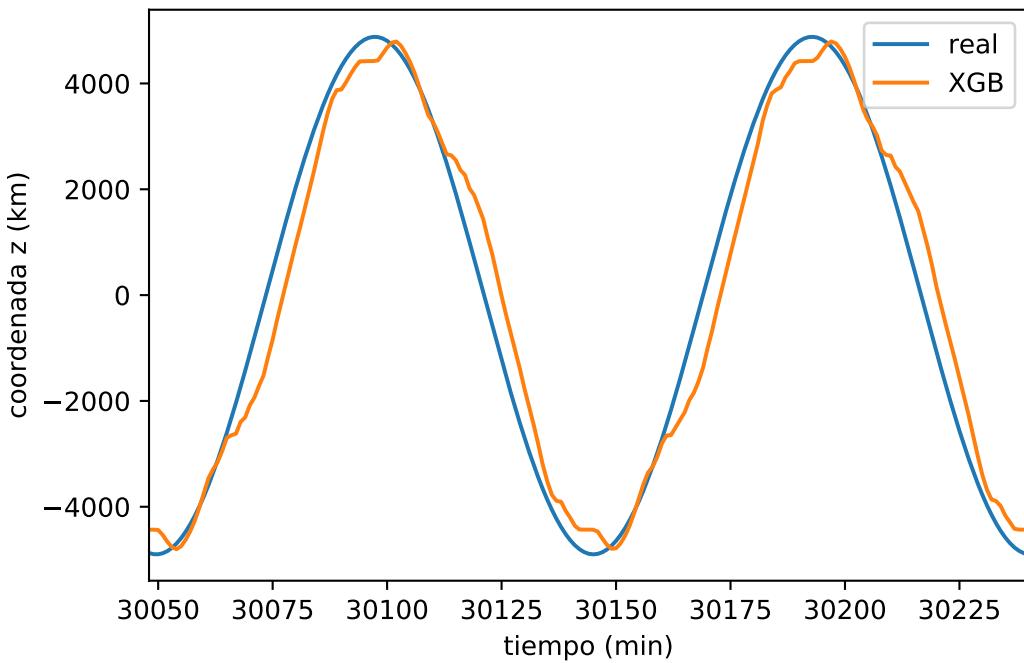
**Figura 6.38** Evolución de la órbita durante las dos primeras rotaciones, eje z.



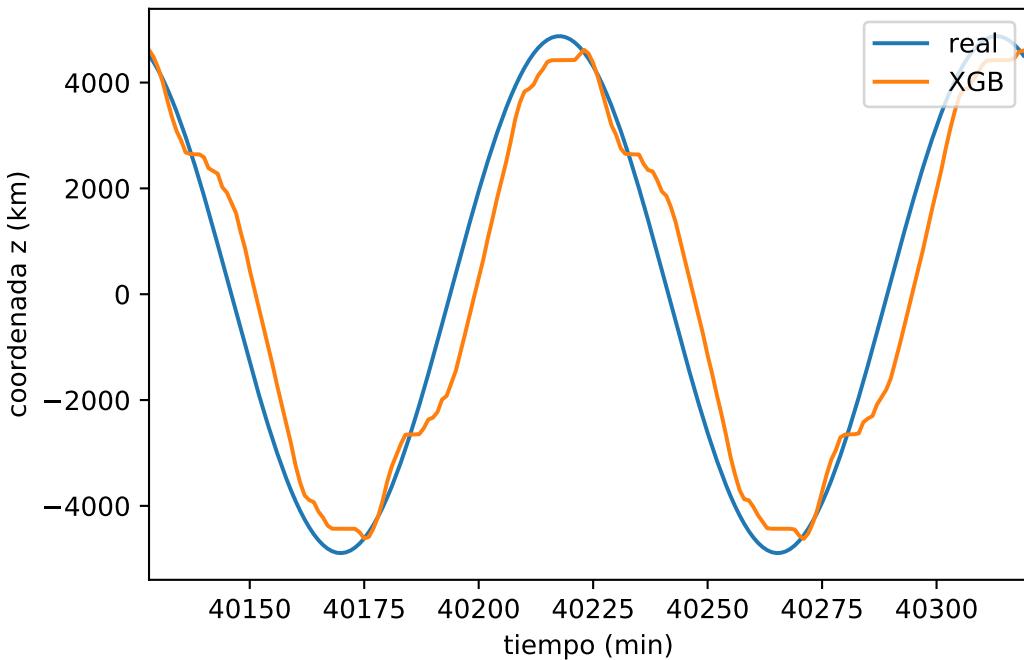
**Figura 6.39** Evolución de la órbita tras una semana, eje z.



**Figura 6.40** Evolución de la órbita tras dos semanas, eje z.



**Figura 6.41** Evolución de la órbita tras tres semanas, eje z.



**Figura 6.42** Evolución de la órbita tras cuatro semanas, eje z.

**Tabla 6.6** Error absoluto cada día (XGBoost).

Día	RMSE (km)	Día	RMSE (km)
1	1.2285	15	1023.0926
2	6.7936	16	1447.3550
3	13.1621	17	1858.0476
4	34.5749	18	2039.0472
5	87.1062	19	2032.1054
6	139.9265	20	1590.8971
7	133.9357	21	1318.1062
8	270.6122	22	1832.2412
9	405.3879	23	2511.6664
10	590.1371	24	2993.0180
11	797.3759	25	3149.6827
12	895.7130	26	2593.8413
13	875.6678	27	1604.0927
14	777.7134	28	1260.7394

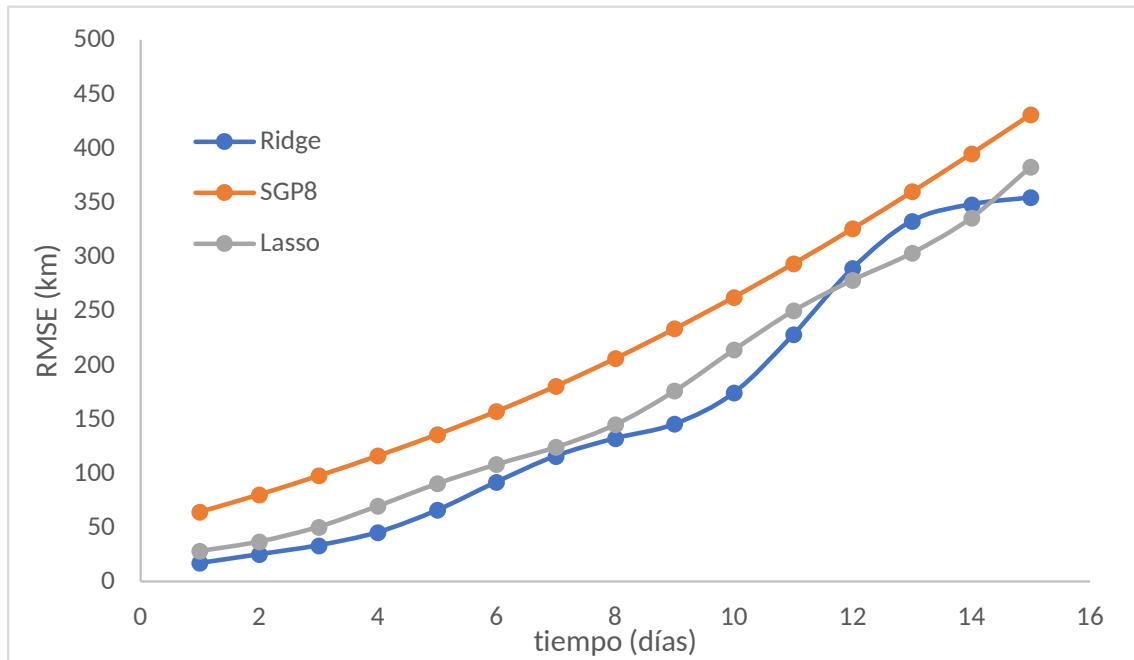
Se pueden observar unos datos de error muy buenos durante la primera semana, pero a lo largo del tiempo van aumentando.

## 6.2 Comparación de resultados

Debido a la diferente finalidad de las dos aproximaciones, estas se analizarán por separado, llegando a unas conclusiones en cada una de ellas. Al final se expondrán todos los resultados conjuntamente para apreciar el funcionamiento global de todos los modelos.

### 6.2.1 Primera aproximación

Se presenta la evolución del error absoluto a lo largo de los días en los 3 modelos, de manera que se puede observar su funcionamiento.



**Figura 6.43** Comparación de los errores absolutos durante dos semanas.

Se puede observar que durante casi todo los días el error del modelo Ridge es ligeramente inferior al modelo Lasso. También se puede observar que mediante el uso de estas técnicas de Machine Learning se puede mejorar ligeramente la predicción a corto plazo realizada por el modelo simplificado de perturbaciones, el que está actualmente en uso.

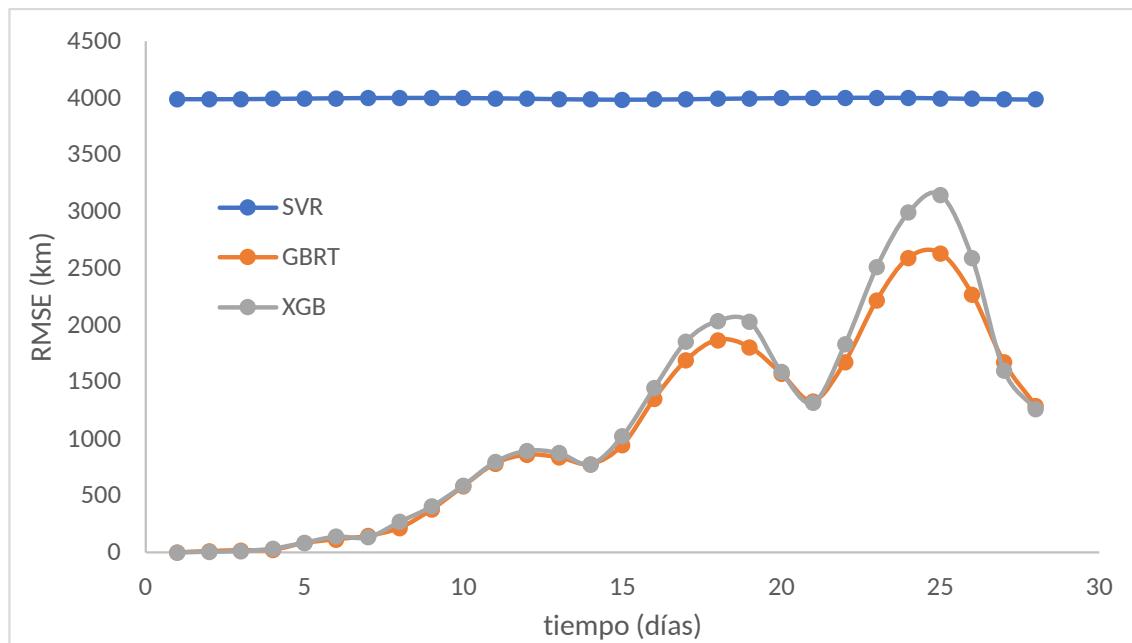
### 6.2.2 Segunda aproximación

Se va a mostrar una gráfica con la evolución de los errores absolutos a lo largo de los días, como rasgo más determinante del funcionamiento del modelo.

En la figura 6.45 se puede observar claramente que el modelo SVR no funciona, mientras que el GBRT y el XGBoost funcionan de forma parecida, aunque ligeramente mejor el GBRT. Es curioso que funcione mejor el modelo GBRT, puesto que tiene menor número de estimadores (1500 frente a 10000) y estos estimadores tienen menor profundidad (15 frente a 25).

Aun así, se puede apreciar que, existe un aumento del error cíclico, con un período de 7 días. Este error se produce debido a la incapacidad del modelo de predecir sucesos con períodos de este tamaño, pues propaga a partir del ciclo que se repite cada 96 minutos, es decir, una revolución alrededor de la Tierra.

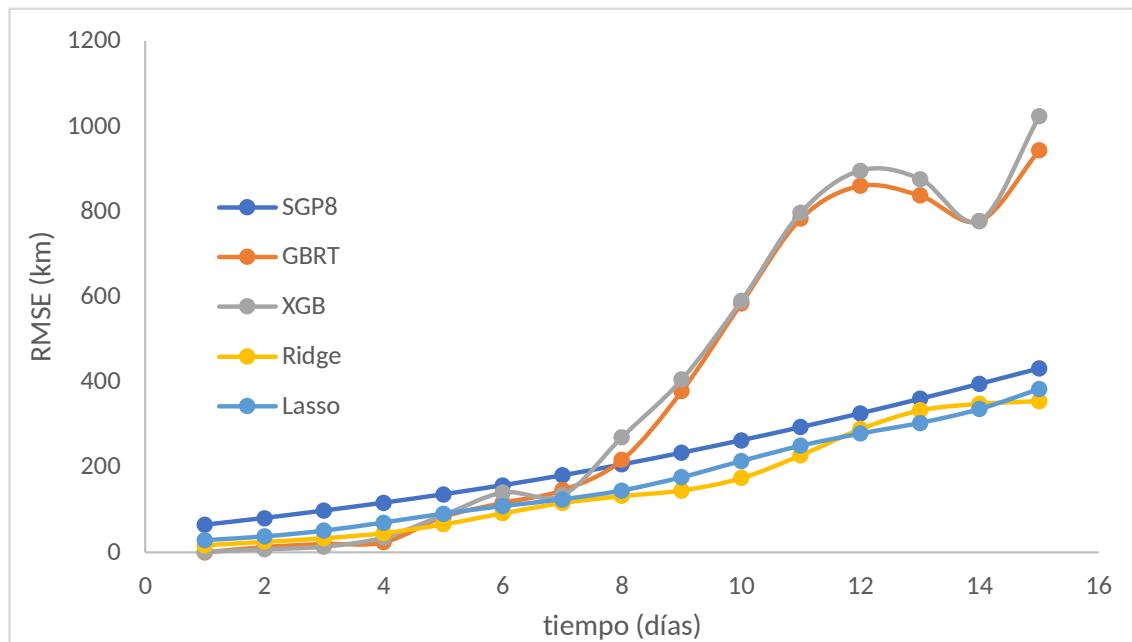
Para mitigar este error se debería propagar a partir de los datos de 7 días. El problema de ello es la capacidad de cálculo que se requeriría para ello.



**Figura 6.44** Comparación de los errores absolutos durante cuatro semanas.

### 6.2.3 Comparación global

A continuación, se presentará la gráfica que recopila los errores de todos los modelos durante los períodos empleados. No se va a representar el modelo de SVR, puesto que debido a su mal funcionamiento solo empeoraría la calidad de visualización del resto de modelos.



**Figura 6.45** Comparación de los errores absolutos de todos los modelos.

Se puede observar que durante los primeros días el error producido por los modelos de propagación (segunda aproximación son inferiores). Después de una semana de funcionamiento los errores se equiparan, y a partir de ahí crecen en mayor medida debido al error cíclico presente en la segunda aproximación.

Se puede observar que los modelos específicos empleados para el cálculo de la órbita del CubeSat (primera aproximación generan menos error en el medio plazo, aunque son menos robustos y versátiles, y tienen un rango de uso muy inferior.

Por tanto, de esta comparación se podría deducir que para predicciones de hasta una semana funciona mejor la segunda aproximación, y la segunda semana la primera aproximación. A partir de ahí, solo se podría utilizar la segunda aproximación.

### 6.3 Análisis de resultados

Se puede observar, que las dos aproximaciones abordan el problema de manera totalmente diferenciada. En la primera aproximación, se busca un aprendizaje de los datos correspondientes a un único satélite con el objeto de mejorar en la medida de lo posible las predicciones de un modelo ya existente. En este caso se busca un funcionamiento rápido del algoritmo, que se pueda acoplar al modelo de perturbaciones y que a tiempo real sea capaz de mejorar los resultados que ofrece en este momento.

En cambio, en la segunda aproximación se buscan modelos más versátiles que puedan funcionar en gran cantidad de objetos residentes en el espacio, no solo para el caso estudiado. Además, tiene la ventaja de que, conociendo solo la información de una revolución entorno a la tierra, es capaz de propagar la posición durante tiempo indefinido (teniendo en cuenta que el error crecerá).

Por lo tanto, el uso recomendado para cada aproximación es totalmente distinta.

Se puede tomar como un éxito los resultados obtenidos en la primera aproximación, puesto que con solo almacenar datos de posición durante dos semanas y aplicando modelos muy sencillos, se pueden obtener las posiciones futuras de los satélites de manera más precisa que el modelo vigente, el SGP8.

En lo referente a la segunda aproximación, se puede concluir que lo idóneo sería tomar como tiempo a propagar 7 días, para mitigar el error con ese período. Además, podemos observar que el método GBRT, pese a ser más lento, es más robusto y puede ofrecer mejores resultados con hiperparámetros teóricamente de peor calidad. Se puede concluir que, con una mayor capacidad de computación, se podrían mejorar tanto los hiperparámetros como el período de propagación, de manera que se pueda utilizar el Machine learning para la predicción de órbitas a medio plazo.

Aun así, para predicciones a corto plazo hasta una semana vista, estos modelos de propagación ofrecen muy buenos resultados incluso propagando solo una revolución.

# 7 Conclusiones

---

La realización de este estudio permite atisbar una pequeña muestra del inmenso abanico de posibilidades que puede ofrecer el Machine Learning. Con el rápido avance en materia de computación, cada vez se podrán realizar mejores estimaciones de todo nuestro entorno en menor tiempo.

Se puede observar como, aplicando algoritmos de gran sencillez en el entorno adecuado se pueden mejorar las herramientas actuales en uso, como se ha realizado en la segunda aproximación, que mejora los datos del modelo empleado en la actualidad, el SGP8.

Además, utilizando técnicas de propagación (mayor grado de complejidad), con información de una sola revolución, se pueden obtener estimaciones mucho más precisas que las vigentes.

También se puede comprobar como, a medida que el marco temporal aumenta y se trata de generalizar los modelos, las técnicas se vuelven más complejas y pueden requerir grandes recursos computacionales, como ocurre en la segunda aproximación.

A partir de los resultados obtenidos del experimento, se concluiría con que lo óptimo para predecir una órbita sería: el modelo GBRT propagando una revolución, durante la primera semana; para la segunda semana funcionaría mejor el modelo Ridge en apoyo al modelo SGP8; y a partir de la tercera semana se debería utilizar un modelo de propagación con un período de 7 días, para recoger con exactitud fenómenos con períodos mayores.

Todo esto, se podría sustituir solamente por un método de propagación con un gran período, pero debido a los recursos que necesita, solo se considera necesario para el largo plazo.

Aparte del uso del Machine Learning, una buena medida para afrontar el problema de predicción de órbitas a largo plazo (períodos mayores a dos semanas) puede ser el Deep Learning, que, aunque requiere grandes recursos computacionales, podría ser el siguiente paso en esta investigación. Las técnicas de Deep Learning usan redes neuronales para el aprendizaje de los modelos, creando inteligencias artificiales que funcionan de manera equivalente al cerebro humano.



# Apéndice A

## Código de MATLAB

En este apéndice se incluirán los principales códigos utilizados en este estudio en el lenguaje MATLAB

### A.1 SGP4

Este código no ha sido utilizado, pero se ha hablado de este modelo de perturbaciones en el apartado 2.1 [4].

**Código A.1** Código del modelo de perturbaciones simplificado SGP4.

```
%-----  
%----- sgp4 -----  
%-----  
function [pos, vel] = sgp4(tsince, satdata)  
ae = 1.0;  
tothrd = (2.0/3.0);  
XJ3 = -2.53881e-6;  
e6a = 1.0E-6;  
xkmper = 6378.135;  
ge = 398600.8; % Constante gravitacional de la Tierra  
CK2 = (1.0826158e-3 / 2.0);  
CK4 = (-3.0 * -1.65597e-6 / 8.0);  
% Constantes  
s = ae + 78 / xkmper;  
qo = ae + 120 / xkmper;  
xke = sqrt((3600.0 * ge) / (xkmper^3));  
qoms2t = ((qo - s)^2)^2;  
temp2 = xke / (satdata.xno);  
a1 = temp2^tothrd;  
cosio = cos (satdata.xincl);  
theta2 = (cosio^2);  
x3thm1 = 3.0 * theta2 - 1.0;  
eosq = (satdata.eo^2);  
betao2 = 1.0 - eosq;  
betao = sqrt(betao2);  
del1 = 1.5 * CK2 * x3thm1 / ((a1^2) * betao * betao2);
```

```

ao = a1 * ( 1.0 - del1*((1.0/3.0) + del1 * (1.0 + (134.0/81.0) * del1)))
;
delo = 1.5 * CK2 * x3thm1 / ((ao^2) * betaao * betaao2);
xnodp = (satdata.xno)/(1.0 + delo);
aodp = ao/(1.0 - delo);
% Inicialización

% Cuando el perigeo es menor a 220 km, la isim flag está fijada y las
% ecuaciones están truncadas
% a una variación lineal en sqrt a y variación cuadrática en anomalía
% media. También, el término c3, el término delta omega, y el término
% delta
% m se descartan.

isimp = 0;
if ((aodp * (1.0 - satdata.eo)/ ae) < (220.0/xkmper + ae))
    isimp = 1;
end
% Con perigeos por debajo de los 156 km, los valores de s y qoms2t son
% alterados.

s4 = s;
qoms24 = qoms2t;
perige = (aodp * (1.0 - satdata.eo) - ae) * xkmper;
if (perige < 156)
    s4 = perige - 78.0;
    if (perige <= 98)
        s4 = 20.0;
    end
    qoms24 = (((120.0 - s4) * ae / xkmper)^4.0);
    s4 = s4 / xkmper + ae;
end
pinvsq = 1.0 / ( (aodp^2) * (betaao2^2) );
tsi = 1.0 / (aodp - s4);
eta = aodp * (satdata.eo) * tsi;
etasq = (eta^2);
eeta = (satdata.eo) * eta;
psisq = abs( 1.0 - etasq);
coef = qoms24 * (tsi^4.0);
coef1 = coef / (psisq^3.5);
c2 = coef1 * xnodp * (aodp * (1.0 + 1.5 * etasq + eeta * (4.0 + etasq))
    + 0.75 * CK2 * tsi / psisq * x3thm1 * (8.0 + 3.0 * etasq * (8.0 +
    etasq)));
c1 = (satdata.bstar) * c2;
sinio = sin(satdata.xincl);
a3ovk2 = -XJ3 / CK2 * (ae^3.0);
c3 = coef * tsi * a3ovk2 * xnodp * ae * sinio / (satdata.eo);
x1mth2 = 1.0 - theta2;
c4 = 2.0 * xnodp * coef1 * aodp * betaao2 * ( eta * (2.0 + 0.5 * etasq) +
    (satdata.eo) * (0.5 + 2.0 * etasq) - 2.0 * CK2 * tsi / (aodp *

```

```

psisq) * ( -3.0 * x3thm1 * ( 1.0 - 2.0 * eeta + etasq * (1.5 - 0.5*
eeta)) + 0.75 * x1mth2 * (2.0 * etasq - eeta * (1.0 + etasq)) * cos
(2.0 * (satdata.omegao))));;
c5 = 2.0 * coef1 * aodp * betao2 * (1.0 + 2.75 * (etasq + eeta) + eeta *
etasq);
theta4 = (theta2^2);
temp1 = 3.0 * CK2 * pinvsq * xnodp;
temp2 = temp1 * CK2 * pinvsq;
temp3 = 1.25 * CK4 * pinvsq * pinvsq * xnodp;
xmdot = xnodp + 0.5 * temp1 * betao * x3thm1 + 0.0625 * temp2 * betao *
(13.0 - 78.0 * theta2 + 137.0 * theta4);
x1m5th = 1.0 - 5.0 * theta2;
omgdot = -0.5 * temp1 * x1m5th + 0.0625 * temp2 * (7.0 - 114.0 * theta2
+ 395.0 * theta4) + temp3 * (3.0 - 36.0 * theta2 + 49.0 * theta4);
xhdot1 = -temp1 * cosio;
xnodot = xhdot1 + (0.5 * temp2 * (4.0 - 19.0 * theta2) + 2.0 * temp3 *
(3.0 - 7.0 * theta2)) * cosio;
omgcof = (satdata.bstar) * c3 * cos(satdata.omegao);
xmcof = -(2.0/3.0) * coef * (satdata.bstar) * ae / eeta;
xnodcf = 3.5 * betao2 * xhdot1 * c1;
t2cof = 1.5 * c1;
xlcof = 0.125 * a3ovk2 * sinio * (3.0 + 5.0 * cosio) / (1.0 + cosio);
aycof = 0.25 * a3ovk2 * sinio;
delmo = ((1.0 + eta * cos(satdata.xmo))^3);
sinmo = sin(satdata.xmo);
x7thm1 = 7.0 * theta2 - 1.0;
if (isimp==0)
  c1sq = (c1^2);
  d2 = 4.0 * aodp * tsi * c1sq;
  temp = d2 * tsi * c1 / 3.0;
  d3 = (17.0 * aodp + s4)*temp;
  d4 = 0.5 * temp * aodp * tsi * (221.0 * aodp + 31.0 * s4) * c1;
  t3cof = d2 + 2.0*c1sq;
  t4cof = 0.25 * (3.0 * d3 + c1 * (12.0 * d2 + 10.0 * c1sq));
  t5cof = 0.2 * (3.0 * d4 + 12.0 * c1 * d3 + 6.0 * d2 * d2 + 15.0 *
c1sq * (2.0 * d2 + c1sq));
end
% Actualización para la gravedad secular y el drag atmosférico
xmdf = satdata.xmo + xmdot * tsince;
omgadf = satdata.omegao + omgdot * tsince;
xnoddf = satdata.xnodeo + xnodot * tsince;
omega = omgadf;
xmp = xmdf;
tsq = (tsince^2);
xnode = xnoddf + xnodcf * tsq;
tempa = 1.0 - c1 * tsince;
tempe = (satdata.bstar) * c4 * tsince;
templ = t2cof * tsq;
if (isimp == 0)
  delomg = omgcof * tsince;

```

```

delm = xmcof*((1.0 + eta * cos(xmdf))^ 3.0) - delmo);
temp = delomg + delm;
xmp = xmdf + temp;
omega = omgadf - temp;
tcube = tsq * tsince;
tfour = tsince * tcube;
tempa = tempa - d2 * tsq - d3 * tcube - d4 * tfour;
tempe = tempe + (satdata.bstar) * c5 * (sin(xmp) - sinmo);
templ = templ + t3cof * tcube + tfour * (t4cof + tsince * t5cof);
end
a = aodp * (tempa^2);
e = (satdata.eo) - tempe;
xl = xmp + omega + xnode + xnodp*templ;
beta = sqrt(1.0 - (e^2));
xn = xke / (a^1.5);
% Largo período
axn = e * cos(omega);
temp = 1.0 / (a * (beta^2));
xll = temp * xlcof * axn;
aynl = temp * aycof;
xlt = xl + xll;
ayn = e * sin(omega) + aynl;
% Resolución de la ecuación de Kepler
capu = fmod2p(xlt - xnode);
temp2 = capu;
i=1;
while(1)
    sinepw = sin(temp2);
    cosepw = cos(temp2);
    temp3 = axn * sinepw;
    temp4 = ayn * cosepw;
    temp5 = axn * cosepw;
    temp6 = ayn * sinepw;
    epw = (capu - temp4 + temp3 - temp2) / (1.0 - temp5 - temp6) + temp2
    ;
    temp7 = temp2;
    temp2 = epw;
    i = i+1;
    if ((i>10) || (abs(epw - temp7) <= e6a))
        break
    end
end

% Valores preliminares del corto período
ecose = temp5 + temp6;
esine = temp3 - temp4;
elsq = (axn^2) + (ayn^2);
temp = 1.0 - elsq;
pl = a * temp;
r = a * (1.0 - ecose);

```

```

temp1 = 1.0 / r;
rdot = xke * sqrt(a) * esine * temp1;
rfdot = xke * sqrt(pl) * temp1;
temp2 = a * temp1;
betal = sqrt(temp);
temp3 = 1.0 / (1.0 + betal);
cosu = temp2 * (cosepw - axn + ayn * esine * temp3);
sinu = temp2 * (sinepw - ayn - axn * esine * temp3);
u = actan(sinu, cosu);
sin2u = 2.0 * sinu * cosu;
cos2u = 2.0 * (cosu^2) - 1.0;
temp = 1.0 / pl;
temp1 = CK2 * temp;
temp2 = temp1 * temp;
% Actualización del corto período
rk = r * (1.0 - 1.5 * temp2 * betal * x3thm1) + 0.5 * temp1 * x1mth2 *
cos2u;
uk = u - 0.25 * temp2 * x7thm1 * sin2u;
xnodek = xnode + 1.5 * temp2 * cosio * sin2u;
xinck = (satdata.xinck) + 1.5 * temp2 * cosio * sinio * cos2u;
rdotk = rdot - xn * temp1 * x1mth2 * sin2u;
rfdotk = rfdot + xn * temp1 * (x1mth2 * cos2u + 1.5 * x3thm1);
% Vectores de orientación
MV.v(1) = -sin(xnodek) * cos(xinck);
MV.v(2) = cos(xnodek) * cos(xinck);
MV.v(3) = sin(xinck);

NV.v(1) = cos(xnodek);
NV.v(2) = sin(xnodek);
NV.v(3) = 0;

for i=1:3
    UV.v(i) = MV.v(i) * sin(uk) + NV.v(i) * cos(uk);
    VV.v(i) = MV.v(i) * cos(uk) - NV.v(i) * sin(uk);
end

% posición + velocidad
for i=1:3
    pos.v(i) = rk * UV.v(i);
    vel.v(i) = rdotk * UV.v(i) + rfdotk * VV.v(i);
end

[pos, vel] = Convert_Sat_State(pos, vel);

```

## A.2 SGP8

Este es el código que representa en código las ecuaciones explicadas en el apartado A.2 [19]:

---

**Código A.2** Código del modelo de perturbaciones simplificado SGP8.

---

```
%-----
%----- sgp8 -----
%-----
function [pos, vel] = sgp8 (tsince, satdata)

ae = 1.0;
xmnpda = 1440.0;
tothrd = (2.0/3.0);
XJ3 = -2.53881e-6;
e6a = 1.0E-6;
xkmper = 6378.135;
ge = 398600.8; % Constante gravitacional de la Tierra
CK2 = (1.0826158e-3 / 2.0);
CK4 = (-3.0 * -1.65597e-6 / 8.0);

% Constantes
rho = (2.461 * 0.00001 * xkmper);
s = ae + 78 / xkmper;
qo = ae + 120 / xkmper;
xke = sqrt((3600.0 * ge)/(xkmper^3));
qoms2t = ((qo - s)^2)^2;
temp2 = xke / (satdata.xno);
a1 = (temp2^tothrd);
cosio = cos(satdata.xincl);
sinio = sin(satdata.xincl);
delta1 = 1.5 * CK2 * (3.0 * cosio * cosio - 1.0) / ((a1^2) * ((1.0 - (
    satdata.eo^2))^1.5));
a0 = a1 * (1.0 - (1.0/3.0) * delta1 - (delta1^2) - (134.0/81.0) * (
    delta1^2) * delta1);
delta0 = 1.5 * CK2 * (3.0 * cosio * cosio - 1.0) / ((a0^2) * ((1.0 - (
    satdata.eo^2))^1.5));
noii = satdata.xno / (1.0 + delta0);
aoii = a0 / (1.0 - delta0);
%Término B
B = 2.0 * (satdata.bstar) / rho;
%constantes II
beta = sqrt(1.0 - (satdata.eo^2));
theta = cosio;
temp = (noii * CK2) / ((aoii^2) * (beta^2) * beta);
M1dot = -1.5 * temp * (1.0 - 3.0 * (theta^2));
omega1dot = -1.5 * (temp / beta) * (1.0 - 5.0*(theta^2));
xnode1dot = -3.0 * (temp / beta) * theta;
M2dot = (3.0/16.0) * (temp * CK2 / ((aoii^2) * (beta^4.0))) * (13.0 -
    78.0 * (theta^2) + 137.0 * (theta^4.0));
```

---

```

omega2dot = (3.0/16.0) * (temp * CK2 / ((aoii^2) * (beta^5.0))) * (7.0 -
114.0 *(theta^2)+ 395.0 * (theta^4)) + 1.25 * (noii * CK4) / ((aoii
^4.0) * (beta^8.0))* (3.0 - 36.0 * (theta^2) + 49.0 * (theta^4));
xnode2dot = 1.5 * (temp * CK2 / ((aoii^2) * (beta^5.0))) * theta * (4.0
- 19.0 * (theta^2))+ 2.5 * (noii * CK4) / ((aoii^4.0) * (beta^8.0))
* theta * (3.0 - 7.0*(theta^2));
ldot = noii + M1dot + M2dot;
omegadot = omega1dot + omega2dot;
xnodedot = xnode1dot + xnode2dot;
tsi = 1.0/(aoii * (beta^2) - s);
eta = satdata.eo * s * tsi;
phi = sqrt(1.0 - (eta^2));
alpha = sqrt(1.0 + (satdata.eo^2));
C0 = 0.5 * B * rho * qoms2t * noii * aoii * (tsi^4.0) / (alpha * (phi
^7.0));
C1 = 1.5 * noii * (alpha^4.0) * C0;
D1 = (tsi * (1.0 / (phi^2))) / (aoii * (beta^2));
D2 = 12.0 + 36.0 * (eta^2) + 4.5 * (eta^4.0);
D3 = 15.0 * (eta^2) + 2.5 * (eta^4.0);
D4 = 5.0 * eta + (15.0 / 4.0) * (eta^3.0);
D5 = tsi / (phi^2);
B1 = -CK2 * (1.0 - 3.0 * (theta^2));
B2 = -CK2 * (1.0 - (theta^2));
A30 = -XJ3 * (ae^3.0) / CK2;
B3 = A30 * sinio;
C2 = D1 * D3 * B2;
C3 = D4 * D5 * B3;
n0dot = C1 * ((2.0 + (eta^2) * (3.0 + 34.0 * (satdata.eo^2)) + 5.0 *
satdata.eo) * (eta) * (4.0 + (eta^2)) + 8.5 * (satdata.eo^2)) + D1 *
D2 * B1 + C2 * cos(2.0 * satdata.omegao) + C3 * sin(satdata.omegao)
);
D6 = 30 * eta + 22.5 * (eta^3);
D7 = 5.0 * eta + 12.5 * (eta^3);
D8 = 1.0 + 6.75 * (eta^2) + (eta^4);
C4 = D1 * D7 * B2;
C5 = D5 * D8 * B3;
e0dot = -C0 * (eta * (4.0 + (eta^2) + (satdata.eo^2) * (15.5 + 7.0 *
(eta^2))) + satdata.eo * (5.0 + 15.0 * (eta^2)) + D1 * D6 * B1 + C4 *
cos(2.0 * satdata.omegao) + C5 * sin(satdata.omegao));
alphadottoalpha = satdata.eo * e0dot / ((alpha^2));
C6 = (1.0/3.0) * (n0dot/noii);
tsidottotsi = 2.0 * aoii * tsi * (C6 * (beta^2) + satdata.eo * e0dot);
etadot = (e0dot + satdata.eo * tsidottotsi) * s * tsi;
phidottophi = -eta * etadot / (phi^2);
C0dottoC0 = C6 + 4.0 * tsidottotsi - alphadottoalpha - 7.0 * phidottophi
;
C1dottoC1 = n0dot / noii + 4.0 * alphadottoalpha + C0dottoC0;
D9 = 6.0 * eta + 20.0 * satdata.eo + 15.0 * satdata.eo * (eta^2) + 68.0
* (satdata.eo^2) * eta;

```

```

D10 = 20.0 * eta + 5.0 * (eta^3) + 17.0 * satdata.eo + 68 * (satdata.eo)
    * (eta^2);
D11 = eta * (72.0 + 18.0 * (eta^2));
D12 = eta * (30.0 + 10.0 * (eta^2));
D13 = 5.0 + 11.25*(eta^2);
D14 = tsidottotsi - 2.0 * phidottophi;
D15 = 2.0 * (C6 + satdata.eo * e0dot / (beta^2));
D1dot = D1 * (D14 + D15);
D2dot = etadot * D11;
D3dot = etadot * D12;
D4dot = etadot * D13;
D5dot = D5 * D14;
C2dot = B2 * (D1dot * D3 + D1 * D3dot);
C3dot = B3 * (D5dot * D4 + D5 * D4dot);
D16 = D9 * etadot + D10 * e0dot + B1 * (D1dot * D2 + D1 * D2dot) + C2dot
    * cos(2.0 * satdata.omegao) + C3dot * sin(satdata.omegao)+omega1dot
    * (C3 * cos(satdata.omegao) - 2 * C2 * sin(2.0 * satdata.omegao));
n0ddot = n0dot * C1dottoC1 + C1 * D16;
e0ddot = e0dot * C0dottoC0 - C0 * ((4.0 + 3.0 * (eta^2) + 30.0 * satdata
    .eo * eta + 15.5 * (satdata.eo^2) + 21.0 * (eta^2) * (satdata.eo^2))
    * etadot + (5.0 + 15.0 * (eta^2) + 31.0 * satdata.eo * eta + 14.0 *
    satdata.eo * (eta^3)) * e0dot + B1 * (D1dot * D6 + D1 * etadot *
    (30.0 + 67.5 * (eta^2)))+ B2 * (D1dot * D7 + D1 * etadot * (5.0 +
    37.5 * (eta^2))) * cos(2.0 * satdata.omegao)+ B3 * (D5dot * D8 + D5
    * eta * etadot * (13.5 + 4 * (eta^2))) * sin(satdata.omegao) +
    omega1dot * (C5 * cos(satdata.omegao) - 2.0 * C4 * sin(2.0 * satdata.
    omegao)));
D17 = n0ddot / noii - (n0dot/noii)^2;
tsiddottotsi = 2.0 * (tsidottotsi - C6) *tsidottotsi + 2.0 * aoii * tsi
    * ((1.0/3.0) * D17 * (beta^2) - 2 * C6 * satdata.eo * e0dot + (e0dot
    ^2) + satdata.eo * e0ddot);
etaddot = (e0ddot + 2.0 * e0dot * tsidottotsi) * s * tsi + eta *
    tsiddottotsi;
D18 = tsiddottotsi - (tsidottotsi^2);
D19 = -(phidottophi^2) * (1.0 + 1.0 / (eta^2)) - eta * etaddot / (phi^2)
    ;
D1ddot = D1dot * (D14+D15) + D1 * (D18 - 2.0 * D19 + (2.0/3.0) * D17 +
    2.0 * (alpha^2) * (e0dot^2) / (beta^4.0) + 2.0 * satdata.eo * e0ddot
    / (beta^2));
n0tdot = n0dot * (2.0 * (2.0/3.0) * D17 + 3.0 * ((e0dot^2) + satdata.eo
    * e0ddot) / (alpha^2) - 6.0 * ( satdata.eo * e0dot / (alpha^2))^2 +
    4.0 * D18 - 7.0 * D19 ) + C1dottoC1 * n0ddot;
n0tdot = n0tdot + C1 * (C1dottoC1 * D16 + D9 * etaddot + D10 * e0ddot +
    (etadot^2) * (6.0 + 30.0 * (satdata.eo) * eta + 68.0 * (satdata.eo
    ^2)) + etadot * e0dot * (40.0 + 30.0 * (eta^2) + 272.0 * satdata.eo
    * eta) + (e0dot^2) * (17.0 + 68.0 * (eta^2)) + B1 * (D1ddot * D2+
    2.0 * D1dot * D2dot + D1 * (etaddot * D11 + (etadot^2) * (72. + 54.
    * (eta^2))))+ B2 * (D1ddot * D3 + 2.0 * D1dot * D3dot + D1 * (
    etaddot * D12 + (etadot^2) * (30.0 + 30.0 * (eta^2)))) * cos(2.0 *
    satdata.omegao) + B3 * ((D5dot * D14 + D5 * (D18 - 2.0 * D19)) * D4
    );

```

```

+ 2.0 * D4dot * D5dot + D5 * (etaddot * D13 + 22.5 * eta * (etadot
^2)) * sin(satdata.omegao) + omega1dot * ((7.0 * (1.0/3.0) * (n0dot
/ noii) + 4.0 * satdata.eo * e0dot / (beta^2)) * (C3 * cos(satdata.
omegao) - 2.0 * C2 * sin(2.0 * satdata.omegao))+ ((2.0 * C3dot * cos
(satdata.omegao) - 4.0 * C2dot * sin(2.0 * satdata.omegao) -
omega1dot * (C3 * sin(satdata.omegao) + 4 * C2 * cos(2.0 * satdata.
omegao))))));
smalln0ddot = n0ddot * 1.0E9;
temp = (smalln0ddot^2) - n0dot * 1.0E18 * n0tdot;
p = (temp + (smalln0ddot^2)) / temp;
gamma = - n0tdot / (n0ddot * (p - 2));
nd = n0dot / (p * gamma);
q = 1.0 - e0ddot / (e0dot * gamma);
ed = e0dot / (q * gamma);
% Drag atmosférico y gravedad
if ((abs( n0dot / noii * xmnpda)) < 0.00216)
    n = noii + n0dot * tsince;
    edot = -(2.0/3.0) * n0dot * (1.0 - satdata.eo) / noii;
    e = satdata.eo + edot * tsince;
    Z1 = 0.5 * n0dot * (tsince^2);
else
    n = noii + nd * (1.0 - ((1.0 - gamma * tsince)^p));
    edot = e0dot;
    e = satdata.eo + ed * (1.0 - ((1.0 - gamma * tsince)^q));
    Z1 = gamma * tsince;
    Z1 = 1.0 - Z1;
    Z1 = (Z1^(p+1.0));
    Z1 = Z1 - 1.0;
    Z1 = Z1 / (gamma * (p+1));
    Z1 = Z1 + tsince;
    Z1 = Z1 * n0dot / (p * gamma);
end
omega = satdata.omegao + omegadot * tsince + (7.0 * Z1) / (3.0 * noii) *
omega1dot;
xnode = satdata.xnodeo + xnodedot * tsince + xnode1dot * (7.0 * Z1)/(3.0
* noii);
M = satdata.xmo + tsince * ldot + Z1 + M1dot * (7.0 * Z1) / (3.0 * noii)
;
% Kepler
M = fmod2p(M);
temp = M + satdata.eo * sin(M) * (1 + e * cos(M));
i=1;
while(1)
    deltaEw = (M + e * sin(temp) - temp) / (1.0 - e * cos(temp));
    Ew = deltaEw + temp;
    temp = Ew;
    i=i+1;
    if ((i>10) || (abs(deltaEw) <= e6a))
        break
    end

```

```

end
% Corto período
a = ((xke / n)^tothrd);
beta = sqrt(1.0 - (e^2));
sinf = beta * sin(Ew) * (1.0 / (1.0 - e * cos(Ew)));
cosf = (cos(Ew) - e) * (1.0 / (1.0 - e * cos(Ew)));
f = actan(sinf, cosf);
sinu = sinf * cos(omega) + cosf * sin(omega);
cosu = cosf * cos(omega) - sinf * sin(omega);
rii = (a * (1.0 - (e^2))) / (1.0 + (e * cosf));
deltaR = (0.5 * CK2 * (1.0 / (a * (1.0 - (e^2))))) * ((1.0 - (theta^2))
    * (2.0 * (cosu^2) - 1.0) - 3.0 * (3.0 * (theta^2) - 1.0)) - (0.25 *
    A30 * sinio) * sinu;
temp = 3.0 * (0.5 * CK2 * (1.0 / (a * (1.0 - (e^2)))))^2 * sinio * (2.0
    * (cosu^2) - 1) - (0.25 * A30 * (1.0 / (a * (1 - (e^2))))) * e * sin
    (omega);
deltaI = temp * cosio;
deltaU = sin(satdata.xincl / 2) * ((0.5 * CK2 * (1.0 / (a * (1 - (e^2)))
    )^2) * (0.5 * (1.0 - 7.0 * (theta^2)) * (2.0 * sinu * cosu) - 3.0 *
    (1.0 - 5.0 * (theta^2)) * (f - M + e * sinf)) - (0.25 * A30 * (1.0 /
    (a * (1.0 - (e^2))))) * sinio * cosu * (2.0 + (e*cosf)) - 0.5 *
    (0.25 * A30 * (1.0 / (a * (1 - (e^2))))) * (theta^2) * e * cos(omega)
    ) / cos(satdata.xincl/2);
lambda = f + omega + xnode + (0.5 * CK2 * (1.0 / (a * (1.0 - (e^2))))^2)
    * (0.5 * (1.0 + 6.0 * cosio - 7.0 * (theta^2)) * (2.0 * sinu * cosu)
    ) - 3.0 * ((1.0 - 5.0 * (theta^2)) + 2.0 * cosio) * (f - M + e *
    sinf) + (0.25 * A30 * (1.0 / (a *(1 - (e^2))))) * sinio * (cosio *
    e * cos(omega) / (1.0 + cosio) - (2.0 + (e*cosf)) * cosu);
y4 = sin(satdata.xincl / 2.0) * sinu + cosu * deltaU + 0.5 * sinu * cos(
    satdata.xincl / 2.0)*deltaI;
y5 = sin(satdata.xincl / 2.0) * cosu - sinu * deltaU + 0.5 * cosu * cos(
    satdata.xincl / 2.0)*deltaI;
r = rii + deltaR;
rdot = n * a * e * sinf / beta + (-n * (a / rii)^2) * (1.0 * CK2 * (1.0
    / (a * (1.0 - (e^2)))) * (1.0 - (theta^2)) * (2.0 * sinu * cosu) +
    (0.25 * A30 * sinio) * cosu);
rfdot = n * (a^2) * beta / rii + (-n * (a/rii)^2) * deltaR + a * (n * (a
    / rii)) * sinio * temp;
% Vector de orientación
cosI2 = sqrt(1.0 - (y4^2) - (y5^2));
UV.v(1) = 2.0 * y4 * (y5 * sin(lambda) - y4 * cos(lambda)) + cos(lambda)
    ;
UV.v(2) = -2.0 * y4 * (y5 * cos(lambda) + y4 * sin(lambda)) + sin(lambda)
    ;
UV.v(3) = 2.0 * y4 *cosI2;
VV.v(1) = 2.0 * y5 * (y5 * sin(lambda) - y4 * cos(lambda)) - sin(lambda)
    ;
VV.v(2) = -2.0 * y5 * (y5 * cos(lambda) + y4 * sin(lambda)) + cos(lambda)
    ;
VV.v(3) = 2.0 * y5 * cosI2;

```

```
% Posición + velocidad
for i=1:3
    pos.v(i) = r * UV.v(i);
    vel.v(i) = rdot * UV.v(i) + rfdot * VV.v(i);
end

[pos, vel] = Convert_Sat_State(pos, vel);
```

### A.3 Test SGP8

A continuación, se muestra el código para implementar las ecuaciones del modelo SGP8, que parte del código de [19] y está modificado para crear un bucle para predicciones minuto a minuto durante 4 semanas en lugar de una predicción única en un tiempo determinado.

**Código A.3** Código de implementación del modelo de perturbaciones simplificado SGP8.

```
clc
clear
format long g
tic
global const
SAT_Const

ge = 398600.8; % Constante gravitatoria de la Tierra
TWOPI = 2*pi;
MINUTES_PER_DAY = 1440.;
MINUTES_PER_DAY_SQUARED = (MINUTES_PER_DAY * MINUTES_PER_DAY);
MINUTES_PER_DAY_CUBED = (MINUTES_PER_DAY * MINUTES_PER_DAY_SQUARED);

% Nombre del archivo TLE
fname = 'tle.txt';

% Abre el TLE y lee sus elementos
fid = fopen(fname, 'r');

% 19-32 04236.56031392 Epoca (UTC)
% 3-7 25544 Número de catálogo del satélite
% 9-16 51.6335 Inclinación de órbita (degrees)
% 18-25 344.7760 Ascensión (derecha) del nodo ascendente (degrees)
% 27-33 0007976 Excentricidad (decimal point assumed)
% 35-42 126.2523 Argumento del perigeo (degrees)
% 44-51 325.9359 Anomalía media (degrees)
% 53-63 15.70406856 Movimiento medio (revolutions/day)
% 64-68 32890 Número de revolución en la época

% Lee primera linea
tline = fgetl(fid);
Cnum = tline(3:7);                      % Número de catálogo (NORAD)
SC   = tline(8);                         % Clasificación de seguridad
```

```

ID    = tline(10:17);                      % Número de identificación
year = str2double(tline(19:20));            % Año
doy  = str2double(tline(21:32));            % Día del año
epoch = str2double(tline(19:32));           % Epoca
TD1   = str2double(tline(34:43));           % Primera derivada
TD2   = str2double(tline(45:50));           % Segunda derivada
ExTD2 = tline(51:52);                      % Exponente de la segunda
                                             derivada
BStar = str2double(tline(54:59));           % Término de drag
ExBStar = str2double(tline(60:61));          % Exponente del término de
                                             drag
BStar = BStar*1e-5*10^ExBStar;
Etype = tline(63);                         % Tipo de efeméride
Enum  = str2double(tline(65:end));           % Número de Elemento

% read second line
tline = fgetl(fid);
i = str2double(tline(9:16));                 % Inclinación de órbita (
                                             degrees)
raan = str2double(tline(18:25));              % Ascensión (derecha) del
                                             nodo ascendente (degrees)
e = str2double(strcat('0.',tline(27:33)));  % Excentricidad
omega = str2double(tline(35:42));             % Argumento del perigeo (
                                             degrees)
M = str2double(tline(44:51));                 % Anomalía media (degrees)
no = str2double(tline(53:63));                 % Movimiento medio
a = ( ge/(no*2*pi/86400)^2 )^(1/3);         % semieje mayor (m)
rNo = str2double(tline(65:end));               % Número de revolución de la
                                             época

fclose(fid);

satdata.epoch = epoch;
satdata.norad_number = Cnum;
satdata.bulletin_number = ID;
satdata.classification = SC; % casi siempre 'U'
satdata.revolution_number = rNo;
satdata.ephemeris_type = Etype;
satdata.xmo = M * (pi/180);
satdata.xnodeo = raan * (pi/180);
satdata.omegao = omega * (pi/180);
satdata.xincl = i * (pi/180);
satdata.eo = e;
satdata.xno = no * TWOPI / MINUTES_PER_DAY;
satdata.xndt2o = TD1 * 1e-8 * TWOPI / MINUTES_PER_DAY_SQUARED;
satdata.xndd6o = TD2 * TWOPI / MINUTES_PER_DAY_CUBED;
satdata.bstar = BStar;

longt=1440*7*4;%tiempo que se propaga
tsince = 1:longt;

```

```

rteme= zeros (3,longt);
vteme= zeros (3,longt);

for c =1:longt
[rteme(:,c), vteme(:,c)] = sgp8(c, satdata);
end

% lee los parámetros de orientación de la Tierra
fid = fopen('eop19620101.txt','r');
%
-----
% / Fecha   MJD      x          y          UT1-UTC      LOD      dPsi      dEpsilon
%           dX       dY      DAT           "           "           "           "
% / (0h UTC)           "          "           "           s           s           "
%           "          "          s
%
eodata = fscanf(fid,'%i %d %d %i %f %f %f %f %f %f %f %f %i',[13 inf]);
fclose(fid);

if (year < 57)
    year = year + 2000;
else
    year = year + 1900;
end

MJD_UTC = zeros (longt);
reci= zeros (3,longt);
veci= zeros (3,longt);
recef= zeros (3,longt);
vecef= zeros (3,longt);
rtod= zeros (3,longt);
vtod= zeros (3,longt);
[mon,day,hr,minute,sec] = days2mdh(year,doy);
MJD_Epoch = Mjday(year,mon,day,hr,minute,sec);

for k=1:longt

MJD_UTC(k) = MJD_Epoch+k/1440;

% Parámetros de orientación de la Tierra
[x_pole,y_pole,UT1_UTC,LOD,dpsideps,dx_pole,dy_pole,TAI_UTC] = IERS(
    eodata,MJD_UTC(k),'l');
[UT1_TAI,UTC_GPS,UT1_GPS,TT_UTC,GPS_UTC] = timediff(UT1_UTC,TAI_UTC);
MJD_UT1 = MJD_UTC(k) + UT1_UTC/86400;
MJD_TT = MJD_UTC(k) + TT_UTC/86400;
T = (MJD_TT-const.MJD_J2000)/36525;

```

```
[reci(:,k), veci(:,k)] = teme2eci(rteme(:,k),vteme(:,k),T,dpsideps);
[recef(:,k),vecef(:,k)] = teme2ecef(rteme(:,k),vteme(:,k),T,MJD_UT1
+2400000.5,LOD,x_pole,y_pole,0);
[rtod(:,k), vtod(:,k)] = ecef2tod(recef(:,k),vecef(:,k),T,MJD_UT1
+2400000.5,LOD,x_pole,y_pole,0,dpsi,deps);

end
JD_UTC=MJD_UTC(1)+2400000.5;
JD_Epoch=MJD_Epoch+2400000.5;
A=[tsince;reci;veci];
B=A';
csvwrite('outputTLE.csv',B);
toc
```

---

# Apéndice B

## Código de Python

---

En este apéndice se van a mostrar algunos ejemplos de las construcciones del código generado con el software Python. Debido a que ciertas estructuras se repiten, los siguientes apartados se han diseñado para no repetir código al mismo tiempo que se da una visión completa de los mismos.

### B.1 Primera aproximación

Para la primera aproximación, se ha empleado un código con estructura similar para los dos modelos. Los únicos elementos que cambian son la palabra *Ridge* por *Lasso* y el parámetro llamado *alpha* que cambia de 472,2 a 2,89.

Este es el código empleado:

---

#### Código B.1 Código del modelo Ridge.

```
#Importamos las librerías necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.pyplot as pl

#Cargamos los datos y los dividimos en entrenamiento y test
from sklearn import model_selection
df_x = pd.read_csv('outputTLE.csv')
df_y = pd.read_csv('outputSimulinkV2.csv')
x = df_x.to_numpy()
y = df_y.to_numpy()
x_train = x[:1400*14,:]
y_train = y[:1400*14,:]
x_test = x[1400*14:,:]
y_test = y[1400*14:,:]
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)

#Se carga el modelo de ML con el parámetro alpha óptimo
from sklearn.linear_model import Ridge
```

```

from sklearn import metrics
reg = Ridge(alpha=472.72, fit_intercept=False, normalize=True)
reg.fit(x_train, y_train)
pred_test= reg.predict(x_test)
print("Error test: " + str(metrics.mean_squared_error(y_test, pred_test,
    squared=False)))
pred_train = reg.predict(x_train)
print("Error training: " + str(metrics.mean_squared_error(y_train,
    pred_train, squared=False)))

#Se crea la gráfica con la posición diaria y el error diario
y_predx = []
y_truex = []
y_tlex = []
y_predy = []
y_truey = []
y_tley = []
y_predz = []
y_truez = []
y_tlez = []
error = []
for i in range(pred_test.shape[0]):
    if i % 1440 == 0:
        y_predx.append(pred_test[i,1])
        y_truex.append(y_test[i,1])
        y_tlex.append(x_test[i,1])
        y_predy.append(pred_test[i,2])
        y_truey.append(y_test[i,2])
        y_tley.append(x_test[i,2])
        y_predz.append(pred_test[i,3])
        y_truez.append(y_test[i,3])
        y_tlez.append(x_test[i,3])

        error = metrics.mean_squared_error(y_test[i,1:4] , pred_test[i
            ,1:4], squared=False)
        print("Mean Error: " + str(error))
        errortle = metrics.mean_squared_error(y_test[i,1:4] , x_test[i
            ,1:4], squared=False)
        print("Mean Error TLE: " + str(errortle))

plt.plot(y_truex,label="real")
plt.plot(y_predx,label="Ridge")
plt.legend(loc='upper right')
pl.xlabel("tiempo (días)")
pl.ylabel("coordenada x (km)")
pl.savefig("Ridge_x.pdf")
plt.show()

plt.plot(y_truey,label="real")
plt.plot(y_predy,label="Ridge")

```

```

plt.legend(loc='upper right')
pl.xlabel("tiempo (días)")
pl.ylabel("coordenada x (km)")
pl.savefig("Ridge_y.pdf")
plt.show()

plt.plot(y_truez,label="real")
plt.plot(y_predz,label="Ridge")
plt.legend(loc='upper right')
pl.xlabel("tiempo (días)")
pl.ylabel("coordenada x (km)")
pl.savefig("Ridge_z.pdf")
plt.show()

plt.plot(y_truex,label="real")
plt.plot(y_tlex,label="perturbaciones")
plt.legend(loc='upper right')
pl.xlabel("tiempo (días)")
pl.ylabel("coordenada x (km)")
pl.savefig("TLE_x.pdf")
plt.show()

plt.plot(y_truey,label="real")
plt.plot(y_tley,label="perturbaciones")
plt.legend(loc='upper right')
pl.xlabel("tiempo (días)")
pl.ylabel("coordenada y (km)")
pl.savefig("TLE_y.pdf")
plt.show()

plt.plot(y_truez,label="real")
plt.plot(y_tlez,label="perturbaciones")
plt.legend(loc='upper right')
pl.xlabel("tiempo (días)")
pl.ylabel("coordenada z (km)")
pl.savefig("TLE_z.pdf")
plt.show()

```

## B.2 Segunda aproximación

Esta sección se va a dividir en tres partes: en la primera se mostrará el código para hallar el modelo GBRT en la coordenada x. Este código se repite para y y z y cambiando las líneas de la generación del modelo, para SVR y para XGBoost; en la segunda se mostrará el código donde se recopilan las tres coordenadas y se obtienen los resultados del GBRT. Este código es idéntico para las otras dos técnicas.

### B.2.1 Modelo de una coordenada

En esta sección se mostrará el código para hallar el modelo GBRT en la coordenada x. Este código se repite para y y z y cambiando las líneas de la generación del modelo, para SVR y para XGBoost.

**Código B.2** Código del modelo de una coordenada.

```
# Importamos las librerías necesarias
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import copy

# Definición del Dataset
# Dividimos los datos en x e y con la longitud de propagación deseada
def create_dataset(data, n_in=5): #n_in es el número de minutos pasados
    usados para propagar
    x = np.zeros((data.shape[0]-n_in, n_in))
    for i in range(n_in):
        x[:, i] = data[i:-n_in+i] #cada columna sin los 96 últimos datos
    y = data[n_in:] #Datos después de n_in
    return x, y

# Se cargan los datos y se aplica la definición del Dataset
from sklearn import model_selection
val_split = 0.1 #conjunto de validación
df = pd.read_csv('outputSimulink8semanas.csv') #Se carga el modelo dinámico
data = df["xs"].to_numpy() #coordenada deseada
x, y = create_dataset(data, n_in=96) #se aplica la función anteriormente
# definida
#Se divide entre conjunto de entrenamiento y de test
x_train, x_test, y_train, y_test = model_selection.train_test_split(x, y
    , test_size=val_split, random_state=42)
print(x_train.shape) #Aquí están todas las muestras
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)

#Se carga y crea el modelo de ML
from sklearn import metrics
from sklearn.ensemble import GradientBoostingRegressor
# Se fijan todos los hiperparámetros del modelo
reg = GradientBoostingRegressor(random_state=0, n_estimators=1500,
    max_depth=15, verbose=1,
    n_iter_no_change=10, validation_fraction
    =0.2, learning_rate=0.01, tol=1e-7,
    loss='ls')

reg.fit(x_train, y_train)
prediction = reg.predict(x_test)
print("Error validation: " + str(metrics.mean_squared_error(y_test,
    prediction, squared=False))) #Devuelve el RMSE
prediction = reg.predict(x_train)
print("Error training: " + str(metrics.mean_squared_error(y_train,
    prediction, squared=False)))
```

```

# Dibuja los valores reales y los predecidos en función del tiempo
#Se entra con el modelo de ML, los datos reales y el nº de minutos a
dibujar
def plot_prediction(model, targets, num_steps=200):
    predictions = targets[:96]
    for i in range(num_steps):
        inp = predictions[-96:]
        inp = np.reshape(inp, (1, 96))
        output = model.predict(inp)
        predictions = np.concatenate((predictions, output), axis=0)
    plt.plot(targets[:num_steps])
    plt.plot(predictions[:num_steps])
    plt.show()

# Calcula el RMSE después de cada día
def get_error(model, targets, num_days=5):
    predictions = targets[:96]
    for i in range(1440*num_days):
        inp = predictions[-96:]
        inp = np.reshape(inp, (1, 96))
        output = model.predict(inp)
        predictions = np.concatenate((predictions, output), axis=0)
        if i % 1440 == 0:
            error = metrics.mean_squared_error(np.reshape(targets, (
                targets.shape[0], 1))[predictions.shape[0]], np.reshape(
                predictions, (predictions.shape[0], 1))[-1], squared=
                False)
            print("Mean Error: " + str(error))

# Se cargan las funciones definidas para comprobar su funcionamiento
df = pd.read_csv('outputSimulink8semanas.csv')
targets = df["xs"].to_numpy()
plot_prediction(reg, targets, 1000)
get_error(reg, targets, 20)

#Se guarda el modelo generado para el programa con los 3 ejes
from joblib import dump, load
dump(reg, 'x.joblib')
#clf = load('x.joblib')

```

## B.2.2 Modelo completo

En esta sección se mostrará el código donde se recopilan las tres coordenadas y se obtienen los resultados del GBRT. Este código es idéntico para las otras dos técnicas.

---

### Código B.3 Código del modelo general.

---

```

# Importamos las librerías necesarias
import numpy as np

```

```

import pandas as pd
import xgboost as xgb
import matplotlib.pylab as plt
import matplotlib.pyplot as pl
import copy

# Definición del Dataset
# Dividimos los datos en x e y con la longitud de propagación deseada
def create_dataset(data, n_in=5):#n_in es el número de minutos pasados
    usados para propagar
    x = np.zeros((data.shape[0]-n_in, n_in, 3))
    for i in range(n_in):
        x[:, i, :] = data[i:-n_in+i, :]#cada columna sin los 96 ultimos
datos
    y = data[n_in:, :]#Datos después de n_in
    return x, y

# Se cargan los datos
from sklearn import model_selection
from sklearn import metrics
from sklearn.ensemble import GradientBoostingRegressor
val_split = 0.1 #conjunto de validación
df = pd.read_csv('outputSimulink8semanas.csv')
targets = df[['xs", "ys", "zs']].to_numpy()
print(targets.shape)

# Se cargan los modelos
from joblib import dump, load
reg_x = load('x.joblib')
reg_y = load('y.joblib')
reg_z = load('z.joblib')
models = [reg_x, reg_y, reg_z]

#Dibuja los valores reales y los predecidos en función del tiempo
#Se entra con el modelo de ML, los datos reales y el nº de minutos a
dibujar
def plot_predictions(models, targets, num_steps=200):
    predictions = targets[:96, :]
    for i in range(num_steps):
        inp = predictions[-96:, :]
        x = models[0].predict(np.reshape(inp[:, 0], (1, 96)))
        y = models[1].predict(np.reshape(inp[:, 1], (1, 96)))
        z = models[2].predict(np.reshape(inp[:, 2], (1, 96)))
        output = np.array([x, y, z])
        output = np.reshape(output, (1, 3))
        predictions = np.concatenate((predictions, output), axis=0)
    plt.plot(targets[:num_steps, 0], label="real")
    plt.plot(predictions[:num_steps, 0], label="GBRT")
    # plt.title("xs")
    plt.legend(loc='upper right')

```

```

pl.xlabel("tiempo (min)")
pl.ylabel("coordenada x (km)")
pl.xlim([0, 192])
pl.savefig("GBRT_x.pdf")
plt.show()

plt.plot(targets[:num_steps, 1],label="real")
plt.plot(predictions[:num_steps, 1],label="GBRT")
# plt.title("ys")
plt.legend(loc='upper right')
pl.xlabel("tiempo (min)")
pl.ylabel("coordenada y (km)")
pl.xlim([0, 192])
pl.savefig("GBRT_y.pdf")
plt.show()

plt.plot(targets[:num_steps, 2],label="real")
plt.plot(predictions[:num_steps, 2],label="GBRT")
# plt.title("zs")
plt.legend(loc='upper right')
pl.xlabel("tiempo (min)")
pl.ylabel("coordenada z (km)")
pl.xlim([0, 192])
pl.savefig("GBRT_z.pdf")
plt.show()

# Calcula el RMSE después de cada día
def get_error(models, targets, num_days=200):
    predictions = targets[:96, :]
    for i in range(1440*num_days):
        inp = predictions[-96:, :]
        x = models[0].predict(np.reshape(inp[:, 0], (1, 96)))
        y = models[1].predict(np.reshape(inp[:, 1], (1, 96)))
        z = models[2].predict(np.reshape(inp[:, 2], (1, 96)))
        output = np.array([x, y, z])
        output = np.reshape(output, (1, 3))
        predictions = np.concatenate((predictions, output), axis=0)
        if i % 1440 == 0:
            error = metrics.mean_squared_error(np.reshape(targets, (
                targets.shape[0], 3))[predictions.shape[0]-1], np.reshape(
                predictions, (predictions.shape[0], 3))[-1], squared=
            False)
            print("Mean Error: " + str(error))

#Dibuja la función de las gráficas previamente definida
plot_predictions(models, targets, 200)

#Imprime la función del error diario previamente definida

```

```
get_error(models, targets, 28)
```

---

### B.2.3 Bloques de técnicas de Machine Learning

En esta sección se recopilarán las líneas de código en los tres modelos donde se diseñan las técnicas con sus hiperparámetros correspondientes.

---

#### Código B.4 Código de diseño de SVR.

```
#Se carga el modelo de ML y las librerías para normalizar los datos
from sklearn.svm import SVR
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
#regr = make_pipeline(StandardScaler(), SVR(C=1.0, epsilon=0.2, tol=1e
-7, verbose=True))
#Se normalizan los datos
scaler_x = StandardScaler().fit(x_train)
scaler_y = StandardScaler().fit(y_train)
x_train_t = scaler_x.transform(x_train)
y_train_t = scaler_y.transform(y_train)
x_test_t = scaler_x.transform(x_test)
y_test_t = scaler_y.transform(y_test)
reg = SVR(C=1.0, epsilon=0.2, tol=1e-7, cache_size=1000) #Se fijan los
hiperparámetros

reg.fit(x_train_t, y_train_t)
prediction = reg.predict(x_test_t)
prediction = np.reshape(prediction, (prediction.shape[0], 1))
prediction = scaler_y.inverse_transform(prediction)
print("Error validation: " + str(metrics.mean_squared_error(y_test,
    prediction, squared=False))) #Devuelve el RMSE
prediction = reg.predict(x_train_t)
prediction = np.reshape(prediction, (prediction.shape[0], 1))
prediction = scaler_y.inverse_transform(prediction)
print("Error training: " + str(metrics.mean_squared_error(y_train,
    prediction, squared=False)))
```

---

#### Código B.5 Código de diseño de GBRT.

```
#Se carga y crea el modelo de ML
from sklearn import metrics
from sklearn.ensemble import GradientBoostingRegressor
# Se fijan todos los hiperparámetros del modelo
reg = GradientBoostingRegressor(random_state=0, n_estimators=1500,
    max_depth=15, verbose=1,
    n_iter_no_change=10, validation_fraction
    =0.2, learning_rate=0.01, tol=1e-7,
    loss='ls')
```

---

```
reg.fit(x_train, y_train)
prediction = reg.predict(x_test)
print("Error validation: " + str(metrics.mean_squared_error(y_test,
    prediction, squared=False)))#Devuelve el RMSE
prediction = reg.predict(x_train)
print("Error training: " + str(metrics.mean_squared_error(y_train,
    prediction, squared=False)))
```

**Código B.6** Código de diseño de XGBoost.

```
#Se carga y crea el modelo de ML
from sklearn import metrics
es = xgb.callback.EarlyStopping(
    rounds=2,
    save_best=True,
    metric_name="rmse",
)#Parada del programa si se llega a una solución válida
#Se fijan los hiperparámetros del modelo
reg = xgb.XGBRegressor(objective='reg:squarederror', n_estimators=10000,
    max_depth=25, subsample=0.6, learning_rate=0.01, #colsample_bytree
    =0.8,
    eval_metric=metrics.mean_absolute_error)
reg.fit(x_train, y_train, callbacks=[es], eval_set=[(x_test, y_test)])
prediction = reg.predict(x_test)
print("Error validation: " + str(metrics.mean_squared_error(y_test,
    prediction, squared=False)))#Devuelve el RMSE
prediction = reg.predict(x_train)
print("Error training: " + str(metrics.mean_squared_error(y_train,
    prediction, squared=False)))
```



# Índice de Figuras

---

1.1	CubeSats desde 1U hasta 16U	3
1.2	Imagen del Veery Hatchling "Clay" donde se puede comparar con el tamaño de una mano	4
1.3	Entorno de trabajo de MATLAB	5
1.4	Estructura de un árbol de decisiones	7
1.5	Ejemplo gráfico del funcionamiento de una técnica de Gradient Boosting [27]	8
1.6	Representación gráfica del funcionamiento de la técnica SVR [15]	9
2.1	Diagrama de flujo del proceso de propagación de órbita para los modelos SGP4 y SDP4 [30]	12
2.2	Elementos keplerianos para un satélite en órbita	13
2.3	Código TLE de estudio con la explicación de sus cifras	14
3.1	Capa superior del proyecto de Simulink del modelo dinámico	22
3.2	Bloque principal de representación del entorno del satélite	23
3.3	Cuadro de diálogo para escoger los parámetros de posicionamiento del satélite e instante de inicio de la simulación	24
3.4	Interior del bloque del modelo del satélite	25
3.5	Sistema de control de actitud	26
3.6	Sistema de dinámica del vehículo	27
3.7	Evolución de coordenadas geodéticas durante una hora de simulación	28
3.8	Evolución de coordenadas ECEF durante una hora de simulación	28
3.9	Evolución de coordenadas ECI durante una hora de simulación	29
3.10	Bloque de gráficas y sistema de almacenamiento de información	29
3.11	Representación gráfica del satélite orbitando alrededor de la Tierra	30
4.1	Trayectorias durante la primera hora de simulación del modelo de perturbaciones simplificado y el modelo dinámico	31
4.2	Trayectorias durante la última hora de simulación del modelo de perturbaciones simplificado y el modelo dinámico	32
4.3	Distancia entre la trayectoria de los dos modelos a lo largo del tiempo	32
4.4	Diferencia entre los módulos de la velocidad de los dos modelos a lo largo del tiempo	33
5.1	Error cuadrático medio con respecto al parámetro $\alpha$ para el modelo de regresión Ridge	36
5.2	Error cuadrático medio con respecto al parámetro $\alpha$ para el modelo de regresión Lasso	36
6.1	Evolución de la órbita a lo largo del tiempo, eje x	42
6.2	Evolución de la órbita a lo largo del tiempo, eje y	42
6.3	Evolución de la órbita a lo largo del tiempo, eje z	43

6.4	Evolución de la órbita a lo largo del tiempo, eje x	44
6.5	Evolución de la órbita a lo largo del tiempo, eje y	44
6.6	Evolución de la órbita a lo largo del tiempo, eje z	45
6.7	Evolución de la órbita a lo largo del tiempo, eje x	46
6.8	Evolución de la órbita a lo largo del tiempo, eje y	46
6.9	Evolución de la órbita a lo largo del tiempo, eje z	47
6.10	Evolución de la órbita a lo largo del tiempo, eje x	48
6.11	Evolución de la órbita a lo largo del tiempo, eje y	48
6.12	Evolución de la órbita a lo largo del tiempo, eje z	49
6.13	Evolución de la órbita durante las dos primeras rotaciones, eje x	51
6.14	Evolución de la órbita tras una semana, eje x	52
6.15	Evolución de la órbita tras dos semanas, eje x	52
6.16	Evolución de la órbita tras tres semanas, eje x	53
6.17	Evolución de la órbita tras cuatro semanas, eje x	53
6.18	Evolución de la órbita durante las dos primeras rotaciones, eje y	54
6.19	Evolución de la órbita tras una semana, eje y	54
6.20	Evolución de la órbita tras dos semanas, eje y	55
6.21	Evolución de la órbita tras tres semanas, eje y	55
6.22	Evolución de la órbita tras cuatro semanas, eje y	56
6.23	Evolución de la órbita durante las dos primeras rotaciones, eje z	56
6.24	Evolución de la órbita tras una semana, eje z	57
6.25	Evolución de la órbita tras dos semanas, eje z	57
6.26	Evolución de la órbita tras tres semanas, eje z	58
6.27	Evolución de la órbita tras cuatro semanas, eje z	58
6.28	Evolución de la órbita durante las dos primeras rotaciones, eje x	59
6.29	Evolución de la órbita tras una semana, eje x	60
6.30	Evolución de la órbita tras dos semanas, eje x	60
6.31	Evolución de la órbita tras tres semanas, eje x	61
6.32	Evolución de la órbita tras cuatro semanas, eje x	61
6.33	Evolución de la órbita durante las dos primeras rotaciones, eje y	62
6.34	Evolución de la órbita tras una semana, eje y	62
6.35	Evolución de la órbita tras dos semanas, eje y	63
6.36	Evolución de la órbita tras tres semanas, eje y	63
6.37	Evolución de la órbita tras cuatro semanas, eje y	64
6.38	Evolución de la órbita durante las dos primeras rotaciones, eje z	64
6.39	Evolución de la órbita tras una semana, eje z	65
6.40	Evolución de la órbita tras dos semanas, eje z	65
6.41	Evolución de la órbita tras tres semanas, eje z	66
6.42	Evolución de la órbita tras cuatro semanas, eje z	66
6.43	Comparación de los errores absolutos durante dos semanas	68
6.44	Comparación de los errores absolutos durante cuatro semanas	69
6.45	Comparación de los errores absolutos de todos los modelos	69

# Índice de Tablas

---

6.1	Error absoluto cada día (SGP8)	43
6.2	Error absoluto cada día (Ridge)	45
6.3	Error absoluto cada día (Lasso)	47
6.4	Error absoluto cada día (SVR)	49
6.5	Error absoluto cada día (GBRT)	59
6.6	Error absoluto cada día (XGBoost)	67



# Índice de Códigos

---

A.1	Código del modelo de perturbaciones simplificado SGP4	73
A.2	Código del modelo de perturbaciones simplificado SGP8	78
A.3	Código de implementación del modelo de perturbaciones simplificado SGP8	83
B.1	Código del modelo Ridge	87
B.2	Código del modelo de una coordenada	90
B.3	Código del modelo general	91
B.4	Código de diseño de SVR	94
B.5	Código de diseño de GBRT	94
B.6	Código de diseño de XGBoost	95



# Bibliografía

---

- [1] *Numpy*, 11 2021, <https://numpy.org/>.
- [2] *Simulación y diseño basado en modelos con simulink*, 11 2021, <https://es.mathworks.com/products/simulink.html>.
- [3] Careweather, *Hatching veery "clay"*.
- [4] MATLAB Central, *Sgp4*, 11 2021, <https://es.mathworks.com/matlabcentral/fileexchange/62013-sgp4>.
- [5] Nanosats Database, *What is a cubesat other picosatellites*, 11 2021, <https://www.nanosats.eu/cubesat>.
- [6] Federico Díaz, Nicanor Casas, Graciela De Luca, Sergio Martín, Daniel Alberto Julianelli, and Fernando Gustavo Tinetti, *Análisis de rendimiento del algoritmo sgp4/sdp4 para predicción de posición orbital de satélites artificiales utilizando contadores de hardware*, Congreso Argentino de Ciencias de la Computación, vol. 18, 2013.
- [7] Matplotlib 3.4.3 documentation, *Matplotlib: Python plotting*, 11 2021, <https://matplotlib.org/>.
- [8] Python España, *Aprende python*, 11 2021, <https://es.python.org/pages/aprende-python.html>.
- [9] Máxima Formación, *Qué son los árboles de decisión y para qué sirven*.
- [10] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *Boosting and additive trees*, The elements of statistical learning, Springer, 2009, pp. 337–387.
- [11] CG Hilton and JR Kuhlman, *Mathematical models for the space defense center*, Philco-Ford Publication No. U-3871 **17** (1966), 28.
- [12] Arthur E Hoerl and Robert W Kennard, *Ridge regression: Biased estimation for nonorthogonal problems*, Technometrics **12** (1970), no. 1, 55–67.
- [13] Felix R Hoots, *Spacetrack report no. 3, models for propagation of norad element sets*, <http://www.itc.nl/-bakker/orbit.html> (1980).
- [14] Richard S Hujšak, *A restricted four body solution for resonating satellites without drag*, Tech. report, Aerospace Defense Command Peterson AFB co Office of Astrodynamics, 1979.
- [15] Aprende IA, *Vectores de soporte regresión*, 11 2021, <https://aprendeia.com/maquina-de-vectores-de-soporte-regresion-teoria/>.

- [16] Yoshihide Kozai, *The motion of a close earth satellite*, The Astronomical Journal **64** (1959), 367.
- [17] Scikit learn 1.0.1 documentation, *Scikit-learn: machine learning in python*, 11 2021, <https://scikit-learn.org/stable/>.
- [18] Python Data Analysis Library, *Pandas*, 11 2021, <https://pandas.pydata.org/>.
- [19] MATLAB, *Sgp8*, 11 2021, <https://es.mathworks.com/matlabcentral/fileexchange/75492-sgp8>.
- [20] n2yo, *Veery-rl1 satellite details 2021-023a norad 47965*, 11 2021, <https://www.n2yo.com/satellite/?s=47965>.
- [21] ESA Space Debris Office, *Esa's annual space environment report*, Tech. report, ESA, 5 2021.
- [22] H Peng and X Bai, *Limits of machine learning approach on improving orbit prediction accuracy using support vector machine*, 1 2017, p. 15.
- [23] Hao Peng and Xiaoli Bai, *Improving orbit prediction accuracy through supervised machine learning*, Advances in Space Research **61** (2018), no. 10, 2628–2646.
- [24] The CubeSat Program, *The cubesat standard*, 11 2021, <https://www.cubesat.org/about>.
- [25] Zachary Reinke, *Training set density estimation for trajectory predictions using artificial neural networks*, (2019).
- [26] SAS, *Aprendizaje automático: Qué es y por qué es importante*, 11 2021, [https://www.sas.com/es\\_es/insights/analytics/machine-learning.html](https://www.sas.com/es_es/insights/analytics/machine-learning.html).
- [27] DATA SCIENCE, *Impulso de gradiente*, 11 2021.
- [28] Matlab Simulink, *Aerospace blockset cubesat simulation library*, 11 2021, <https://es.mathworks.com/matlabcentral/fileexchange/70030-aerospace-blockset-cubesat-simulation-library>.
- [29] MATLAB Simulink, *Matlab: El lenguaje del cálculo técnico*, 11 2021, <https://es.mathworks.com/products/matlab.html>.
- [30] David Vallado and Paul Crawford, *Sgp4 orbit determination*, (2008), 6770.





# Siglas

---

**AI** Artifical Intelligence. 6

**AR** Ascensión Recta. 13

**CSD** CubeSat Design. 2

**DCM** Direct Cosine Matrix. 25

**ECEF** Earth-Centered Earth Fixed coordinates. 22–24, 26–28, 97

**ECI** Earth-Centered Inertial coordinates. 11, 13, 21, 23, 24, 26–29, 97

**GBRT** Gradient Boosting Regression Tree. III, V, IX, X, 37, 39, 51, 59, 68, 70, 71

**GEO** Geosynchronous Earth Orbit. 1

**HEO** High Earth Orbit. 1

**ICRF** International Celestial Reference Frame. 25

**ITRF** International Terrestrial Reference Frame. 25

**LEO** Low Earth Orbit. 1

**MATLAB** MATrix LABoratory. 4, 5, 12, 97

**MEO** Medium Earth Orbit. 1

**MSE** Mean Squared Error. 36

**RBF** Radial Basis Function. 40

**RMSE** Root Mean Squared Error. 35

**RSO** Resident Space Object. 1, 11

**SGP** Simplified General Perturbations. 11

**SVM** Support Vector Machine. 1, 8

**SVR** Support Vector Regression. III, V, X, 8, 9, 39, 40, 47, 69, 97

**TLE** Two-Line Element. 4, 13, 14

**XGBoost** eXtreme Gradient Boosting. III, V, X, 8, 39, 59, 68