

Proyecto de
Simulador de
órbita baja terrestre

David Rodríguez García



Trabajo presentado a la
Escuela Técnica Superior de
Ingenieros Aeronáuticos

perteneciente a la
Universidad Politécnica de Madrid

como parte de la asignatura
Proyecto de Fin de Carrera

incluida en los estudios de
Ingeniero Aeronáutico

Febrero de 2015

Índice general

1. Introducción	6
El problema	6
Objetivos	7
Retos	7
Sobre Python	9
Sobre NumPy	10
Sobre Tkinter	10
Sobre Matplotlib	10
Panorámica del software actual	11
2. La maquinaria	13
Ecuaciones del movimiento	13
Medida del tiempo	14
Hora solar	14
Hora sidérea	14
Tiempo solar	15
Tiempo universal y tiempo universal coordinado	16
Días julianos	16
Sistema de referencia	17
Earth Centered Inertial (ECI)	18
Earth Centered, Earth Fixed (ECEF)	18
Coordenadas cartesianas	20
Coordenadas geográficas y proyecciones	21
Coordenadas celestiales	23
Posición del sol	24
Problema de los dos cuerpos	26
Constantes del movimiento	28
Referencia perifocal	29
Trayectoria elíptica	29
Elementos clásicos de la órbita	32
Maniobras y triedro orbital	33
Cuaternios	36
Definición y operaciones	37
Cuaternios vectoriales	38
Perturbaciones	39
Tierra esférica	40
Campo gravitatorio	42
Cuerpo casi esférico	44

ÍNDICE GENERAL

Armónicos esféricos	47
Atmósfera	47
Velocidad relativa $\tilde{v_w}$	49
Área frontal A_f , coeficiente de resistencia C_D	49
Densidad ρ	50
Perturbaciones lunisolares	51
Presión de radiación	52
Problema perturbado	53
Integración numérica	54
Precisión y estabilidad	56
Runge-Kutta	58
Adams-Bashforth	59
Adams-Moulton	59
Predictor-Corrector	59
3. Manual de usuario	61
Funciones generales	61
Barra de estado	61
Área de visualización	61
Botones de objetos	62
Árbol de objetos	62
Botones de control	62
Barra de herramientas	62
Organizando la simulación	63
Configurando los parámetros	63
Añadiendo elementos	65
Añadir satélites	65
Añadir órbitas	67
Añadir bases	67
Añadir sensores	68
Controlando la simulación	69
Avance fijo	70
Avance continuo	70
Resultados	70
Representación gráfica	71
Acceso a datos instantáneos	73
Acceso a todos los datos	74
Herramientas	75
Asistente de visibilidad	75
Asistente de maniobras	77
4. El entorno	80
Objetivo	80
Consideraciones iniciales	80
El lenguaje	81
Estructura básica	82
Objetos	83
Herramientas	90
Interfaz gráfica	94

ÍNDICE GENERAL

5. Validación	109
Objetivo	109
Métodos de validación	110
Validación del estado inicial	111
Validación de la integración a corto plazo	112
Validación de la integración a largo plazo	113
Validación de las perturbaciones	115
6. Conclusiones	118
Objetivos	118
El camino	119
Aspectos a mejorar	120
A. Datos de validación	129
Datos NASA	130
Posición ISS inicial	130
Datos STK	134
Integración a corto plazo	134
Integración a largo plazo	136
Propagación con J2	137
Datos simulador	140
Integración a corto plazo	140
Adams-Bashforth 4	140
Runge-Kutta 4	142
Predictor corrector	144
Cambio de Δt	146
Integración a largo plazo	150
Propagación con J2	151
Propagación con atmósfera	158
B. Ejemplos de código	159
orbita.py	160
integracion.py	174
cuaternios.py	190
mision.py	193

Tema 1

Introducción

El problema

Llevar a buen término un proyecto de mecánica orbital es uno de los procesos más complejos a los que se puede enfrentar un grupo de ingenieros espaciales. La precisión necesaria en los cálculos, los grandes tiempos involucrados en el proceso y la inmensa cantidad de datos a generar hacen que el proyecto resulte largo y tedioso, y requiera numerosas iteraciones con cálculos y modelos cada vez más refinados. El coste de todos estos sucesivos pasos es cada vez más elevado, por lo que resulta de suma importancia asegurarse de que las decisiones tomadas al principio son adecuadas.

Los comienzos siempre son duros. Al principio del proyecto se tienen muy pocos datos y nociones del camino, y los que se tienen están basados en los juicios y la experiencia de los ingenieros encargados más que en cálculos específicos. Pero, por desgracia, el problema suele ser lo suficientemente complejo como para necesitar un apoyo informático a la hora de realizar los cálculos necesarios para avanzar. Llegado este punto, se suelen tener tres caminos para elegir como continuar

- Usar software propio. Si se reutiliza código de proyectos anteriores es poco probable que esté diseñado para cubrir las variaciones del actual, y crear un programa diferente para cada proyecto es un proceso tedioso y que consume mucho tiempo que podría usarse en otra parte. Esta vía proporciona el mayor control sobre el proyecto, pero aumenta los costes de las críticas primeras etapas.
- Usar software comercial. Existen numerosas licencias de software creado por un equipo especializado que cumplen multitud de tareas necesarias en el proceso. Sin embargo la comercialización de estos paquetes, normalmente a alto precio, hace que de nuevo los costes de las primeras etapas suban, siendo inabarcables normalmente para investigadores, estudiantes o simplemente curiosos. Además, al ser código cerrado, el ingeniero tiene que fiarse de que cumple los requisitos sin poder comprobarlo, y no se pueden introducir modificaciones para adecuarlo y personalizarlo al problema particular.
- Usar software libre. Hay publicados numerosos programas bajo una licen-

cia de software libre, que permite acceso al ingeniero al código abierto del programa. Son altamente personalizables y flexibles, pero se suelen presentar en forma de librerías de funciones, que a su vez el ingeniero tiene que unir e implementar para obtener el problema concreto.

Objetivos

En este proyecto se va a intentar, de alguna manera, unificar estas tres vías en un simulador de órbita baja terrestre, que sea capaz por si sólo de enfrentarse a las primeras etapas de un proceso de análisis de misión LEO con cierta precisión, fiabilidad y flexibilidad, de código abierto que permita a estudiantes e investigadores acceder y complementar sus funciones sin ningún coste. Para ello se pide que el programa cumpla los siguientes requisitos:

- Debe ser sencillo e intuitivo de usar, con una presentación de datos limpia y clara y una interfaz gráfica amigable que incluya todos los controles necesarios.
- Debe abarcar cierto grado de personalización dentro del programa, permitiendo configurar elementos claves dentro de la simulación. Por ejemplo, debe dejar elegir qué perturbaciones se quieren utilizar, o qué datos debe proporcionar.
- Debe ser personalizable en el código, permitiendo añadir modificaciones al código base que permitan al ingeniero usar el programa como base sobre la que construir su problema particular. Deberá ser modular, para que las distintas funciones queden claramente definidas en sus competencias y las modificaciones sean sencillas de implementar.
- Debe proporcionar datos numéricos de fácil acceso de varias variables para su posterior tratamiento, pero también debe tener cierta representación gráfica de los datos más básicos.

Retos

A la hora de afrontar un proyecto de la magnitud de este, sea de la temática que sea, se esperan encontrar numerosos problemas por el camino. Investigar, desarrollar e integrar la cantidad de elementos que forman parte del todo al final del proceso son tareas esenciales, complejas y costosas por derecho propio, incluso conociendo previamente el contexto del problema. De nada sirve toda la capacidad de análisis del mundo si no responde a la pregunta última, o si la pregunta no es la adecuada. Inútil resulta plantearse metas estratosféricas sin la capacidad y los medios para llevarlas a cabo. Y poco valor personal tiene el viaje si resulta rutinario. Encontrar el equilibrio a lo largo de todo el proyecto entre estos grandes frentes resulta fundamental, vital, para la llegada a buen puerto de la nave, sin descontar de la planificación los posibles peligros que se escondan en el océano.

A lo largo de este proyecto en particular se preveen muchas complicaciones que, se espera, se puedan superar y dejar atrás en buenos términos.

- *Implementación* de los conocimientos adquiridos a lo largo de la carrera. Numerosas han sido las asignaturas a lo largo del plan de estudios que han ampliado, creado y respondido a la visión del mundo y las distintas leyes y principios físicos que lo gobiernan y predicen, permitiendo a un ingeniero contribuir a dar minúsculos empujones hacia la mejora de la sociedad. Resulta, por supuesto, imposible retener todos estos conocimientos de manera tangible pero el emprendimiento de un proyecto de estas características permite visitar una vez más las antiguas cavernas y caminos que todo el proceso de adquisición ha dejado. Permite también ejercitarse en nuevas técnicas cuyo ejercicio no se da de manera regular en el entorno del aprendizaje más regulado, y desarrollar nuevas herramientas fundamentales a la hora de afrontar problemas prácticos y concretos, de los que está lleno el mundo. Será un reto rescatar de la memoria (Interna y externa) todos los elementos necesarios, juzgar su aplicabilidad y adecuación y, en último término, llevarlos a cabo dentro de un objetivo común.
- *Equilibrio*. No por pocas razones resulta la desmesura uno de los peores pecados que se pueden cometer. Es fácil caer en la tentación de no conformarse con los resultados obtenidos y querer abarcar y mejorar hasta el límite y después más allá, resultando esta una de las sendas más rápidas hasta el fracaso. Resultaría muy fácil, desde el punto de vista conceptual, implementar en el simulador los modelos más complejos a los que se tenga acceso, con cuidadosos y precisos cálculos de la infinidad de parámetros necesarios y permitir el control absoluto sobre todos y cada uno de estos detalles. Pero entonces se acabaría con una red tan compleja de entender y usar que resultaría absurda, unos resultados tan llenos de ruido externo que resultarían inútiles y, siempre, demasiados pocos recursos. *La virtud está en el término medio*, dijo un griego hace mucho tiempo, y encontrar el punto adecuado entre expectativas excesivas y simpleza desmesurada resulta fundamental. Será un reto lograr el equilibrio entre modelos y procesos suficientemente detallados y lo suficientemente simples para que sean fáciles de encontrar, entender y modificar en caso de que se quieran modificar.
- *Programación*. A lo largo de la carrera de Ingeniería Aeronáutica han sido pocas las oportunidades de desarrollar competencias específicas en un ámbito tan cercano y común hoy en día como son los ordenadores. Las licencias de software adecuadas resultan muy caras y especializadas, por lo que se dejan a menudo para los cursos superiores, y la programación directa ocupa un segundo plano en un ya de por sí apretado y extenuante plan de estudios. Los pocos conocimientos en este ámbito suelen adquirirse de forma lateral por el alumno al realizar trabajos de distintas asignaturas, sin recibir la instrucción formal que a veces sería necesaria. Será un reto concretar todas las leyes físicas en declaraciones de código en un lenguaje de programación que se irá descubriendo más en profundidad según se avance en el proyecto.

Sobre Python

Para llevar a cabo este proyecto se ha decidido utilizar el lenguaje de programación Python. El parecido de muchas funciones básicas a MatLab hace que para el familiarizado con este último, como es mi caso, no resulten tan traumáticos la adopción inicial y los primeros pasos. Pero es mucho más amplio, mucho más flexible y sobre todo mucho más personalizable que Matlab. Es un lenguaje con mucho futuro y capacidades, por lo que aprender su sintaxis y sus técnicas habituales será un gran añadido y un punto importante del proyecto, resultando en una ventaja para mi futura carrera profesional.

Python es un lenguaje de programación nacido a finales de los años 80, con la filosofía de crear un lenguaje flexible, dinámico y altamente legible. Permite la ejecución dinámica del código en su consola o, con la ayuda de algunas herramientas, crear paquetes autoejecutables independientes que pueden ser usados directamente en los sistemas operativos más populares. Su carácter dinámico permite además crear código y ejecutarlo al vuelo instantáneamente, complementándolo con fragmentos ya creados con anterioridad. Está publicado como *software libre*, lo que ha permitido desde su creación el desarrollo por parte de la comunidad de numerosas librerías y paquetes de código (Llamadas *módulos*) que amplían las funciones del código básico. Esta capacidad para descentralizar la creación de un amplio catálogo de librerías especializadas es, sin duda, una de las razones principales del éxito del lenguaje en la actualidad.

El lenguaje integra de buena manera dos filosofías de diseño no siempre compatibles: Orientación a estructuras y orientación a objetos. La primera favorece la creación de una clara jerarquía de operaciones y funciones en rutinas y subrutinas de carácter modular, cada una siendo una pequeña caja negra con pocas salidas y entradas, de manera que se apliquen en orden a un proyecto, intercambiando la información necesaria entre ellas para llevarse a cabo. La segunda apuesta por la creación de grandes paquetes de información en un objeto definido por su *clase*, información a la que después se le aplican distintas funciones llamadas *métodos* que se definen específicamente para cada clase. Esto favorece un flujo de información más del tipo colmena, donde cada método accede a los datos que necesita y realiza las transformaciones que tiene asignadas independientemente del resto de métodos, guardando la nueva información en una dirección de nuevo común. Ambas orientaciones son válidas y pueden ser preferibles según el problema al que se enfrentan, por lo que su integración mutua ha sido otro de los factores que han favorecido el crecimiento del lenguaje.

El lenguaje ha sufrido numerosas actualizaciones desde su creación. La versión que se usa en este proyecto es la 2.7.6, publicada en octubre de 2013, que incluye muchas mejoras actuales sin suponer la parcial ruptura de retrocompatibilidad que supusieron las distintas iteraciones de la versión 3, pudiendo acceder así a un amplio catálogo de módulos que agilizarán el proceso de creación del programa. En los siguientes apartados se incluye una breve descripción de los módulos externos usados, junto a algo de información de sus funciones más básicas. Es **necesario** tener instalada esta versión del lenguaje junto con los distintos módulos descritos para que el programa funcione correctamente.

La sintaxis de Python es limpia y elegante, con un diseño claro, simple, y explícito. Las llamadas de funciones y métodos se realizan simplemente nombrándolos, pudiendo ser flexible en los argumentos aportados. El anidado de estructuras de control se realiza mediante la inserción del código en una zona

indentada con un nivel de sangrado superior, siendo evidente a primera vista la composición del programa. Esta simplicidad está claramente dirigida a hacer el código altamente legible, lo que ayuda a la claridad y personalización que este proyecto pretende implementar.

Sobre NumPy

NumPy es un módulo que extiende las funciones básicas de Python con distintos objetos y funciones matemáticos. Python no está diseñado desde su base para ser un lenguaje de operaciones numéricas, pero sus virtudes pronto atrajeron la atención de la comunidad científica, que vieron en Python una fuerte herramienta para atacar sus distintas necesidades. Varias implementaciones de esta comunidad fueron unificadas cuando en 2006 se lanzó NumPy como parte de un paquete más amplio llamada SciPy, que integra varios paquetes muy útiles para aplicaciones científicas. Cabe destacar el parecido de sus elementos con aquellos encontrados en el popular lenguaje y programa Matlab, lo que facilita la portabilidad de programas en ambos sentidos.

NumPy amplia la librería de funciones con elementos tan básicos como funciones trigonométricas o *arrays* de distintas dimensiones (incluyendo vectores y matrices) y las distintas operaciones matemáticas (como la suma de vectores o el producto de un vector por una matriz) que son necesarias para aplicarlos a las distintas ramas científicas, incluyendo la mecánica, optimizados específicamente para esta clase de tratamiento.

Sobre Tkinter

Tkinter es un módulo que viene incluido en la instalación estándar de Python, y que permite la creación y manejo de distintas interfaces gráficas que a su vez ordenen y faciliten el uso del código que se pretende usar. Permite la implementación de distintos objetos llamados *wIDGETS*, y su organización en ventanas. El rango de funciones que cumplen los widgets es muy amplio, desde simples etiquetas donde mostrar información a botones que ejecuten funciones, y las ventanas permiten organizar estos widgets en estructuras complejas y ordenadas con relativa facilidad.

Sobre Matplotlib

Matplotlib es una librería que permite generar una gran cantidad de gráficos de manera sencilla a partir de datos numéricos. Aunque está plenamente diseñada e integrada en Python, presenta, de nuevo, numerosas similitudes con el lenguaje de Matlab, haciendo la portabilidad entre los dos programas muy sencilla. Su versatilidad, sencillez y funciones han conseguido que se convierta en casi un estándar dentro de la comunidad, y se distribuye dentro de numerosos paquetes conocidos como SciPy o PyLab, además de individualmente. Tiene algunos packs de herramientas muy útiles además del básico, como la posibilidad de hacer representaciones en 3 dimensiones o la capacidad de generar distintos mapas personalizables. Por desgracia, es una herramienta pensada para la creación de dibujos estáticos, por lo que las aplicaciones que se hacen en este proyecto de representación en tiempo real funcionan con lentitud.

Panorámica del software actual

El desarrollo de paquetes de software de simulación orbital es un campo de mucho interés para los ingenieros. Ser capaz de predecir con exactitud, consistencia y fiabilidad el comportamiento y evolución de un satélite en sus órbitas alrededor de la tierra resulta fundamental incluso para las misiones más básicas. Las misiones espaciales son especialmente caras, por lo que todo análisis previo que ahorre tiempo y dinero resulta muy útil a la hora de afrontarlas. La simulación y la explotación de misiones reales son dos procesos simbióticos, donde los datos recogidos de misiones reales permiten elaborar modelos más precisos, que a su vez permiten diseñar misiones más ambiciosas y baratas. Por ello, de la mano del desarrollo espacial, ha ido siempre el desarrollo de software de simulación y de análisis de misión. Vamos a enumerar y describir brevemente algunos de los programas más famosos en la actualidad que cumplen esta tarea.

- Systems Tool Kit (STK) de Analytical Graphichs, Inc. (AGI). Quizá el programa más completo y desarrollado de la actualidad, y sin duda uno de los más usados a través de la industria. Permite hacer todo tipo de simulaciones orbitales, de comunicaciones, visibilidad y cobertura, e incluye también aplicaciones terrestres de guiado y control. Es un paquete comercial y aunque haya disponible una opción gratuita, esta está muy limitada en sus funciones. El programa se usa mediante una interfaz gráfica convencional.
- FreeFlyer, desarrollado por a.i. solutions, incluye muchas de las funcionalidades más útiles y necesarias para una misión de mecánica orbital, divididas en varios escalones: Propagación de órbitas, actitud, cobertura, maniobras, etc. También es un paquete comercial, pero carece de versiones de prueba o gratuitas disponibles para el público. Se usa también mediante una interfaz gráfica.
- General Mission Analysis Tool (GMAT), desarrollado por la NASA y distribuido como software libre, presenta herramientas para la simulación orbital y el desarrollo de misiones. Sin embargo, se centra principalmente en misiones interplanetarias y carece de herramientas de análisis esenciales para misiones cercanas a la tierra.
- Java Astrodynamics Toolkit (JAT), desarrollado y distribuido por la Universidad de Texas, Austin como software libre. No es un programa en sí, sino una librería de funciones que incluyen cálculos de propagación de órbitas, actitud, perturbaciones, etc. Es una librería muy completa y con mucho apoyo y desarrollo, pero su carencia de interfaz gráfica exige que el usuario tenga conocimientos de programación, tiempo y recursos con los que implementar sus funciones.
- Kerbal Space Program, desarrollado por SQUAD. No compite ni pretende hacerlo con el resto de programas de esta lista, y su aspecto de simulación queda muy velado por numerosas simplificaciones, falta de rigurosidad o simples carencias. Sin embargo, resulta muy útil en otro campo en que ninguno de los otros programas puede competir: La divulgación. Mediante una interfaz muy sencilla y unas mecánicas divertidas, permite al usuario no iniciado desarrollar conocimientos e intuición de mecánica orbital y

aprender sobre la materia y las misiones espaciales. Se pueden encontrar además modificaciones que añaden algo de realismo a la simulación. Es un paquete comercial, pero muy barato y asequible al gran público.

Estos son algunos de los programas más usados y conocidos en la actualidad, y la base del enfoque que se da a este proyecto. Partiendo de la existencia de todos estos paquetes, se intenta encontrar un punto medio: Un simulador completo y riguroso, a la vez que mantiene cierto grado de personalización y sin comprometer la facilidad de uso para el usuario final.

Tema 2

La maquinaria

En este capítulo se pretende dar a conocer toda la física involucrada en el problema de mecánica orbital, todos los aspectos que hay que tener en cuenta a la hora de abordar su solución, así como los distintos modelos que se han usado para simular la realidad en el entorno del programa. Se ha intentado mantener las explicaciones básicas, cortando los desarrollos en la medida de lo posible para que se pueda seguir la explicación, y haciendo hincapié más en la solución y su aplicabilidad que en el desarrollo. Si el lector quiere conocer más a fondo todas las explicaciones, en la bibliografía del final se citan algunos textos que puedan ayudar a esta tarea.

Ecuaciones del movimiento

El punto de partida de todo problema de dinámica es la conocida segunda ley de Newton: *El cambio de movimiento es proporcional a la fuerza impresa, y ocurre según la linea recta a lo largo de la cual aquella fuerza se imprime*. A la constante de proporcionalidad se le denomina *masa inercial*, y la ley escrita en términos matemáticos resulta ser

$$\vec{F} = m \cdot \vec{a} \quad (2.1)$$

o, descomponiendo los vectores en sus componentes rectangulares

$$\begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix} = m \cdot \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} \quad (2.2)$$

donde \vec{F} es la resultante de todas las fuerzas que actúan sobre la masa m , y \vec{a} es la aceleración, la derivada segunda de la posición de m , medida respecto a un sistema de referencia inercial. La solución del problema será tan sencilla o complicada como lo sea la formulación de los integrantes de esta ecuación. Para la aplicación que incumbe a este programa, aunque existen soluciones analíticas para ciertos casos, la existencia de perturbaciones justifica la integración numérica que se hace de 2.1

Medida del tiempo

Medir el paso del tiempo puede parecer sencillo, incluso trivial. Con un reloj lo suficientemente preciso, se puede determinar con toda seguridad cuantos segundos han transcurrido entre dos sucesos. Cuando alguien da la hora, está estableciendo el intervalo transcurrido desde la última medianoche. Sin embargo, para determinadas aplicaciones, esta manera de medir el tiempo no es suficiente. ¿La medianoche local o la de Greenwich? ¿Dónde estará la tierra dentro de exactamente 56 días 4 horas y 37 segundos? Aunque es posible mantener la coherencia temporal interna de un sistema sin necesidad de más artificios, a la hora de relacionarlo con su entorno (en nuestro caso la tierra, el sol y las estrellas) hace falta algún mecanismo más refinado, con unos puntos de referencia definidos y claros. Para ello, se definen los siguientes conceptos.

Hora solar

Debido a la naturaleza más o menos estable del movimiento aparente del sol alrededor de la tierra, el ser humano ha usado constantemente este astro como patrón fijo con el que medir y poner orden al caos de la vida terrestre. Se define el día solar como *El intervalo de tiempo entre dos pasos sucesivos del disco solar por el meridiano del observador*. A este paso se le denomina mediodía y, por comodidad, se fija la hora del suceso a las 12:00. La hora solar local se definirá entonces como el ángulo girado por la tierra desde el paso del sol por el meridiano local. Esta medida del tiempo, aunque muy intuitiva y sencilla de llevar, no tiene la precisión que requerimos. El día solar no es constante debido principalmente a la excentricidad de la órbita de la tierra, la inclinación del eje de giro y diversas irregularidades en su movimiento de rotación. Es necesario definir entonces el día solar medio y la hora solar media de Greenwich (GMT), como el ángulo que forma un sol ficticio en su giro uniforme alrededor de la tierra. Este punto de referencia depende de la posición relativa entre el sol y la tierra y por tanto cambia a lo largo del año, por lo que hace falta un sistema de referencia más *fijo*.

Hora sidérea

Se define de manera parecida a la hora solar, pero con el paso de un astro determinado, mucho más lejano que el sol y, por consiguiente, con menor variación de su posición a lo largo del año. Se elige como referencia el equinoccio vernal, la intersección de la eclíptica con el ecuador terrestre. Esta medida resulta muy conveniente para observaciones astronómicas pues permite referenciar fácilmente la posición de un astro determinado, pero poco adecuada para el propósito que buscamos, ya que ambas referencias se mueven con el tiempo.

El día solar es más largo que el denominado día sidéreo debido al movimiento de traslación de la tierra a lo largo de su órbita. En efecto, en el movimiento de rotación de la tierra, el día sidéreo se define como el intervalo de tiempo entre dos pasos de un astro lejano sobre el meridiano local. Debido a la gran distancia que nos separa de cualquier otro astro que no sea el sol, puede considerarse que la estrella permanece fija y por tanto la dirección en el espacio es la misma, equivaliendo el día sidéreo a un giro de exactamente 360 grados de la tierra sobre su eje. Sin embargo, la distancia al sol es mucho más pequeña y los efectos

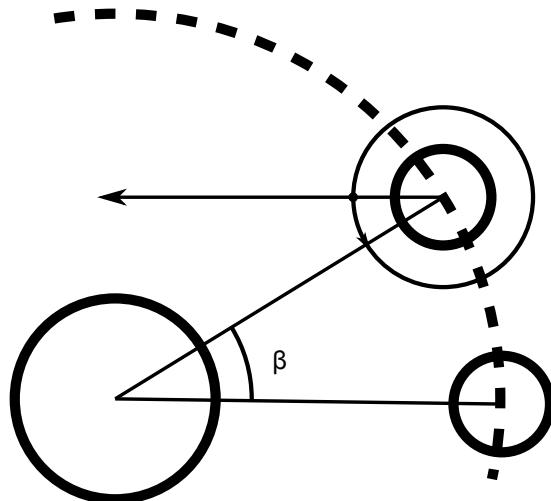


Figura 2.1: Días solar y sidéreo

del movimiento de la tierra son apreciables en el transcurso de un día. La tierra tiene que girar un ángulo extra, β , para volver a tener el sol sobre el meridiano local. Como el ángulo β es aproximadamente de un grado, el día solar resulta ser unos 4 minutos más largo que el día sidéreo.

Tiempo solar

La tierra es muy grande y está habitada por mucha gente, cada uno con gustos distintos. Podría dejarse al libre albedrío de cada uno elegir su propia referencia a la hora de medir el tiempo, su propio astro y distribución horaria, aquella que le conviniese o simplemente gustase más, pero entonces concertar citas y acordar horarios resultaría harto difícil. Por ello, es necesaria la presencia de un patrón global, que resulte conveniente a todo el mundo y que sea compartido por todos, resultando claro e inequívoco el momento al que nos referimos.

Este papel lo cumple el tiempo medio de Greenwich o GMT. Se define la hora GMT como el ángulo (medido en horas) que forma el meridiano de Greenwich con el sol medio, añadiendo o quitando doce horas según sea necesario. Así, un día civil comienza cuando el sol medio atraviesa la vertical del meridiano opuesto al de Greenwich, momento que se define como las 00:00:00. A partir de ahí, la tierra continua su movimiento de rotación y el tiempo solar aumenta hasta que se alcanzan las 12:00:00, momento en el cual el sol medio atraviesa el meridiano de Greenwich.

Esta hora solar resulta muy adecuada para la zona que se encuentra cercana al meridiano. Sin embargo, si otros lugares de la tierra siguiesen este patrón habría zonas donde el mediodía se correspondería con las 3 de la mañana, o la medianochе con las 5 de la tarde. Por ello, a la hora de definir la hora local, se suma a la central GMT la longitud del punto de observación, permitiendo así a todos los lugares de la tierra tener, además de la misma referencia, más o menos las mismas horas locales para el movimiento del sol. Esta manera de medir el tiempo se denomina tiempo zonal.

$$GMT = ZT \pm longitud$$

La longitud se suma o resta convenientemente según sea este u oeste.

Por cuestiones sociopolíticas, resulta conveniente agrupar ciertas regiones en grandes grupos y asignarles un mismo tiempo zonal. Para algunas personas en los extremos oriental y occidental de dichas zonas el sol irá ligeramente retrasado y para otras ligeramente adelantado, pero para la gran mayoría será suficiente para, sin perder mucha precisión, coordinarse mejor con sus alrededores. Estas zonas se denominan *husos horarios*, y se definen un total de 24 a lo ancho del globo, separados por 15 grados. En cada una se usa el tiempo solar medio del meridiano central, obteniendo así cambios entre las horas zonales de un número entero de horas. Esto permite definir la hora local de, por ejemplo, Europa central, como GMT +1, indicando que la hora zonal se obtiene a partir de GMT añadiendo una hora. Hacia el oeste se encuentran las zonas -1,-2,... y hacia el este las positivas, hasta que ambas se encuentran en el meridiano opuesto a Greenwich. El paso de la zona -12 a la +12 se define como la línea de cambio de fecha, y atravesarla supone cambiar la hora local un día completo, 24 horas.

Tiempo universal y tiempo universal coordinado

Se define una escala de tiempos basada en GMT, con distintas correcciones en un intento de independizarlo de los movimientos de la tierra. Así se obtienen una serie de correcciones llamadas UT0, UT1 y UT2, que usan distintos observatorios con el objetivo de universalizar sus distintas mediciones locales y que tienen en cuenta varias imperfecciones en el giro de la tierra, como algunas periódicas debidas a las mareas o variaciones estacionales, a la hora de definir con exactitud la hora local.

La llegada de los relojes atómicos propició una medida del tiempo mucho más exacta, que dio lugar al tiempo universal coordinado (UTC), que añade a las correcciones anteriores determinados *segundos de salto*, corrigiendo convenientemente la escala de tiempos para incorporar irregularidades en el ratio de giro de la tierra. Estos *segundos de salto* son calculados y publicados por el *International Earth Rotation and Reference Systems Service*, y son el punto de partida para la medición del tiempo en el programa.

$$UTC = UT1 + \Delta UT$$

En la actualidad, estos segundos de salto se han acumulado hasta suponer aproximadamente 35 segundos. Todos han sido hasta la fecha positivos indicando que el movimiento de rotación de la tierra se va frenando poco a poco. Este año, el 30 de Junio, está prevista la introducción de una nueva corrección.

Días julianos

El cálculo del tiempo transcurrido entre dos sucesos cercanos es relativamente fácil de realizar. Sin embargo, el intervalo entre dos fechas distantes se complica con numerosos años bisiestos y meses de duración distinta, por lo que es necesaria la introducción de una herramienta, llamada días julianos, que facilite este cálculo.

Esta técnica permite asignar a cada fecha un único valor real, inequívoco que representa el número de días transcurridos desde una fecha inicial que sirve de patrón. Así, si se conocen las fechas julianas de dos días distintos, por muy separados en el tiempo que estén, puede determinarse mediante una simple resta el número de días que han transcurrido entre ellos, sin necesidad de contar las irregularidades del calendario civil. Esta técnica es muy útil para aplicaciones astronómicas, pues las posiciones de los distintos astros van siempre acompañadas de su *época*, la fecha en la que se tomaron y a partir de la cual se calcula la posición en cualquier otro instante de tiempo. El programa usa esta técnica para poder calcular la posición del sol y de la tierra en todo instante, con similitud a las posiciones reales en dicho instante.

Se divide el tiempo en períodos de 7980 años, llamados eras julianas, y se sitúa el comienzo de la primera era en el 1 de Enero del 4713 a.C. a las 12:00. A partir de esta fecha, se cuentan los días transcurridos (Incluida la fracción del día), que determina inambiguamente la fecha juliana. Conociendo pues las fechas julianas de dos sucesos distintos, se obtiene fácilmente el tiempo transcurrido entre ellas.

Aunque resulte tentador contar uno a uno los más de 2 millones de días transcurridos desde esta fecha, hay algoritmos ya probados que realizan el cálculo mucho más sencillamente, como el publicado por el *United States Naval Observatory*¹ o, el usado en el programa, el publicado por el *National Renewable Energy Laboratory*²

$$JD = [365,25 \cdot (año + 4716)] + [30,6001 \cdot (mes + 1)] \\ + dia + B - 1524,5 + hora/24 + minutos/1440 + segundos/86400 \quad (2.3)$$

$$B = \begin{cases} 0 & \text{para } JD < 2299160 \\ 2 - \lfloor \frac{año}{100} \rfloor + \left\lfloor \frac{\lfloor \frac{año}{100} \rfloor}{4} \right\rfloor & \text{para } JD > 2299160 \end{cases}$$

donde los corchetes $\lfloor \rfloor$ indican producto entero, la parte entera de la división.

Sistema de referencia

La ecuación 2.1 sin modificar es válida sólo para un sistema de referencia inercial, siendo necesario en caso contrario incluir las conocidas fuerzas de inercia, fuerzas ficticias que simulan el movimiento acelerado de la referencia. En principio, cualquiera de los dos caminos resulta igualmente válido para la determinación de la solución. Sin embargo, el cálculo de las fuerzas de inercia requiere conocer con mucha precisión el movimiento absoluto del sistema de referencia, cosa que no siempre resulta fácil o barata de calcular. Además, las fuerzas de inercia suelen ser de distinto orden de magnitud que el resto de las fuerzas presentes en el problema, por lo que su acoplamiento a la integración numérica puede resultar en errores importantes. Por todo esto, se prefiere la resolución en un sistema de ejes iniciales.

¹Puede encontrarse en http://aa.usno.navy.mil/faq/docs/JD_Formula.php

²Puede encontrarse en <http://www.nrel.gov/docs/fy08osti/34302.pdf>

Encontrar un sistema así en un universo en constante movimiento resulta poco menos que imposible, por lo que hay que conformarse con una aproximación. Se han definido muchos sistemas inerciales, cada uno con distintos elementos y distintas aplicaciones, resultando su elección importante y dependiente de la misión que se le vaya a dar. Un sistema de ejes ligados a un sólido, muy conveniente para describir su actitud alrededor de su centro de masas, resultará muy inapropiado para describir su movimiento a través del sistema solar.

Como el programa está dedicado a calcular y describir órbitas cercanas a la tierra, resulta natural y conveniente usar un sistema ligado a ella. Para ello, se usan dos sistemas de referencia distintos.

Earth Centered Inertial (ECI)

Se define un sistema inercial, centrado en el centro de masas de la tierra, con los ejes apuntando a direcciones fijas. Se elige el eje z de manera que apunte en la dirección del polo celeste, la dirección de la velocidad angular de la tierra. A continuación, se elige el eje x de manera que apunte a otra dirección fija, en este caso el equinoccio vernal. Por último, el eje y se elige de manera que forme un triángulo ortogonal a derechas con los otros dos. Como ya se vio anteriormente, tanto el ecuador terrestre como la eclíptica solar no son referencias fijas, por lo que a la definición anterior hay que añadirle la especificación del momento que se usa para calcular estas direcciones. A este momento se le denomina **época**, y su definición clara resulta tan fundamental como el resto de elementos del sistema de referencia.

A lo largo de los años de estudio celeste, se han definido muchas épocas, según resultaba más conveniente con los datos obtenidos. La fecha más usada en la actualidad es el 1 de Enero del año 2000 a las 12:00. Esta época, junto con la definición de ejes anterior, da lugar al sistema *J2000* o *EME2000*, que tomaremos como base para los cálculos en el programa.

Nótese que al tomar este sistema de referencia como inercial, estamos despreciando los efectos de la aceleración de la tierra en su viaje alrededor del sol. Resulta conveniente esta simplificación para evitar añadir complejidad adicional y errores numéricos al problema, ya que las fuerzas de inercia derivadas de este hecho son varios órdenes de magnitud inferiores al resto de fuerzas implicadas.

Earth Centered, Earth Fixed (ECEF)

El uso de un sistema de referencia ECI resulta muy conveniente para los distintos cálculos matemáticos del problema, pero su traducción a términos cotidianos puede resultar engorrosa. Por ello, se define otro sistema de ejes, esta vez ligados a la tierra. Con centro en el centro de masas de la tierra, eje z apuntando al polo (Más concretamente, a un polo ficticio llamado International Reference Pole), eje x apuntando al meridiano de Greenwich y eje y formando un triángulo a derechas, se consigue geolocalizar mucho más fácilmente la posición de un satélite. Por ello, en la representación 2D sobre el mapa global, el programa usa este sistema, considerando a la tierra fija.

Cabe mencionar que el polo terrestre no coincide con el eje de rotación instantáneo de la tierra, ya que el eje de giro está afectado por distintos movimientos de nutación y precesión. Sin embargo, debido a que estos movimientos son pequeños y a que el programa usa este sistema de referencia para calcular

una representación y no para realizar cálculos físicos, se desprecian estos dos movimientos y se asume que ambos ejes si coinciden, por lo que se simplifica enormemente el cambio entre los dos sistemas.

Una buena aproximación para la transformación necesaria para cambiar coordenadas de un sistema ECI a un ECEF viene dada por el *United States Naval Observatory* (USNO), que ofrece un algoritmo sencillo y fiable para calcular el giro propio de la tierra. El algoritmo permite calcular el ángulo hora que forma el meridiano de Greenwich con la dirección del equinoccio vernal, y por tanto implementar el giro necesario para la transformación entre los dos sistemas. El algoritmo³ puede describirse como:

1. Cálculo de la fecha juliana y la fecha juliana modificada. Se define JD como la fecha juliana del momento en que se quiere calcular la posición de la tierra, y JD_0 como la fecha juliana de la última medianoche (00h), de manera que

$$JD = JD_0 + H/24$$

donde H son las horas transcurridas desde la última medianoche, con las posiciones decimales necesarias. Nótese que, con la definición de fecha juliana que se ha dado anteriormente, cualquier medianoche de cualquier día acaba exactamente en .5.

2. Cálculo de los días transcurridos desde las 12 del mediodía del 1 de Enero del 2000

$$D = JD - 2451545,0$$

$$D_0 = JD_0 - 2451545,0$$

3. Cálculo del tiempo sidéreo medio de Greenwich, en horas

$$\begin{aligned} GMST &= 6,697374558 + 0,06570982441908 \cdot D_0 + \\ &\quad + 1,00273790935 \cdot H + 0,000026 \cdot T^2 \end{aligned}$$

donde en término T corresponde al número de siglos transcurridos desde el año 2000, pudiéndose expresar como

$$T = \left\lfloor \frac{D}{36525} \right\rfloor$$

y siendo nulo para la mayoría de aplicaciones actuales, cercanas en el tiempo.

El algoritmo también describe métodos para introducir correcciones originadas por la nutación de la tierra, pero como ya se ha dicho anteriormente, su efecto es pequeño y se desprecia en aras de simplificar el problema.

³Puede encontrarse íntegramente en <http://aa.usno.navy.mil/faq/docs/GAST.php>

Coordenadas cartesianas

A la hora de describir una posición, un movimiento, son necesarios varios aspectos. En los puntos anteriores, se ha descrito *respecto a qué* describimos el movimiento, qué punto de referencia tomamos. Sin embargo, también es necesario responder a la pregunta de *cómo* describimos la trayectoria. Por ejemplo, en un mapa se puede describir un punto como 100 kilómetros al sur y 100 kilómetros al oeste, o se puede decir que está a 141 kilómetros en dirección suroeste. De esta descripción es de la que se encarga el sistema de coordenadas.

Quizá uno de los sistemas más intuitivos a la hora de describir un movimiento sean las coordenadas cartesianas. Si se tienen tres ejes perpendiculares entre sí, la posición de un punto P puede describirse mediante sus proyecciones en cada uno de los ejes. Trazando un plano perpendicular a cada eje y que pase por P, la intersección del plano y el eje dará el valor de esa coordenada, y repitiendo el proceso 3 veces obtendremos las tres coordenadas necesarias para describir un punto en el espacio.

Las coordenadas cartesianas presentan una clara ventaja operativa frente a otros sistemas de coordenadas. En algunas ocasiones puede descomponerse el movimiento en tres movimientos diferentes, uno en cada eje, y en algunos casos completamente independientes, lo que permite técnicas de separación de variable que simplifican los problemas. En otras la estructura geométrica del problema favorece la incorporación de este tipo de coordenadas. Pero sin duda la razón más importante, y aquella por la que se introducen en el programa, son los factores de escala.

Los factores de escala tienen que ver con la geometría del sistema de coordenadas, y se obtienen estudiando la evolución de la posición frente a variaciones unitarias en cada una de las coordenadas. Para un sistema de coordenadas q

$$e_i = \frac{\partial \mathbf{x}}{\partial q_i}$$

son los vectores directores. Por supuesto, si se ha elegido un sistema de coordenadas ortogonales entre sí, se cumple además

$$e_i \cdot e_j = 0 \quad \text{para } i \neq j$$

y si normalizamos estos vectores

$$\hat{e}_i = \frac{e_i}{|e_i|} = \frac{e_i}{h_i}$$

A los distintos h_i se les conoce como factores de escala, y su cálculo resulta fundamental antes de poder realizar alguna operación en el sistema de coordenadas elegido. Por ejemplo, una diferenciación simple

$$d\mathbf{x} = \sum \frac{\partial \mathbf{x}}{\partial q_i} dq_i = \sum e_i dq_i = \sum \hat{e}_i h_i dq_i$$

La ventaja de usar unas coordenadas cartesianas radica en que todos los factores de escala son unitarios, por lo que los procesos de derivación, diferenciación e integración se simplifican enormemente.

$$dS_x = \frac{\partial \mathbf{x}}{\partial x} = dx \cdot u_x; \quad h_x = 1$$

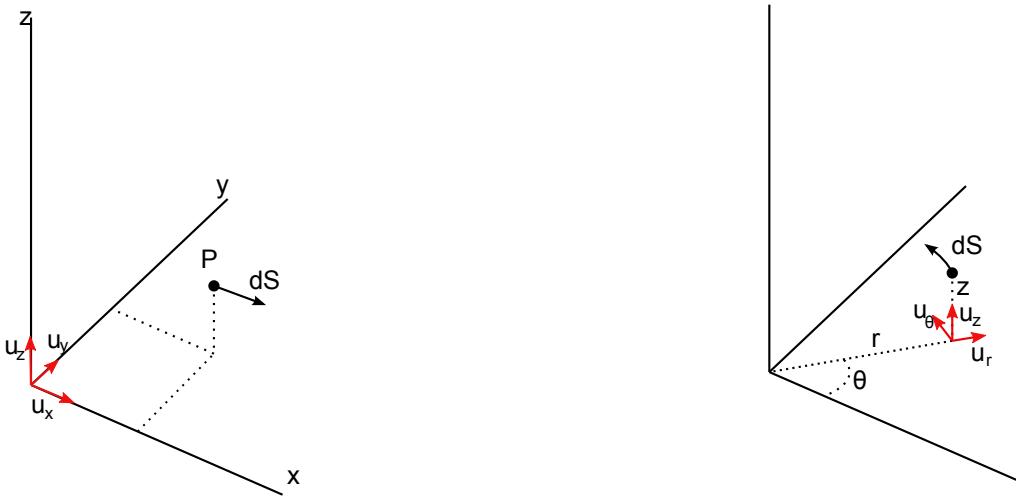


Figura 2.2: Coordenadas cartesianas frente a cilíndricas

y sin embargo, en sistema de coordenadas cilíndricas $q = (r, \theta, h)$

$$dS_\theta = \frac{\partial \mathbf{x}}{\partial \theta} = r \cdot d\theta \cdot u_\theta; \quad h_\theta = r$$

En el programa se usa un sistema de coordenadas cartesianas para representar las posiciones y velocidades de los distintos objetos, y para realizar las distintas operaciones a las que se someten.

Coordenadas geográficas y proyecciones

Desde hace mucho tiempo, la humanidad se ha servido de una herramienta para representar las tierras en las que vivía, para trazar rutas comerciales o para marcar puntos importantes geográficos: La cartografía. El dibujado de mapas ha sido un aspecto fundamental del desarrollo tecnológico y cultural de la humanidad, y la necesidad de llenar los espacios en blanco ha movido a muchos exploradores a arriesgar sus vidas por el conocimiento.

El principio de un mapa es sencillo. Se referencia la posición de un lugar respecto a otros, de manera que se pueda *navegar* fácilmente entre ellos. Se mencionan los puntos de interés, o los detalles característicos, de manera que el lector del mapa pueda reconocerlos fácilmente, conocer su posición y tomar decisiones al respecto. Sin embargo, a medida que el conocimiento de la humanidad sobre la tierra crece, es necesario representar puntos cada vez más alejados, y es necesaria una aproximación más exacta que un mapa plano.

La tierra tiene forma esférica (en realidad, una forma mucho más complicada que se describirá más adelante) y por tanto su representación en un plano es compleja y presenta un problema. Para solucionarlo, se cuenta con varias herramientas. Una de ellas es la representación de la tierra como lo que es, una esfera. Se descarta la concepción de mapa plano, y en su lugar se construye un mapa esférico, sobre el que se dibujan los elementos necesarios. Son los conocidos globos terráqueos que, aunque presentan ciertas ventajas, son muy complicados de usar con precisión y suelen ofrecer poca información.

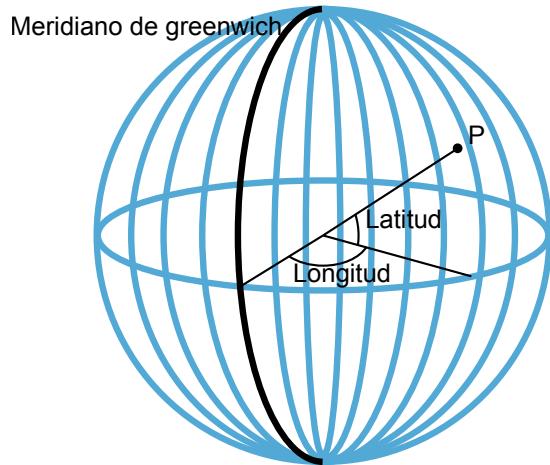


Figura 2.3: Coordenadas geográficas

Con el desarrollo de la tecnología informática, Internet y los sistemas de posicionamiento global ha venido la implementación de herramientas muy potentes, que permiten no sólo ver el globo en su conjunto, sino también mapas globales, trazado de rutas e incluso visualización de imágenes *a pie de calle*, que facilitan mucho la navegación.

La última técnica, quizás la más tradicional, es la proyección de la tierra sobre un mapa bidimensional. Existen varias proyecciones, como la azimutal, que muestra la vista desde un punto sobre una zona específica, o la pseudocilíndrica, que intenta desarrollar la esfera en una forma bidimensional compleja, cada una con sus ventajas e inconvenientes. Sin embargo, la proyección más extendida, y a la que más gente está acostumbrada es la proyección cilíndrica.

Se definen dos ángulos. La latitud es el ángulo que forma el radiovector que une el centro de la tierra y el punto que se quiere representar con su proyección sobre el plano del ecuador. La longitud, por otro lado, es el ángulo que forma dicha proyección con una dirección fija (a la tierra), por convenio el meridiano de Greenwich. Mediante este par de valores, se puede referenciar cualquier punto sobre la superficie de la tierra.

Este par de valores (lat , lon) ahora puede representarse en un plano (y , x), de manera que puntos con la misma latitud estén en la misma coordenada y , y puntos con la misma longitud tengan el mismo valor de x . Si los intervalos de latitud y longitud se reparten uniformemente sobre el plano, obtenemos la proyección más habitual, la proyección cilíndrica recta. Si definimos el meridiano de Greenwich para que tenga longitud 0, puntos hacia el este tendrán valores de longitud positivos y puntos hacia el oeste valores negativos. Igualmente, puntos en el hemisferio norte tendrán valores de latitud positivos y puntos del hemisferio sur negativos.

Esta proyección, a cambio de ofrecer de un vistazo rápido una vista general de toda la tierra de manera sencilla, tiene un gran inconveniente. Las zonas cercanas a los polos quedan muy distorsionadas y sobredimensionadas en relación con aquellas más ecuatoriales, y las trayectorias que en el espacio son líneas rectas y circunferencias pierden su forma y pasan a ser curvas más complejas, como se

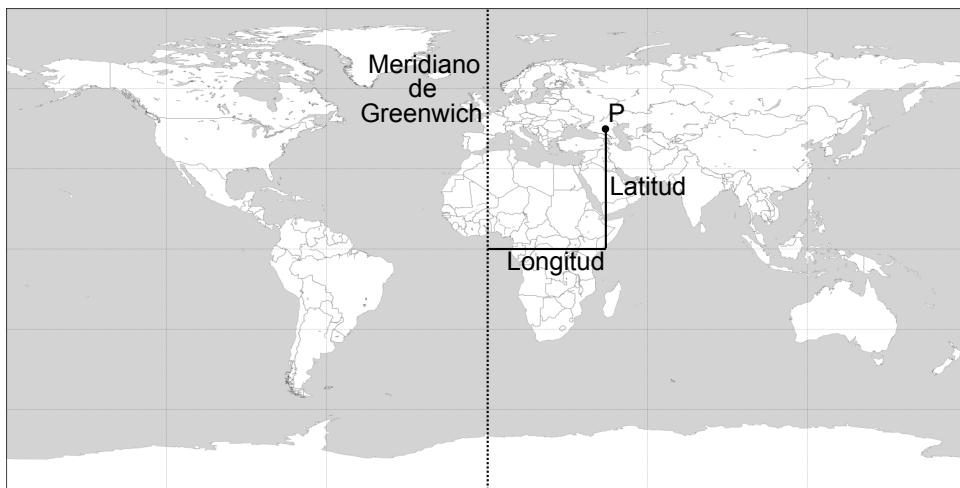


Figura 2.4: Coordenadas geográficas sobre el mapa

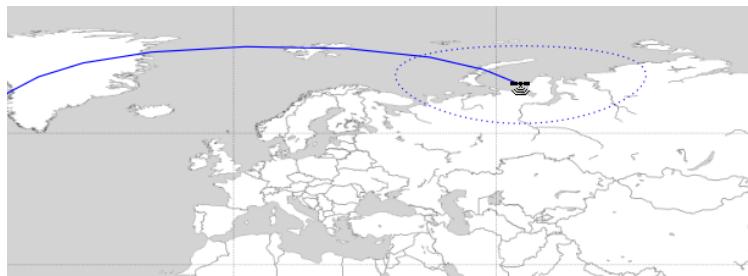


Figura 2.5: Distorsión de la proyección cilíndrica

puede ver en la figura 2.5

El programa usa esta proyección para representar las trazas de las trayectorias de los distintos satélites sobre la superficie de la tierra. Como ya se verá más adelante, la introducción de la consideración de la tierra como no esférica lleva a un cálculo mucho más complicado de las coordenadas geográficas que el que se ha referido aquí. Pueden verse en detalle los cálculos y ecuaciones implicadas en este proceso en el capítulo de perturbaciones.

Coordenadas celestiales

Aunque las coordenadas cartesianas faciliten mucho las operaciones matemáticas internas, no siempre resultan las más adecuadas a la hora de afrontar un problema. Sus expresiones normalmente resultan farragosas y poco aparentes, poco intuitivas para su uso manual e incómodas de usar. También puede haber problemas con cierta simetría, que se beneficien de usar un sistema de coordenadas que comparta esa simetría.

En astronomía hay un sistema de coordenadas que presenta evidentes beneficios a la hora de describir la posición de un astro. Los diferentes cuerpos celestiales están tan lejos de la tierra que su expresión en un sistema de coordenadas rectangulares resulta impracticable. En su lugar, se usa un sistema de

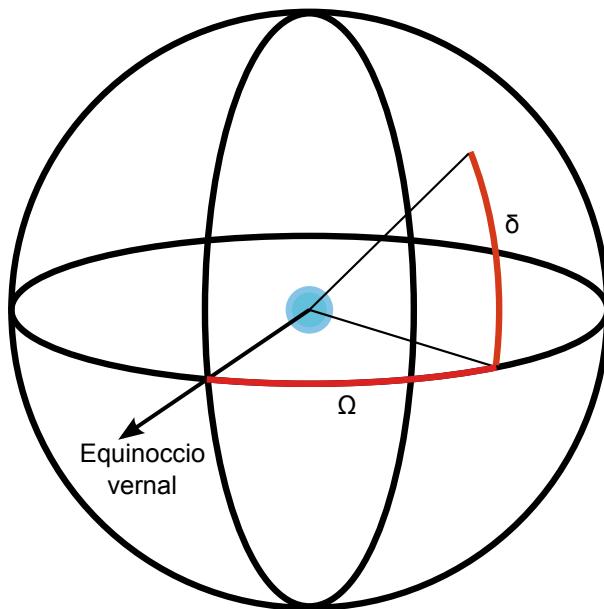


Figura 2.6: Coordenadas celestiales

coordenadas esféricas llamado sistema de coordenadas celestiales.

Se imagina una esfera, llamada esfera celeste, de radio arbitrario pero lo suficientemente grande y centrada en la tierra, sobre la que se proyectan todos los cuerpos celestes. Se define un plano, llamado plano fundamental, resultado de proyectar el ecuador terrestre sobre la esfera celeste. Ahora, la posición de una estrella u otro cuerpo celeste puede describirse por su distancia a la tierra, y dos ángulos distintos: El ángulo que forma el radiovector de la estrella con su proyección recta sobre el plano fundamental, llamado declinación, y el ángulo que forma dicha proyección con una dirección fija, normalmente el equinoccio vernal, llamado ascensión recta. Si el radio de la esfera es suficientemente grande, puede descartarse la distancia y usar simplemente el par de ángulos (Ω, δ)

Usar este par de valores introduce mucha simplicidad la hora de representar y rastrear los distintos cuerpos celestes, y ofrece una conversión rápida y sencilla para que los distintos observatorios locales obtengan fácilmente los distintos pasos de los astros.

Posición del sol

Hace mucho tiempo que se abandonó la concepción geocentrista del universo, de que todos los astros se mueven a nuestro alrededor, para cambiarla por otra mucho más realista, describiendo nuestra posición como parte de un sistema solar en continuo movimiento y una galaxia mucho más grande que nosotros, lo que sin duda ha propiciado el estudio del universo y el desarrollo tecnológico del que disfrutamos hoy en día.

El movimiento más evidente de la tierra es el de traslación alrededor del sol. Sigue una órbita casi circular, de muy baja excentricidad y muy baja inclinación (respecto al ecuador solar) y la combinación de este movimiento con la

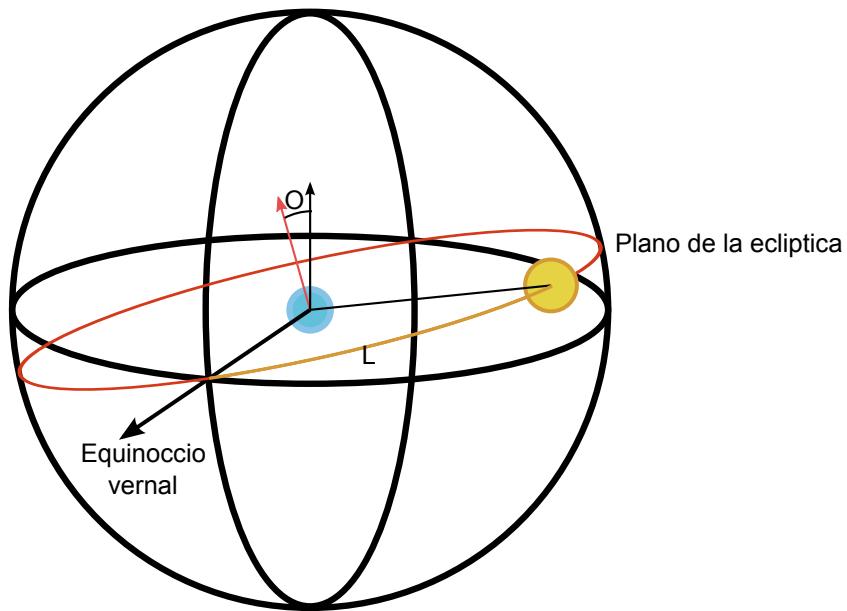


Figura 2.7: Coordenadas eclípticas

inclinación del eje de la tierra origina el fenómeno de las estaciones, que marca y regula la vida en el planeta.

Sin embargo, en ocasiones resulta más provechoso expresar el movimiento inverso. Si nos fijamos a la tierra, el sol tiene un movimiento aparente a su alrededor. Y, para aplicaciones cercanas a la tierra, puede resultar ventajoso describir este movimiento en lugar del inverso, el real, de manera que los cálculos y conversiones se simplifiquen.

Para ello, se usa el llamado sistema de referencia de la eclíptica. Se llama eclíptica al plano de la órbita que sigue la tierra alrededor del sol, y por tanto es un plano que se puede proyectar sobre la esfera celeste. Debido a la inclinación del eje de rotación de la tierra, será un plano inclinado respecto al ecuador celestial, con un ángulo llamado *oblicuidad de la eclíptica*. Como el sol está contenido en ese plano, la otra coordenada será el ángulo que forma el radiovector del sol con una dirección fija, el equinoccio vernal, llamado *longitud de la eclíptica*. La posición del sol puede describirse entonces con estos dos ángulos.

Además de ser capaces de describir la trayectoria del sol a lo largo de la eclíptica, debemos ser capaces de *predecirla*. Para ello, hay varios algoritmos, que calculan la posición del sol en un determinado instante y propagan su movimiento para hallarla en todo momento. El programa usa uno de ellos, el publicado por la Plataforma Solar de Almería⁴, que se describe a continuación.

1. Cálculo de los días transcurridos desde las 12 del mediodía del 1 de Enero del 2000 (JD 2451545.0), dJD
2. Cálculo de las coordenadas eclípticas del sol, en radianes, mediante las fórmulas

$$\Omega = 2,1429 - 0,0010394594 \cdot dJD$$

⁴Puede encontrarse, junto con mucha más información, en <http://www.psa.es>

$$\text{Longitud media} = 4,895063 + 0,017202791698 \cdot dJD$$

$$\text{Anomalia media} = 6,24006 + 0,0172019699 \cdot dJD$$

$$\begin{aligned}\text{Longitud eclíptica} &= \text{Longitud media} + \\ &+ 0,03341607 \cdot \text{sen}(\text{Anomalia media}) + 0,00034894 \cdot \text{sen}(2 \cdot \text{Anomalia media}) - \\ &- 0,0001134 - 0,0000203 \cdot \text{sen}(\Omega) \\ \text{Oblicuidad eclíptica} &= 0,4090928 - \\ &- 6,2140 \cdot 10^{-9} \cdot dJD + 0,0000396 \cdot \cos(\Omega)\end{aligned}$$

3. Conversión de las coordenadas eclípticas al sistema de coordenadas celestiales.
4. Cálculo de las coordenadas cartesianas, necesarias por la estructura del programa.

Problema de los dos cuerpos

El problema de los dos cuerpos se conoce desde el siglo XVI cuando Johann Kepler enunció sus famosas tres leyes a partir de sus extensas observaciones y las de su maestro

- *Los planetas en su movimiento alrededor del sol siguen órbitas elípticas, con uno de los focos ocupado por el sol*
- *Cada planeta describe su órbita de manera que la linea que une el sol con el planeta cubre áreas iguales en tiempos iguales*
- *El cociente entre el cubo del semieje mayor de la elipse y el cuadrado del periodo de dicha elipse es un valor constante e independiente del planeta*

Pero no fue hasta finales del siglo XVII, cuando Newton se basó en estas leyes y en las observaciones de Halley para formular su teoría de la gravitación universal: *La fuerza gravitatoria entre dos objetos es proporcional a cada una de las masas e inversamente proporcional al cuadrado de la distancia que los separa. Esta fuerza actúa sobre la linea recta que une los dos objetos.* La formulación matemática de esta ley puede verse como

$$F = G \cdot \frac{m_1 \cdot m_2}{r^3} \vec{r} \quad (2.4)$$

donde G es la constante de gravitación universal, m_1 y m_2 las masas de los dos objetos y r la distancia entre ellos. A día de hoy, se conoce como problema de los dos cuerpos a la interacción de dos partículas aisladas y sometidas únicamente a

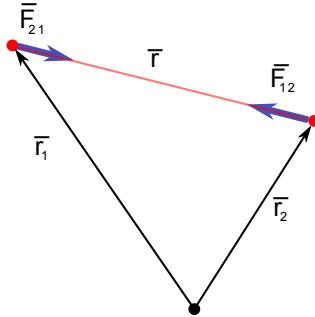


Figura 2.8: Problema de los dos cuerpos

la fuerza gravitatoria entre ellas, y es completamente integrable, es decir, tiene solución analítica.

A partir de esta ley y de 2.1 puede derivarse

$$\begin{aligned} m_1 \cdot \vec{r}_1 &= G \frac{m_1 \cdot m_2}{r^3} \vec{r} \\ m_2 \cdot \vec{r}_2 &= -G \frac{m_1 \cdot m_2}{r^3} \vec{r} \end{aligned} \quad (2.5)$$

y teniendo en cuenta que $\vec{r} = \vec{r}_2 - \vec{r}_1$ y que

$$\vec{r}_G = \frac{m_1 \vec{r}_1 + m_2 \vec{r}_2}{m_1 + m_2} \quad (2.6)$$

se llega fácilmente a

$$\begin{aligned} \vec{r}_2 &= \vec{r}_G + \frac{m_1}{m_1 + m_2} \vec{r} \\ m_2 \vec{r}_2 &= m_2 \vec{r}_G + \frac{m_2 \cdot m_1}{m_1 + m_2} \vec{r} = \frac{m_2 \cdot m_1}{m_1 + m_2} \vec{r} = -G \frac{m_1 \cdot m_2}{r^3} \vec{r} \\ \vec{r} &= -\frac{G(m_1 + m_2)}{r^3} \vec{r} \end{aligned} \quad (2.7)$$

Además, como se consideran las partículas aisladas del resto del universo, entra en juego la primera ley de Newton sobre el sistema

- *Todo cuerpo permanecerá en su estado de reposo o movimiento rectilíneo uniforme a menos que actúen fuerzas sobre él*

y el centro de masas del sistema seguirá una trayectoria rectilínea uniforme.

Llegado este punto, para las aplicaciones de satélites orbitando cuerpos masivos, no supone una idea descabellada despreciar la masa del propio satélite frente a la del cuerpo central, simplificando estas ecuaciones con $G(M+m) \approx GM = \mu$ y obteniendo

$$\vec{r} + \frac{\mu}{r^3} \vec{r} = 0 \quad (2.8)$$

Como además

$$\vec{r}_G = \frac{m_1 \vec{r}_1 + m_2 \vec{r}_2}{m_1 + m_2} \underset{m_1 \gg m_2}{\simeq} \vec{r}_1$$

la ecuación 2.8 describe precisamente el movimiento del satélite alrededor del cuerpo central.

Constantes del movimiento

Se define el momento cinético por unidad de masa \vec{h} como

$$\vec{h} = \vec{r} \times \vec{v}$$

que, derivando respecto al tiempo y usando 2.8

$$\frac{d\vec{h}}{dt} = \vec{v} \times \vec{v} + \vec{r} \times \vec{r} = -\vec{r} \times \frac{\mu}{r^3} \vec{r} = 0 \quad (2.9)$$

obtenemos que \vec{h} permanece siempre constante a lo largo de la trayectoria. Además, como $\vec{h} \cdot \vec{r} = 0$, la trayectoria estará contenida en el plano perpendicular a \vec{h} trazado por el origen.

A partir de 2.8, multiplicando escalarmente ambos miembros por \vec{v} puede obtenerse la conocida ecuación de la energía

$$\frac{1}{2}mv^2 - \frac{m\mu}{r} = E = cte \quad (2.10)$$

y, manejando estos parámetros y las distintas relaciones geométricas, se obtiene además la excentricidad

$$\begin{aligned} \vec{h} \times \vec{r} &= -\frac{\mu}{r^3} \vec{h} \times \vec{r}; \quad \frac{d}{dt}(\vec{h} \times \vec{v}) = -\mu \frac{d}{dt}\left(\frac{\vec{r}}{r}\right) \\ \frac{d}{dt}\left(\vec{h} \times \vec{v} + \mu \frac{\vec{r}}{r}\right) &= 0 \\ \vec{h} \times \vec{v} + \mu \frac{\vec{r}}{r} &= -\mu \vec{e} \end{aligned}$$

El vector \vec{e} es un vector constante en módulo y dirección y se conoce como excentricidad o vector de Laplace. Como $\vec{h} \cdot \vec{e} = 0$, el vector de excentricidad está contenido siempre en el plano del movimiento. Además, este vector apunta siempre al pericentro de la órbita. Despejando y usando la ecuación de la energía puede obtenerse

$$e^2 = 1 + \frac{2Eh^2}{m\mu^2} = cte \quad (2.11)$$

$$\vec{e} = -\frac{\vec{r}}{r} - \frac{1}{\mu}(\vec{h} \times \vec{v}) \quad (2.12)$$

Fórmulas que resultarán útiles más adelante.

Referencia perifocal

A la hora de describir la trayectoria de una partícula resulta muy conveniente usar un sistema de referencia ligado al plano de la órbita de manera que, una vez establecido este, todos los puntos de la curva quedan descritos por 2 parámetros y una variable. Se define la referencia perifocal por medio de 3 vectores.

- \vec{u}_1 en la dirección de \vec{e} , de manera que apunta al pericentro de la trayectoria
- \vec{u}_3 en la dirección de \vec{h} , perpendicular a la órbita
- \vec{u}_2 de manera que forme un triángulo orientado a derechas

Usando este sistema de ejes, la trayectoria queda descrita con 2 parámetros, y cada punto asociado a un único valor

Trayectoria elíptica

En el programa vamos a contemplar únicamente el movimiento elíptico, aunque el problema de los dos cuerpos admite también soluciones parabólicas e hiperbólicas. La razón de esta limitación es debida a que esta clase de órbitas son las más interesantes para el tipo de aplicaciones a las que este programa pretende apuntar. Órbitas de otro tipo implican grandes distancias y tiempos, donde las perturbaciones principales pasan a ser otras y la simulación y la integración numérica pierden fiabilidad. Todavía pueden introducirse unas condiciones iniciales que lleven a una órbita no elíptica, pero el análisis de órbita no funcionará correctamente.

Si se multiplican los términos de la ecuación 2.12 por \vec{r} se obtiene

$$\vec{r} \cdot (\vec{h} \times \vec{v}) + \mu r = -\mu \vec{r} \cdot \vec{e} = -\mu r e \cos \theta$$

$$\vec{r} \cdot (\vec{h} \times \vec{v}) = -\vec{h} \cdot (\vec{r} \times \vec{v}) = -h^2$$

Se obtiene que, la ecuación de la trayectoria elíptica con origen en uno de sus focos es

$$r = \frac{h^2/\mu}{1 + e \cdot \cos \theta} \quad (2.13)$$

A ϑ , el ángulo que forma el radiovector de la posición con el vector de excentricidad se le denomina *anomalía verdadera* y a la cantidad $p = h^2/\mu$ se la conoce como *semiparámetro*.

Siendo \vec{e} el origen de ángulos de la anomalía verdadera, pueden deducirse fácilmente los radios del pericentro y apocentro en las posiciones $\theta = 0$ y $\theta = 180^\circ$ respectivamente

$$r_p = \frac{p}{1 + e} \quad (2.14)$$

$$r_a = \frac{p}{1 - e} \quad (2.15)$$

y por tanto, el eje mayor de la elipse

$$2a = r_a + r_p = \frac{2p}{1 - e^2} = -\frac{m_2 \mu}{E} \quad (2.16)$$

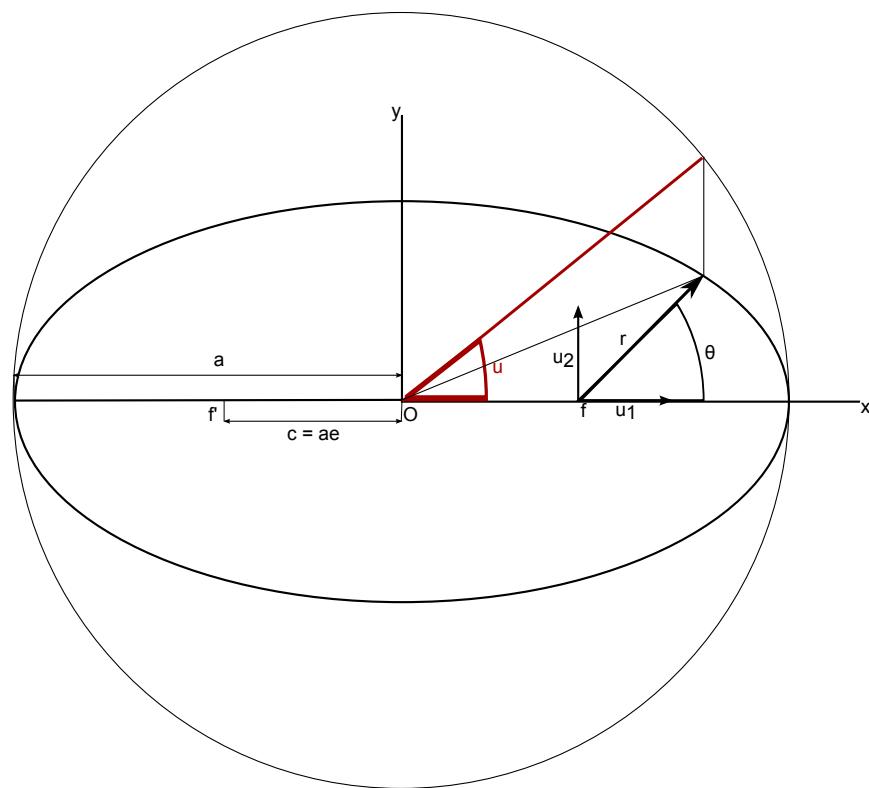


Figura 2.9: Elementos de la órbita elíptica

El valor del semieje mayor de la órbita depende únicamente de la energía. Como, para órbitas elípticas, la energía es negativa (Por el convenio de signos que se aplica al elegir la energía potencial negativa), el semieje mayor será siempre una cantidad positiva para órbitas elípticas, y negativa para trayectorias hiperbólicas.

El semieje menor

$$b = \sqrt{a^2 - c^2} = \sqrt{a^2(1 - e^2)} = \sqrt{ap}$$

Se define el periodo como el tiempo que tarda la partícula en describir una órbita completa o, lo que es lo mismo, el tiempo que tarda el radiovector de la posición en barrer todo el área de la elipse. Como la velocidad areolar es constante y de valor $h/2$, se obtiene fácilmente

$$\begin{aligned} T \cdot \frac{1}{2}h &= \pi ab = \pi a^{3/2} \sqrt{p} = \pi a^{3/2} \frac{h}{\sqrt{\mu}} \\ T &= 2\pi \cdot \sqrt{\frac{a^3}{\mu}} \end{aligned}$$

De nuevo, el periodo depende únicamente del semieje mayor y por tanto de la energía. Si se lanzan varias partículas desde el mismo punto y con la misma velocidad, todas describiría trayectorias muy distintas, pero todas con el mismo eje mayor y, por tanto, todas volverían al lugar de lanzamiento en el mismo instante.

En estos ejes perifocales queda descrita la trayectoria por medio de las ecuaciones

$$\begin{aligned} x &= r \cdot \cos\theta \\ y &= r \cdot \sin\theta \\ z &= 0 \end{aligned} \tag{2.17}$$

y la velocidad \vec{v} (A partir de 2.13, 2.17 y la primera fórmula de Binet)

$$\begin{aligned} v_x &= -\frac{\mu}{h} \sin\theta \\ v_y &= \frac{\mu}{h} (e + \cos\theta) \\ v_z &= 0 \end{aligned} \tag{2.18}$$

Puede resultar conveniente en algunos casos describir la trayectoria no tomando como origen el foco ocupado, sino el centro geométrico de la elipse O (Ver Figura 2.9). Se toman los ejes paralelos a los ejes perifocales, pero con origen en el centro geométrico y se imagina una circunferencia de radio a , sobre la que se proyecta (según el eje y) la posición de la partícula. En este caso, el ángulo u que forma el radiovector con el eje x se denomina *anomalía excéntrica* y sirve para describir la trayectoria en este nuevo sistema de ejes de manera muy sencilla con los dos semiejes de la órbita.

$$\begin{aligned} x &= a \cdot \cos u \\ y &= b \cdot \sin u \\ z &= 0 \end{aligned} \tag{2.19}$$

El cambio entre ambos sistemas se realiza de manera fácil mediante las ecuaciones

$$\sin\theta = \frac{\sqrt{1 - e^2} \sin u}{1 - e \cos u} \quad \cos\theta = \frac{\cos u - e}{1 - e \cos u} \tag{2.20}$$

o sus inversas

$$\sin u = \frac{\sqrt{1 - e^2} \sin \theta}{1 + e \cos \theta} \quad \cos u = \frac{\cos \theta + e}{1 + e \cos \theta} \quad (2.21)$$

La anomalía excéntrica sirve también como paso intermedio para calcular la ley horaria del movimiento. Hasta ahora tenemos una relación $r = r(\theta)$, de manera que para determinar $r = r(t)$ necesitamos $\theta(t)$. A esta última relación se la llama *ley horaria*. Apoyándose en la segunda ley de Kepler y en distintas propiedades geométricas, puede demostrarse que

$$M = n(t - \tau) = u - e \sin u \quad (2.22)$$

con $n = \frac{2\pi}{T}$ la velocidad angular media y τ el tiempo de paso de la partícula por el pericentro. A esta ecuación se la conoce como *ecuación de Kepler* y, salvo casos particulares, debe resolverse numéricamente.

Elementos clásicos de la órbita

El problema de una partícula orbitando alrededor de un cuerpo central queda inequívocamente determinado por 6 parámetros. Estos parámetros son típicamente la posición y velocidad de la partícula en un instante determinado, normalmente el inicial, y a partir de esas 3+3 componentes puede determinarse la posición que ocupará el objeto en cualquier otro instante de tiempo. Sin embargo, en numerosas ocasiones resulta más conveniente y cómodo usar otro conjunto de parámetros llamados elementos clásicos de la órbita, que simplifican mucho el cálculo de la posición y velocidad de la partícula, y permiten clasificar las órbitas más fácilmente. Estos elementos típicamente son:

- *El semieje mayor a* , que da una idea del tamaño de la órbita
- *La excentricidad e* , que indica la forma de la órbita
- *La longitud recta del nodo ascendente Ω , la inclinación i y el argumento del perigeo ω* , una serie de giros consecutivos que permiten situar el plano de la órbita en el espacio.
- *La anomalía verdadera ϑ* , que sitúa la partícula dentro de su órbita. Este parámetro puede ser sustituido por otro equivalente, como la anomalía excéntrica o el tiempo de paso por el perigeo

La determinación de estos elementos a partir de la posición \vec{r} y velocidad \vec{v} en un punto se hace de manera fácil mediante el siguiente algoritmo

- Determinación de \vec{h} y \vec{e}

$$\begin{aligned} \vec{h} &= \vec{r} \times \vec{v} \\ \vec{e} &= -\frac{\vec{r}}{r} - \frac{1}{\mu} (\vec{h} \times \vec{v}) \end{aligned}$$

- Determinación de la referencia perifocal y la dirección del nodo ascendente

$$\begin{aligned} \vec{u}_1 &= \frac{\vec{e}}{e} & \vec{u}_3 &= \frac{\vec{h}}{h} \\ \vec{u}_2 &= \vec{u}_3 \times \vec{u}_1 & \vec{n}_1 &= \frac{\vec{k} \times \vec{u}_3}{|\vec{k} \times \vec{u}_3|} \end{aligned}$$

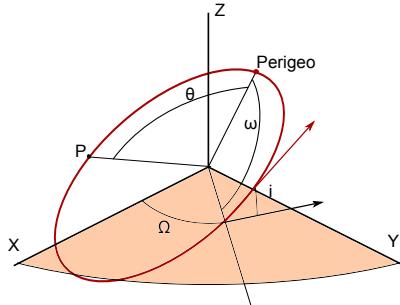


Figura 2.10: Elementos clásicos de la órbita

- Determinación de los elementos de la órbita

$$\begin{aligned}
 \cos \Omega &= \vec{n}_1 \cdot \vec{i} & \sin \Omega &= \vec{n}_1 \cdot \vec{j} \\
 \cos \omega &= \vec{u}_1 \cdot \vec{n}_1 & \sin \omega &= (\vec{n}_1 \times \vec{u}_1) \cdot \vec{u}_3 \\
 \cos i &= \vec{u}_3 \cdot \vec{k} & & (2.23) \\
 e &= |\vec{e}| \\
 a &= \frac{p}{1-e^2} \\
 \cos \theta &= \vec{r} \cdot \vec{u}_1 & \sin \theta &= \vec{r} \cdot \vec{u}_2
 \end{aligned}$$

Maniobras y triedro orbital

Se han presentado dos triedros distintos para describir el movimiento de un satélite alrededor de la tierra, uno ligado a unos ejes fijos y otro ligado al plano de la órbita. Sin embargo, en ocasiones resulta mucho más conveniente describir ciertos parámetros en un tercer sistema de referencia, que facilite cierto tipo de operaciones.

Se define un sistema de referencia ligado al satélite, no inercial, con origen en el centro de masas del objeto y unos ejes definidos como:

- Un eje x llamado posigrado, en la dirección de la velocidad instantánea del sólido
- Un eje z llamado normal, en la dirección perpendicular al plano de la órbita
- Un eje y llamado radial, elegido para que sea normal a los anteriores y forme un triedro a derechas

Este sistema de ejes se denomina **triedro orbital**, y es muy útil a la hora de describir maniobras orbitales, ya que permite clasificar las distintas maniobras según el eje a lo largo del cual se realice el impulso.

Ya se ha visto que la trayectoria que sigue un sólido en su movimiento alrededor de la tierra está perfectamente definida y determinada por su posición y velocidad en un instante. Para cambiar esta trayectoria es necesario introducir

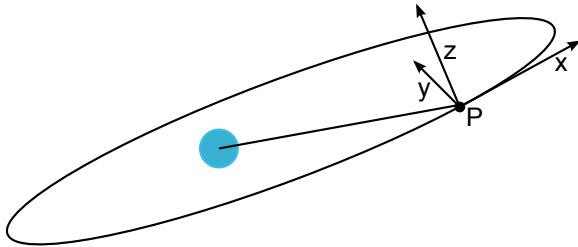


Figura 2.11: Triedro orbital

alguna fuerza extra que perturbe esta solución hasta el punto que deseemos. Con la mejora de los modelos existentes y la capacidad de cálculo, se están diseñando métodos de propulsión que aprovechan las perturbaciones ya existentes, pero sin duda la técnica más clásica, probada e implementada es equipar al satélite con un sistema de propulsión propia, un motor cohete. Esto permite al satélite modificar su órbita para distintos propósitos, desde realizar un *rendezvous* con otro sólido a simplemente mantener su órbita inicial corrigiendo las perturbaciones.

Un motor cohete es un sistema muy complejo, con muchos parámetros que definir y modelar, y muchas variantes que tener en cuenta, que hace que sea imposible tener una representación fiel en las etapas del proyecto que son objetivo de este programa. Sin embargo, la mayoría de las veces puede asimilarse un impulso de un motor cohete como un impulso puntual, instantáneo, que obedece a la conocida ecuación del cohete

$$\Delta v = g_0 I_{SP} \ln \frac{m_i}{m_f}$$

$$\Delta v = v_f - v_i$$

que puede interpretarse como que el encendido del motor provoca un cambio de velocidad del sólido proporcional a las características del motor y al logaritmo neperiano de la masa de combustible quemada. Aunque en la realidad los tiempos de quemado de la masa no sean instantáneos, normalmente son lo suficientemente cortos respecto al periodo de la órbita como para poder considerarlos así.

Puede considerarse entonces que el motor produce cambios instantáneos de la velocidad del satélite sin modificar su posición, y a este modelo se le denomina maniobra puntual. El programa considera todas las maniobras como puntuales, para ahorrar en la cantidad de datos previos necesarios antes de la simulación.

La razón de que se haya definido anteriormente el triángulo orbital es que el resultado de la maniobra no depende únicamente del Δv que proporcione el motor, sino también de la dirección en la que se realice el impulso. Si definimos el llamado *triángulo de velocidades*, se ve que el módulo y la dirección de la velocidad final dependen fuertemente de estos dos parámetros.

Si definimos el incremento de velocidad para que siga la misma dirección de la velocidad inicial, la velocidad final seguirá esta linea y habremos conseguido un fenómeno curioso. La órbita final estará contenida en el mismo plano que la inicial, y únicamente se ha modificado la energía cinética de la partícula. Podemos así controlar el semieje mayor y la excentricidad de la órbita, sin perder el plano orbital deseado. Un ejemplo de este tipo de maniobras sobre

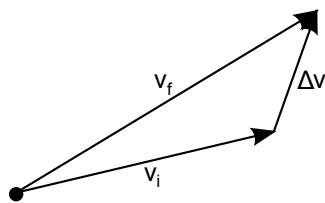


Figura 2.12: Triángulo de velocidades

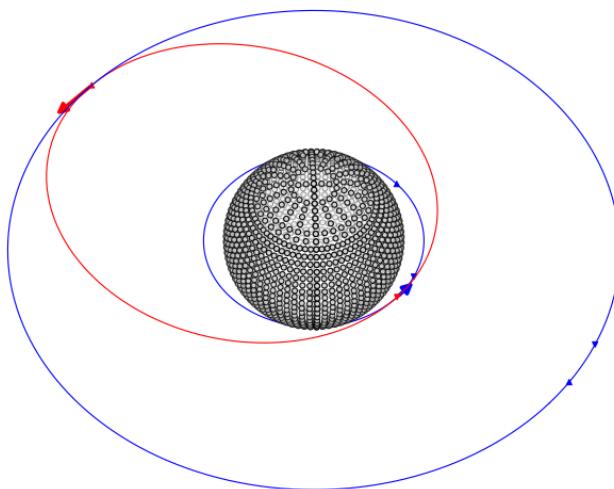


Figura 2.13: Transferencia de Hohmann

el eje posigrado es la transferencia de Hohmann, una maniobra doble entre dos órbitas circulares, con una órbita de transferencia elíptica entre ellas, de manera que los dos impulsos se realicen en el perigeo y el apogeo y el combustible total usado se minimice. Otro tipo de maniobra que se realiza sobre este eje es el *decay aerodinámico* producido por la atmósfera, que hay que corregir si no se quiere que el satélite caiga y se queme en la reentrada.

Sin embargo, si el impulso se realiza en cualquiera de los otros dos ejes, perpendiculares a la velocidad, el módulo de la velocidad final será prácticamente idéntico al de la velocidad inicial, y por tanto la energía de la órbita se mantiene. En este caso, la maniobra mantiene la forma y el tamaño de la órbita, pero cambia el plano orbital, de distintas maneras según la posición y dirección del impulso. Por ejemplo, un impulso según la dirección normal cuando el satélite está pasando por el plano del ecuador conllevará un cambio de inclinación, conservando el resto de los parámetros, mientras que el mismo impulso realizado 90 grados más tarde llevará a un cambio en la ascensión recta del nodo ascendente.

En general una maniobra puntual será una combinación de impulsos en los tres ejes simultáneamente, y la determinación de la órbita final será más compleja. En el simulador se han incluido herramientas para poder realizar estos cálculos más fácilmente.

Cuaternios

Ya se ha visto que manejar los elementos clásicos de la órbita implica considerar cuatro giros distintos a la hora de calcular una posición respecto a los ejes de referencia, y además estos giros deben realizarse en un orden concreto y preciso. Por si fuera poco, otras aplicaciones a otros problemas mecánicos pueden encontrar conveniente usar otra secuencia de giros para describir su estado. Es por ello que se necesita una manera de tratar de forma matemática esta serie de movimientos concatenados, y que dicha manera sea fiable y barata computacionalmente hablando.

Si pensamos en un vector \vec{u} , descompuesto en sus coordenadas cartesianas en un sistema de ejes (\vec{i} \vec{j} \vec{k}) de manera que $\vec{u} = u_1\vec{i} + u_2\vec{j} + u_3\vec{k}$, podemos expresar el vector \vec{u} en otro sistema de ejes resultado de aplicar un giro de ϑ grados alrededor de, por ejemplo, el eje z mediante las ecuaciones

$$\begin{aligned} u'_1 &= u_1 \cdot \cos \theta + u_2 \cdot \sin \theta \\ u'_2 &= -u_1 \cdot \sin \theta + u_2 \cdot \cos \theta \\ u'_3 &= u_3 \end{aligned}$$

o, escribiéndolo de forma matricial

$$\begin{pmatrix} u'_1 \\ u'_2 \\ u'_3 \end{pmatrix} = \begin{pmatrix} \cos \theta & \sin \theta & 0 \\ -\sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = R \cdot \vec{u}$$

A la matriz R se la denomina *matriz de cambio de base*, y su cálculo es sencillo, además de que el giro es aparente a primera vista. La operación de cambio de base también es sencilla y se puede realizar con papel y lápiz rápidamente. El proceso se complica cuando se concatenan varios giros seguidos, ya que al multiplicar las matrices pierden su forma tan regular. Sin embargo, hay que considerar que el ordenador realiza el mismo número de operaciones haya un 0 u otro número, introduciendo errores con cada operación, y también tiene que almacenar los 9 números que contiene la matriz. Es por esta razón por la que, en aplicaciones informáticas, se prefiera y resulta más eficiente el empleo de los denominados cuaternios.

Hasta ahora hemos estudiado el movimiento de partículas puntuales, de las que sólo importa su posición en el espacio. Sin embargo, a la hora de describir un sólido, no hay que indicar únicamente su posición, sino que su orientación también debe quedar descrita. Un sólido puede verse como un conjunto de partículas puntuales, cuyas posiciones están ligadas entre sí. Así, describiendo el movimiento de un punto cualquiera, y la orientación del sólido respecto a una referencia, todas las partículas del sólido quedan perfecta e inequívocamente definidas.

De manera parecida al caso anterior, puede expresarse el giro de un sólido alrededor de uno de sus puntos mediante una matriz, que liga unas coordenadas fijas al sólido (x_1, y_1, z_1) a la referencia exterior (x, y, z) de manera que

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = Q \cdot \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}$$

A la matriz Q se la denomina *matriz de rotación*, y debe cumplir una condición importante: Debe ser ortogonal. Esto implica varias propiedades muy

útiles, como que su matriz inversa se obtenga simplemente transponiéndola o que su determinante sea 1, pero sin duda su característica más importante es la siguiente: Sus 9 elementos pueden expresarse en función de tres parámetros independientes, en lugar de los 9 de una matriz normal.

Existen varios conjuntos de elementos que se usan habitualmente a la hora de representar una matriz de rotación. Uno de ellos son los famosos ángulos de Euler: El estado del sistema se describe como una sucesión de tres giros distintos, tomados en el orden adecuado alrededor de los ejes necesarios. Para los ángulos de Euler esta sucesión es:

1. Un giro alrededor del eje z, llamado de precesión
2. Un giro alrededor del nuevo eje x, llamado de nutación
3. Un giro alrededor del nuevo eje z, llamado de rotación propia

Esta secuencia permite a cualquiera que la conozca reconstruir la orientación del sólido a partir de únicamente los valores de los tres giros, que además resultan muy naturales. Como cada uno de los giros tiene a su vez una matriz de rotación muy sencilla, resulta muy fácil obtener los nueve elementos de la matriz de giro. Los elementos clásicos de la órbita son un giro de este tipo, encadenando la longitud de la linea de nodos, la inclinación y el argumento del perigeo en este orden. Este conjunto de giros presenta, sin embargo, una singularidad cuando la nutación es nula.

Otra secuencia de giros muy utilizada, dependiendo de las aplicaciones, son los llamados ángulos de navegación. Muy usados en la mecánica de vuelo por su inmediata aplicabilidad, se definen los tres giros como

1. Un giro alrededor del eje z, llamado ángulo de guiñada
2. Un giro alrededor del nuevo eje y, llamado de cabeceo
3. Un giro alrededor del nuevo eje x, llamado de balance

De nuevo, obtener la matriz Q resulta fácil a partir de las matrices de cada uno de los giros. Los ángulos de navegación presentan una singularidad cuando el cabeceo llega a 90 grados.

En general, se puede elegir cualquier secuencia de tres giros para describir la actitud de un sólido o un cambio de base, y se suelen notar por la secuencia de ejes en los que se realiza (Los ángulos de Euler serían 313 y los ángulos de navegación 321). Y aunque usar matrices de rotación resulte muy adecuado cuando se usa papel y lápiz, cuando se aplican estos principios a un programa informático, resulta mejor usar otro método para almacenar y operar estos giros: Los cuaternios.

Definición y operaciones

Se define un cuaternion como una extensión a los números reales, un conjunto de 4 números que siguen una serie de reglas definidas, y suelen expresarse de manera

$$\mathbf{q} = q_0 + q_1 \cdot i + q_2 \cdot j + q_3 \cdot k$$

A q_0 se le denomina parte real y al conjunto de componentes q_1 , q_2 y q_3 se le llama parte imaginaria. Se definen la suma y el producto de manera parecida a los

números complejos, teniendo cuidado de no mezclar las distintas componentes. Para los distintos productos cruzados, puede usarse la siguiente tabla, eligiendo el primer factor del producto entre las filas y el segundo entre las columnas.

	1	<i>i</i>	<i>j</i>	<i>k</i>	
1	1	<i>i</i>	<i>j</i>	<i>k</i>	
<i>i</i>	<i>i</i>	-1	<i>k</i>	- <i>j</i>	
<i>j</i>	<i>j</i>	- <i>k</i>	-1	<i>i</i>	
<i>k</i>	<i>k</i>	<i>j</i>	- <i>i</i>	-1	

(2.24)

Por ejemplo, si se tienen dos cuaternios

$$\begin{aligned} q &= 2 + 2i \\ r &= i + j + 2k \end{aligned}$$

el producto $q \cdot r$

$$\begin{aligned} q \cdot r &= (2 + 2i) \cdot (i + j + 2k) = 2 \cdot i + 2j + 2 \cdot 2k + 2i \cdot i + 2i \cdot j + 2i \cdot 2k = \\ &= 2i + 2j + 4k + 2 \cdot (-1) + 2 \cdot (k) + 4 \cdot (-j) = -2 + 2i - 2j + 6k \end{aligned}$$

Merece especial mención el hecho de que el producto entre dos cuaternios no es conmutativo, ya que el producto $r \cdot q$

$$\begin{aligned} r \cdot q &= (i + j + 2k) \cdot (2 + 2i) = i \cdot 2 + i \cdot 2i + j \cdot 2 + j \cdot 2i + 2k \cdot 2 + 2k \cdot 2i = \\ &= 2i + 2 \cdot (-1) + 2j + 2 \cdot (-k) + 4k + 4 \cdot j = -2 + 2i + 6j + 2k \end{aligned}$$

Se define el cuaternion conjugado $\bar{\mathbf{q}}$ como aquel con la misma parte real pero parte imaginaria cambiada de signo (Las tres componentes). Se define la norma de \mathbf{q} como el producto $|\mathbf{q}| = \sqrt{\mathbf{q} \cdot \bar{\mathbf{q}}} = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$. Nótese que el producto del interior de la raíz da siempre como resultado un número real, y por tanto se aplica la definición de raíz clásica. Así mismo, se llama *cuaternion unitario* a aquel que tiene norma unidad.

Cuaternios vectoriales

Del mismo modo que los números complejos resultan muy útiles a la hora de describir y operar con vectores en el plano, los cuaternios pueden usarse para los mismos fines en el espacio \mathbb{R}^3 . Se llama cuaternion vectorial a un cuaternion con parte real nula, de manera que las tres componentes de la parte imaginaria coinciden con las componentes de un vector.

Si se tiene un cuaternion unitario \mathbf{c} , de manera que $c_0^2 + c_1^2 + c_2^2 + c_3^2 = 1$, puede hacerse un cambio de variable

$$\begin{aligned}\cos \frac{\theta}{2} &= c_0 \\ \operatorname{sen} \frac{\theta}{2} &= \sqrt{c_1^2 + c_2^2 + c_3^2} \\ \vec{\mathbf{e}} &= \frac{c_1}{\operatorname{sen} \frac{\theta}{2}} \vec{\mathbf{i}} + \frac{c_2}{\operatorname{sen} \frac{\theta}{2}} \vec{\mathbf{j}} + \frac{c_3}{\operatorname{sen} \frac{\theta}{2}} \vec{\mathbf{k}}\end{aligned}$$

el cuaternion \mathbf{c} puede expresarse de manera

$$\mathbf{c} = \cos \frac{\theta}{2} + \operatorname{sen} \frac{\theta}{2} \vec{\mathbf{e}} \quad (2.25)$$

y el cuaternion \mathbf{c} queda perfectamente definido inequívocamente por el par $(\theta, \vec{\mathbf{e}})$.

Puede demostrarse entonces que la operación

$$\vec{\mathbf{v}} = \mathbf{c} \cdot \vec{\mathbf{u}} \cdot \bar{\mathbf{c}} \quad (2.26)$$

corresponde a un giro del vector $\vec{\mathbf{u}}$, ϑ grados alrededor de la dirección $\vec{\mathbf{e}}$.

Usar esta operación presenta muchas ventajas frente al empleo de matrices de giro. Al ser el cuaternion \mathbf{c} unitario, la relación inversa es obtenible inmediatamente sin más que multiplicar ambos lados de la ecuación 2.26 por $\bar{\mathbf{c}}$ y \mathbf{c} por la izquierda y derecha respectivamente, sin necesidad de la costosa operación de calcular la inversa de una matriz. Los cuaternios puede concatenarse de manera parecida a las matrices, mediante un producto directo, teniendo cuidado con el orden de los factores. Y por último, únicamente requieren almacenar 4 parámetros, en lugar de los 9 de la matriz. Es por todas estas razones que, en aplicaciones informáticas, resulta conveniente usar este sistema para realizar todos los giros necesarios.

Perturbaciones

Hasta ahora se ha visto el denominado como problema de los dos cuerpos sin perturbar, originado por dos partículas perfectamente puntuales moviéndose aisladas del resto del universo, sometidas únicamente a las fuerzas gravitatorias internas entre ellas. Sin embargo, cualquier lector versado puede ver las limitaciones de esta hipótesis. Los cuerpos en realidad son sólidos, con una distribución de masa cuyo parecido con una esfera será pura coincidencia. Los objetos en el espacio en realidad no están aislados, sino que están rodeados por miles y miles de millones de astros y cuerpos celestes, todos ejerciendo su atracción gravitatoria sobre ellos. Y no toda la fuerza que se ejerce entre cuerpos en el universo es del tipo gravitatorio (O , más concretamente, del tipo $1/r^2$).

Así, cuando se quiere incrementar la precisión de los cálculos y se intenta simular la realidad más fielmente, es necesario incluir toda esta batería de efectos en el proceso de cálculo, intentando modelarlos lo más fiel y precisamente que se pueda.

Sin embargo, la solución analítica resulta muy útil y adecuada para ciertas aplicaciones. ¿Es porque han decidido ignorar todos estos efectos y buscan una solución lo más pura posible, por mucho que se aleje de la realidad? No. Es porque la solución analítica todavía es una muy buena aproximación, cuyo cálculo

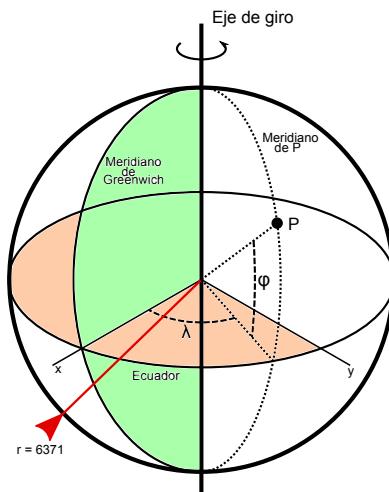


Figura 2.14: Tierra esférica

consume muy pocos recursos y cuyo estudio da pie a entender la mecánica orbital y poco a poco construir un modelo más complejo. Porque todos estos efectos de los que se ha hablado, que modifican el problema de los dos cuerpos puro, son típicamente mucho menores que el principal término que se da entre dos y por tanto pueden considerarse **perturbaciones**, que afectan en menor medida al sólido y provocan una solución que se va a parecer mucho a la del problema de los dos cuerpos sin perturbar.

Por supuesto el universo es enormemente grande y si nos pusiéramos a sumar términos de perturbación probablemente no acabaríamos. Por ello, es necesario priorizar aquellas más relevantes para la aplicación que se busca y despreciar aquellas de menor orden, cuyo cálculo va a resultar demasiado costoso. A continuación se presentan aquellas perturbaciones que suelen ser las más importantes para satélites en una órbita terrestre baja.

Tierra esférica

La esfericidad de la tierra se conoce desde tiempos antiguos. Ya en el siglo III a.C. el astrónomo griego Eratóstenes dio una primera aproximación numérica del radio de la tierra con una precisión espectacular para la época, midiendo la distancia entre Siena y Alejandría y el ángulo que formaba la sombra del sol en un pozo el mismo día en ambas ciudades. Esta medida se ha ido refinando con distintas observaciones posteriores hasta el valor del radio medio de 6371.0 Km que se maneja hoy en día. Así, es posible hacer una primera aproximación de la tierra

Se define la longitud como el ángulo que forma el meridiano que pasa por el punto P con el meridiano de referencia que pasa por Greenwich. La longitud en grados está en el intervalo $[-180, 180]$, siendo positiva al este de Greenwich y negativa al oeste. Se define de manera parecida la latitud, siendo el ángulo que forma el radiovector que une el centro de la tierra con el punto P con el radiovector que une el centro con el punto que está a la vez en el meridiano de P y en el ecuador. La latitud está contenida en $[-90, 90]$ grados, siendo positiva

para el hemisferio norte y negativa para el hemisferio sur.

Sin embargo, unido a la mejora de la precisión en la medida del radio de la tierra, se ha descubierto que la tierra no tiene forma esférica. El valor de 6371.0 no es el radio verdadero, sino una media calculada a través de modelos más precisos. La corrección más importante al modelo esférico es el achatamiento terrestre. La tierra es un planeta *oblato*, con un radio ecuatorial mayor que el radio en los polos. Esto es debido principalmente a la rotación de la tierra y las fuerzas de inercia que se producen. Una vez descartamos la aproximación esférica, la siguiente figura geométrica que aproxima mejor la tierra es un elipsoide.

Se define un cuerpo de revolución, con sección elíptica, de manera que todas las secciones que contienen al eje de revolución son iguales. Como elipse, se usan los parámetros geométricos típicos a , b , $e=c/a$ (semiejes mayor, menor y excentricidad respectivamente) y, por comodidad, se define un parámetro extra llamado *achatamiento*

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

$$e^2 = f \cdot (2 - f)$$

Los valores de estos parámetros han ido recibiendo, de nuevo, más precisión a medida que se recogían datos. Uno de los modelos más aceptados hoy en día, y el que usa el programa, es el **WGS84** (World Geodetic System 1984). Este sistema, aunque comprende modelos más complejos, da valores para un elipsoide

$$\text{Semieje mayor } a = 6378,137 \text{ Km}$$

$$\text{Inverso achatamiento } \frac{1}{f} = 298,257223563$$

El introducir el elipsoide conlleva complejidad adicional a calcular el valor del radio. Se define la *latitud geodésica* φ_g como el ángulo que forma la vertical local por P con el plano del ecuador, mientras que la *latitud geocéntrica* φ_c coincide con la definición anterior. La latitud geodésica es la que aparece en los mapas y la que se usa para georeferenciar un punto.

Las ecuaciones de la elipse en función de la latitud geodésica son

$$x_{P'} = \frac{a \cdot \cos \varphi_g}{\sqrt{1 - e^2 \sin^2 \varphi_g}} \quad (2.27)$$

$$z_{P'} = \frac{a(1 - e^2) \sin \varphi_g}{\sqrt{1 - e^2 \sin^2 \varphi_g}}$$

mientras que la latitud geocéntrica cumple

$$\cos \varphi_c = \frac{x_{P'}}{\sqrt{x_{P'}^2 + z_{P'}^2}} \quad (2.28)$$

$$\sin \varphi_c = \frac{z_{P'}}{\sqrt{x_{P'}^2 + z_{P'}^2}}$$

Para un punto cualquiera del espacio, con la ecuación de la recta normal al elipsoide y las ecuaciones anteriores se obtiene

$$x \cdot \sin \varphi_g - z \cos \varphi_g - a \frac{e^2 \cdot \sin \varphi_g \cdot \cos \varphi_g}{\sqrt{1 - e^2 \sin^2 \varphi_g}} = 0 \quad (2.29)$$

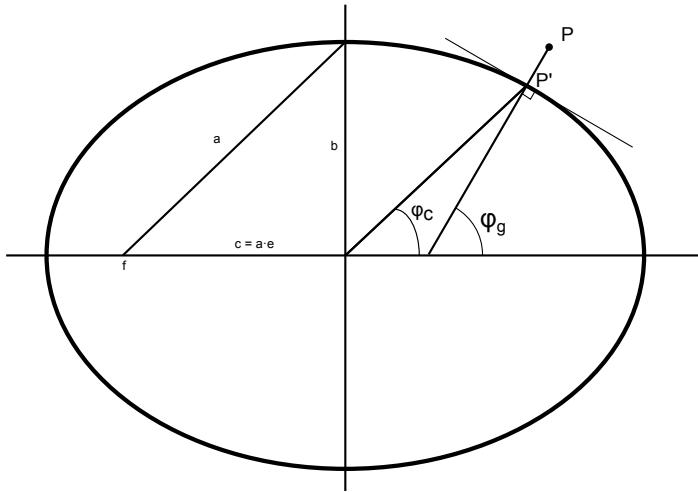


Figura 2.15: Sección del elipsoide de referencia

ecuación que ha de resolverse numéricamente para φ_g . Como primer paso en el proceso iterativo, pueden aproximarse las ecuaciones 2.27 y 2.28 para obtener

$$\varphi_g \approx \varphi_c + \frac{e^2}{2} \operatorname{sen} 2\varphi_c$$

El modelo completo WGS84 no incluye únicamente la definición de un elipsoide de referencia, sino que contempla una figura mucho más compleja y mucho más cercana a la realidad llamada *geoide*. El geoide no se define de la misma manera que los dos modelos anteriores, a partir de una figura geométrica, sino que es una *superficie equipotencial del campo gravitatorio terrestre*, teniendo en cuenta la fuerza de inercia de la rotación de la tierra y todos los armónicos de la gravedad. Las medidas de distintas misiones espaciales han permitido generar modelos tan complejos como la figura 2.16. Se suelen dar los datos como una serie de valores, llamados altura del geoide, que expresan la corrección que hay que hacer al elipsoide en cada punto para hallar la altura correspondiente a la latitud y longitud. La implementación de estos valores suele requerir de grandes tablas de datos, numerosas evaluaciones y varias interpolaciones antes de poder obtener un dato numérico realista.

⁵

Este último modelo escapa a la complejidad y precisión del programa, por lo que nos conformaremos con el uso del elipsoide definido.

Campo gravitatorio

La fórmula de la gravitación universal 2.4 ha asumido que las dos partículas que interaccionan son masas puntuales, objetos ficticios con toda su masa concentrada en un único punto adimensional. Sin embargo, ni la tierra es un objeto puntual ni, dentro de su forma geométrica vista en el apartado anterior,

⁵Imagen obtenida a partir de los datos de la misión GOCE de la ESA, que pueden encontrarse en <https://earth.esa.int/web/guest/missions/>

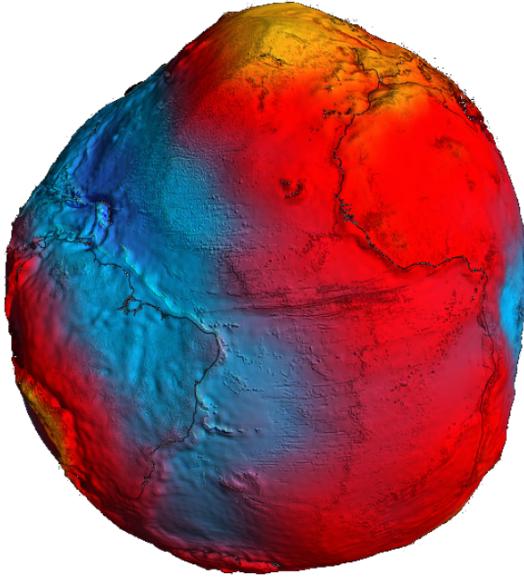


Figura 2.16: Geoide

toda su masa está distribuida de manera uniforme. Esto lleva a que el campo gravitatorio que genera no sea tan sencillo como la aproximación esférica supone.

Se define un campo escalar conservativo llamado potencial gravitatorio, como el trabajo que realiza la fuerza gravitatoria por unidad de masa, de manera que

$$U = \int_{r_{ref}}^r \vec{g} \cdot d\vec{r}$$

y

$$g = -\nabla U \quad (2.30)$$

En el caso de un cuerpo esférico y homogéneo la fuerza gravitatoria que provoca es la enunciada por la ley de la gravitación universal, y por tanto su aceleración, la fuerza por unidad de masa es

$$g = \frac{GM}{r^2}$$

y por tanto

$$U = - \int_{r_{ref}}^r \frac{GM}{r^3} \vec{r} \cdot d\vec{r} = GM \int_{\infty}^r \frac{1}{r^2} dr = -\frac{GM}{r} \quad (2.31)$$

pero esta aproximación sólo resulta adecuada para un cuerpo esférico.

Este potencial toma una forma hiperbólica, con una asíntota en el origen y alcanzando 0 en el infinito, donde la influencia del cuerpo central es nula.

Al producto del valor del potencial en el punto por la masa se le llama energía potencial gravitatoria, y puede verse como la capacidad de una partícula

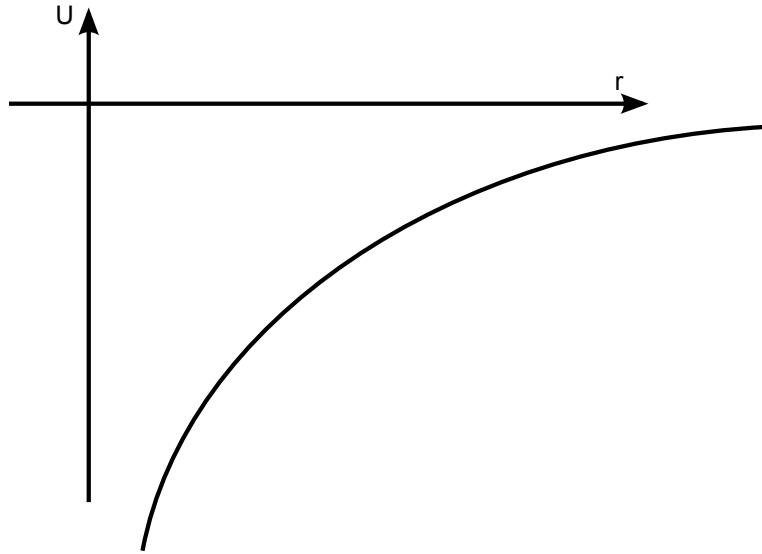


Figura 2.17: Potencial gravitatorio

sometida al campo gravitatorio de cambiar su posición. Nótese que por convenio el potencial gravitatorio siempre tiene un valor negativo.

Cuerpo casi esférico

Podemos considerar que un cuerpo no esférico está compuesto por infinitas masas puntuales de manera que

$$dU = \frac{G}{r'} dM$$

y el potencial total será la suma de cada uno de estos potenciales a lo largo y ancho de todo el cuerpo.

$$U = \int_V dU$$

Un tratamiento típico de esta ecuación es desarrollarla en una serie de armónicos esféricos. Puede hacerse de varias maneras, una de ellas es a través de la geometría del problema, descrita en la figura 2.18

Puede usarse el teorema del coseno para obtener

$$q^2 = r^2 + s^2 - 2 \cdot r \cdot s \cdot \cos \varphi$$

y el potencial originado en cada punto como

$$dU = -\frac{G}{r [1 + (\frac{s}{r})^2 - 2 \frac{s}{r} \cos \varphi]^{\frac{1}{2}}} dM$$

Desarrollando la expresión integral como una serie de potencias de $\frac{s}{r}$ se obtiene

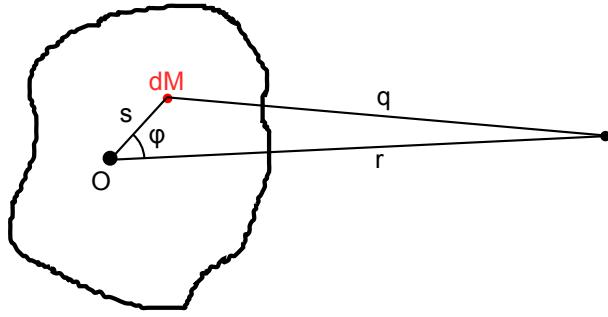


Figura 2.18: Geometría del potencial de un cuerpo general

$$U = -\frac{G}{r} \int dM - \frac{G}{r^2} \int s \cos \varphi dM - \frac{G}{r^3} \int s^2 dM + \frac{3}{2} \frac{G}{r^3} \int s^2 \sin^2 \varphi dM + \dots \quad (2.32)$$

Cuyo significado puede entenderse mejor si se analiza cada término por separado.

El primer término corresponde a la forma esférica del potencial, como si toda la masa estuviese concentrada en el origen.

$$-\frac{G}{r} \int dM = -\frac{GM}{r} \quad (2.33)$$

Este término es idéntico al de una partícula puntual y acentúa la idea de la técnica de las perturbaciones, de superponer términos cada vez más precisos a modelos en un principio sencillos.

El segundo término representa el producto de la masa por la distancia, una especie de momento de inercia de primer orden, llamado momento de área o momento estático.

$$-\frac{G}{r^2} \int s \cos \varphi = 0 \quad (2.34)$$

Si elegimos la posición de O coincidente con el centro de masas del sólido, este término se anula por la propia definición de centro de masas.

El tercer término corresponde al momento de inercia del sólido respecto de su centro. Si se toman unos ejes apuntando en las direcciones principales de inercia, los llamados ejes principales de inercia, puede escribirse este término como

$$-\frac{G}{r^3} \int s^2 dM = -\frac{G}{2r^3}(A + B + C) \quad (2.35)$$

Donde A, B y C son los momentos principales de inercia del sólido.

El último término puede verse como el momento de inercia del sólido respecto al eje que pasa por el centro de masas y por el punto P donde se está calculando el potencial. Llamando I a este momento

$$\frac{3}{2} \frac{G}{r^3} \int s^2 \sin^2 \varphi dM = \frac{3}{2} \frac{G}{r^3} I \quad (2.36)$$

La ecuación 2.32 puede escribirse entonces como

$$U = -\frac{GM}{r} - \frac{G}{2r^3}(A + B + C - 3I) \quad (2.37)$$

Una primera aproximación consiste en suponer que el cuerpo es un cuerpo casi esférico, es decir, un cuerpo de revolución con una generatriz no circular, de manera que sus momentos principales de inercia $A = B \neq C$. Se puede demostrar mediante cálculo tensorial que esta característica permite escribir

$$I = A + (C - A) \cos^2 \varphi$$

donde φ corresponde a la latitud. La potencia de esta simplificación de cuerpo casi esférico radica en que permite escribir la ecuación 2.32, mediante la fórmula de MacCullagh como

$$U = -\frac{GM}{r} + \frac{GJ_2 Ma^2}{r^3} \left(\frac{3}{2} \cos^2 \varphi - \frac{1}{2} \right) \quad (2.38)$$

$$J_2 = \frac{C - A}{Ma^2}$$

y de esta ecuación puede derivarse con facilidad en coordenadas rectangulares

$$\vec{g} = -\nabla U = \begin{pmatrix} -\frac{GM}{r^3}x \\ -\frac{GM}{r^3}y \\ -\frac{GM}{r^3}z \end{pmatrix} + \begin{pmatrix} \frac{-3C \cdot x \cdot (1-5(\frac{z}{r})^2)}{r^5} \\ \frac{-3C \cdot y \cdot (1-5(\frac{z}{r})^2)}{r^5} \\ \frac{-3C \cdot z \cdot (3-5(\frac{z}{r})^2)}{r^5} \end{pmatrix}$$

$$C = \frac{J_2 G M a^2}{2}$$

donde se ha tenido en cuenta a la hora de derivar que

$$\tan \varphi = \frac{z}{\sqrt{x^2 + y^2}}$$

y

$$r = \sqrt{x^2 + y^2 + z^2}$$

El modelo **JGM 3** (Joint Gravitational Model) da un valor ⁶ para el parámetro $J_2 = 1,082636 \cdot 10^{-3}$

Armónicos esféricos

Hasta el momento se ha considerado únicamente el achatamiento terrestre. Pero, aunque es el más importante, no es el único efecto de perturbación sobre el campo gravitatorio. Puede seguir desarrollándose el potencial en una serie de armónicos esféricos, de manera que cumpla la ecuación de Laplace ($\nabla^2 U = 0$) como

$$U = -[\frac{\mu}{r} + \frac{\mu}{r} \sum_{n=2}^N \frac{J_n P_n^0}{(\frac{r}{a})^n} + \frac{\mu}{r} \sum_{n=2}^N \sum_{m=0}^n P_n^m (C_n^m \cos m\theta + S_n^m \sin m\theta)] \quad (2.39)$$

Donde se han separado en otro sumando los términos simétricos en φ . Los diversos P_n^m se conocen como polinomios y funciones de Legendre, y están muy documentados y desarrollados en diversas fuentes hasta un alto orden. Los comportamientos de estos polinomios pueden verse en la figura 2.19, sacada de una de las referencias explicitadas en la bibliografía. Los distintos coeficientes J_n , C_n^m y S_n^m vienen dados en varios modelos empíricos. Uno de los más usados es el **EGM96** (Earth Gravitational Model 1996), que engloba el usado JGM3.
⁷

Este análisis detallado escapa de nuevo al rango del programa, que únicamente usará el término esférico y su perturbación mayor, J_2

Atmósfera

La característica principal del espacio es que está *vacío*, un gran vacío con pequeñas agrupaciones locales de materia que llamamos estrellas, planetas, asteroides y demás astros que pueblan el universo. Y aunque este hecho no es del

⁶<http://www.csr.utexas.edu/publications/statod>

⁷Los coeficientes hasta orden 360 pueden encontrarse en <http://cddis.gsfc.nasa.gov/egm96/getit.html>

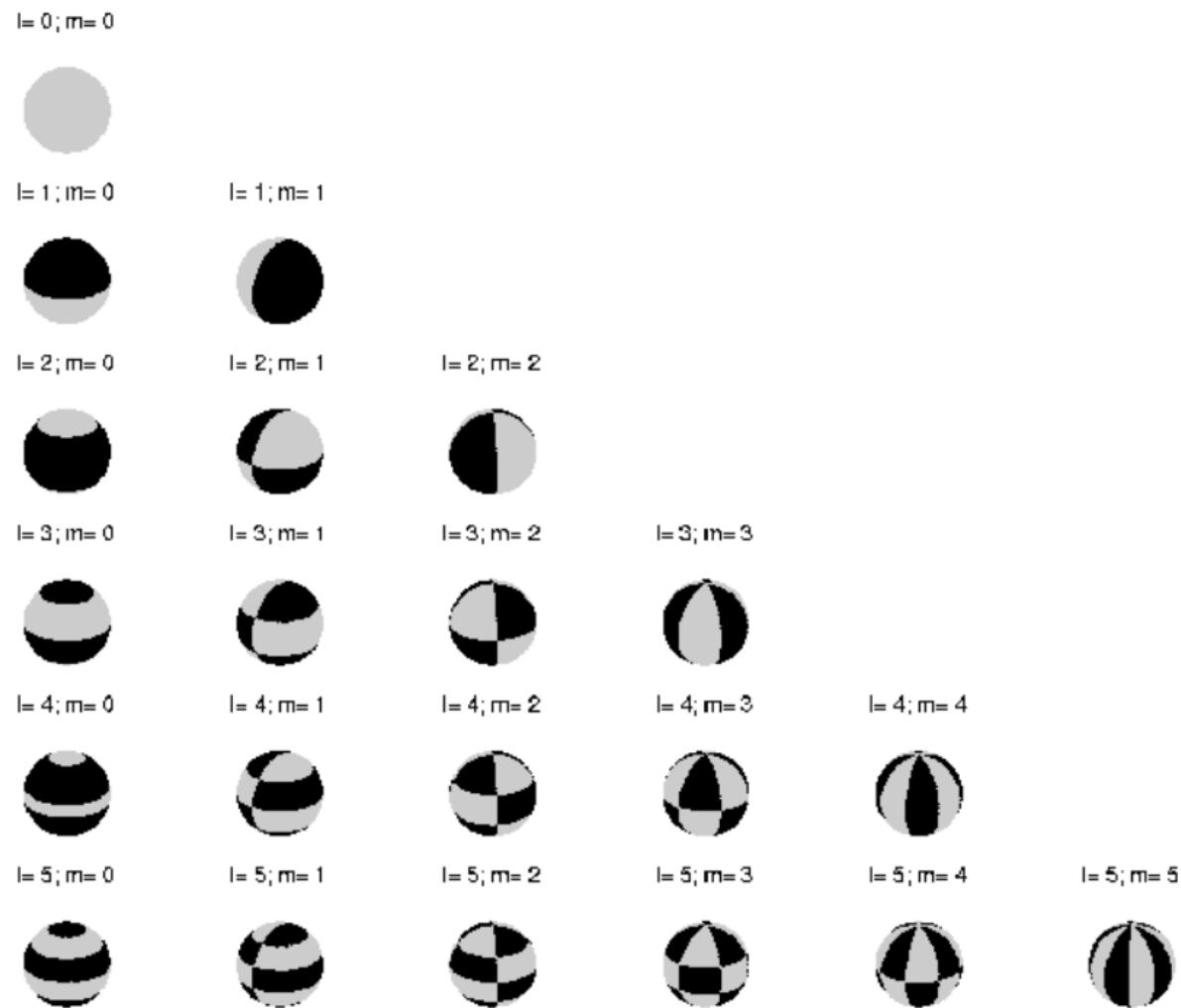


Figura 2.19: Armónicos esféricos

todo cierto, lo es menos aún cuando nos referimos a los alrededores de un astro con atmósfera gaseosa. ¿Donde está el límite entre la atmósfera y el vacío? Lo cierto es que poner una frontera es muy difícil, y es más adecuado tratar la atmósfera como algo continuo. Esta presencia de atmósfera afecta a los satélites en forma de resistencia aerodinámica, de manera similar a los aviones volando en la estratosfera.

La resistencia aerodinámica es fácilmente la perturbación más complicada de calcular de todas aquellas que afecta a los satélites que orbitan la tierra. El modelo más sencillo que podemos pensar es

$$\vec{\mathbf{a}_{\text{atm}}} = -\frac{1}{2} \frac{C_D A_f}{m} \rho v_w^2 \cdot \tilde{\mathbf{v}_w} \quad (2.40)$$

El cálculo de las distintas variables implicadas, no demasiado complicado cuando se trata de vuelo atmosférico, se dispara en el caso de satélites con mucha complejidad.

Velocidad relativa $\tilde{\mathbf{v}_w}$

A priori es el parámetro más sencillo de calcular, pero debido al poco conocimiento que hay del movimiento de las capas altas de la atmósfera y de los vientos que se producen en ellas no se alcanzará una precisión muy alta. Suponiendo que la atmósfera está en calma, está fija a la tierra y gira con ella, se tiene

$$\tilde{\mathbf{v}_w} = \tilde{\mathbf{v}_{\text{sat}}} - \omega_{\mathbf{T}} \times \vec{\mathbf{r}} \quad (2.41)$$

Área frontal A_f , coeficiente de resistencia C_D

El cálculo del coeficiente de resistencia se complica mucho frente al del vuelo atmosférico. A altas altitudes, el camino medio del flujo de aire comienza a ser del orden de los tamaños de los sólidos volando a través de él, y por tanto las hipótesis de flujo continuo dejan de ser válidas. Es necesario calcular ahora el flujo como un choque de partículas sueltas, con choques más o menos elásticos y difusos dependiendo de las propiedades de la superficie y de su orientación. Además, por la dificultad de tomar muestras, se dispone de muy pocos datos experimentales que sirvan para refinar y corregir estos modelos.

El área frontal también puede ser un problema, ya que no siempre es fácil conocer con precisión la actitud de un satélite frente al flujo. La geometría suele ser complicada, con grandes elementos como paneles o antenas extendidos y muchas partes de la estructura expuestas directamente al flujo.

La manera más fácil de tratar todos estos parámetros es agruparlos en un único valor, llamado *coeficiente balístico*, y definido como

$$CB = \frac{m}{C_D A_f} \quad (2.42)$$

aunque a veces se usa la relación inversa. La precisión de la fuerza de resistencia dependerá de la precisión con la que se sea capaz de expresar el CB. Valores típicos del coeficiente balístico rondan en el intervalo [10,50], dependiendo del tamaño del satélite, y serán mayores cuanto más masa tenga el satélite y sea menos susceptible a sufrir desviaciones por el viento. El programa usa un modelo

de CB constante, para evitar la complejidad y la cantidad de datos previos necesarios que habría que calcular en caso de querer determinar con más exactitud este valor.

Densidad ρ

El cálculo de la densidad es complejo, porque los modelos que se tienen están basados en datos y correcciones empíricas y no se tienen muy claros y detallados todos los fenómenos que influyen en este parámetro. Se sabe que el sol sufre un ciclo en su emisión de radiación lo largo de unos 11 años, que afecta directamente a la atmósfera haciendo que “engorde” o “adelgace” según la actividad sea mayor o menor. Esta emisión suele concretarse en el índice $F_{10,7}$, que es una medida de las emisiones del sol en la longitud de onda de 10.7cm, y que resulta muy conveniente ya que se tienen gran cantidad de datos de su comportamiento a lo largo de mucho tiempo⁸. Otro parámetro común que usan algunos modelos es la temperatura exoesférica, T_∞ , que es una medida indirecta del mismo fenómeno. El programa usa una temperatura exoesférica fija que corresponde a una media a lo largo del ciclo solar, $T_\infty = 1000K$.

También hay una variación cíclica basada en el ciclo día-noche, en la que la parte de la atmósfera expuesta en ese momento a la luz del sol está mucho más caliente que la parte a la sombra de la tierra, y por tanto la densidad del aire variará a lo largo de un órbita del satélite. Este efecto también se desprecia al usarse el modelo de temperatura exoesférica constante.

La variación de la densidad con la altura no es menos compleja, ya que, a diferencia de la atmósfera cercana, la toma de datos es complicada y esporádica. La composición de la atmósfera cambia y aparecen especies que no se dan en las capas bajas y que reaccionan de manera distinta a distintas frecuencias de radiación. Distintos modelos han tratado de representar todos estos fenómenos, cada uno trabajando sobre el anterior con nuevos datos para mejorar su precisión. Uno de ellos, el que usa el programa, es el **Jacchia 1977**, que da los distintos datos de densidad, temperatura y presión de las especies más comunes cada Km en un rango de alturas muy amplio, a partir de los cuales se obtienen en el resto de alturas por simple interpolación lineal. El hecho de tener esta estructura tabular hace que el algoritmo sea complejo y farragoso de reproducir aquí, pero puede encontrarse íntegro, junto con muchos modelos más, en <http://nssdcftp.gsfc.nasa.gov/models/>. Puede resumirse en

1. Para alturas inferiores a 86 Km, se usa la atmósfera estándar, computada cada intervalo de 1 kilómetro
2. Para alturas entre 86 y 90 Km, se integra la ecuación barométrica con peso molecular corregido
3. Para alturas superiores a 90 Km, se usa el modelo Jacchia 1977
4. Se calcula la disociación de oxígeno para alturas por encima de 90 Km
5. Se calcula la presencia de hidrógeno
6. Se corrige el modelo anterior con correcciones empíricas

⁸Datos sobre el flujo actual y un historial pueden encontrarse en <http://www.swpc.noaa.gov/phenomena/f107-cm-radio-emissions>

7. Se calculan las distintas densidades parciales de las distintas subespecies presentes, corregidas con los factores anteriores, en intervalos de 1 Km hasta la máxima altura
8. Se calculan las densidades entre los puntos de la tabla mediante interpolación lineal numérica

Para aliviar parte de la carga de cálculo, se considera que la atmósfera llega hasta los 1000 kilómetros de altura, siendo la densidad a partir de esta marca 0.

Naturalmente hemos usado el modelo más sencillo posible a la hora de calcular la fuerza de resistencia. Es posible implementar métodos de paneles, que integren en cada paso las distintas superficies para hallar la resistencia frontal y la de onda, y además tengan en cuenta la perturbación en la actitud que inducen y actúan en consecuencia. Sin embargo, carece de sentido implementar modelos tan complejos a la hora de hacer los primeros análisis de una misión, ya que los numerosos cambios que seguramente vayan a ser introducidos a lo largo del proceso de diseño serán bastante importantes, haciendo inútiles estos cálculos.

Perturbaciones lunisolares

Dos cuerpos celestes suelen considerarse como los más importantes, los que más afectan a la vida en la tierra y sus alrededores: El sol y la luna. Desde el ciclo día-noche a las mareas oceánicas, pasando por las estaciones, todo está influido por estos dos cuerpos y su movimiento relativo alrededor de la tierra. Y, por supuesto, los satélites cercanos a la tierra también. Ambos son cuerpos masivos, con suficiente masa como para generar un campo gravitatorio perceptible y, por si fuera poco, se mueven respecto a la tierra, cambiando la dirección en que actúa esta fuerza gravitatoria. Esta perturbación se conoce comúnmente como perturbación del tercer cuerpo, y la predicción de sus efectos es típicamente compleja.

La razón de que se incluyan en un mismo apartado la perturbación del sol y de la luna es que, normalmente, para satélites estacionados en una órbita terrestre ambas fuerzas son del mismo orden. El sol es un objeto con una gran cantidad de masa, pero la luna está mucho más cerca, y ambos aspectos se equilibran y se obtiene el mismo orden de magnitud. Para otro tipo de misiones, como las interplanetarias, otros cuerpos como Júpiter toman más importancia en el movimiento alrededor del sol (De hecho, Júpiter tiene dos cinturas de asteroides, los *troyanos*, cuya posición corresponde a los puntos de Lagrange, obtenidos de considerar la solución al problema de los tres cuerpos entre el sol, Júpiter y cada asteroide, tal es la magnitud de su perturbación)

El cálculo de estas perturbaciones es complejo. Es necesario conocer con exactitud la posición de los astros en todo momento y su distancia al objeto perturbado, siendo estos términos muy sensibles a la diferencia de órdenes de magnitud normalmente implicada en el problema. Debido a esto, y a que sus efectos son típicamente más pequeños y cíclicos que los originados por el campo

gravitatorio terrestre, se ha decidido despreciar su cálculo y acelerar el proceso de integración del simulador.

Presión de radiación

No toda la fuerza que ejerce un cuerpo celeste es gravitatoria. En muchos de ellos se encuentra una atmósfera, con composiciones variadas, o eyecciones de materia y energía que viajan por el espacio. En el caso de las estrellas, hay que sumar a la ecuación su emisión de radiación.

La presión de radiación se produce cuando una gran cantidad de fotones impacta contra una superficie de un cuerpo, provocando una fuerza no conservativa de perturbación. Esta fuerza es típicamente pequeña, pero en su aprovechamiento se basan técnicas de propulsión espacial tan llamativas como las velas solares.

El modelado de esta fuerza es complejo y se necesitan muchos datos para que sea preciso. El principal parámetro que influye es la presión solar, p_{SR} , cuyo valor varía enormemente entre estaciones y distancia al sol. Además, cambia enormemente (Hasta desaparecer) en la zona de sombra de la tierra.

Otro parámetro importante es el área con la que el satélite recibe esta radiación. La propulsión mediante vela solar necesita desplegar áreas enormes para conseguir una fuerza apreciable, aspecto que las hace inaplicables en órbitas cercanas a la tierra.

No sólo el área influye en la fuerza que se recibe de esta fuente, sino también el conjunto de propiedades superficiales de este área. Un cuerpo que refleje toda la radiación que le llegue recibirá más fuerza que uno que simplemente la absorba, mientras que uno transparente no sufrirá ninguna perturbación. En la realidad, ningún cuerpo será perfectamente reflectante o absorbente, por lo que el cálculo de este parámetro se complica, siendo necesarios varios estudios del diseño de su superficie. Aún más, no toda la radiación se reflejará especularmente (formando el mismo ángulo con la normal a la superficie que el ángulo de incidencia de la radiación), sino que se producirá el fenómeno de la radiación dispersa, modificándose ligeramente la dirección en la que se aplica la fuerza. Toda esta variabilidad puede englobarse en un valor, el coeficiente de reflectividad c_R , que toma un valor de 2.0 para un cuerpo perfectamente reflectante y 0 para uno transparente.

Esta fuerza puede expresarse como

$$\vec{\mathbf{a}}_{SR} = -\frac{p_{SR} \cdot c_R \cdot A_f}{m} \frac{\vec{\mathbf{r}}_{S-s}}{r_{S-s}}$$

donde $\vec{\mathbf{r}}_{S-s}$ es el radiovector que apunta del satélite al sol.

Este cálculo se complica mucho más si tenemos en cuenta que el sol no es la única fuente de radiación presente en el problema. Aunque la radiación de las estrellas es despreciable por su inmensa distancia a la tierra, hay otro factor importante que hay que tener en cuenta: El albedo de la tierra y la luna. La tierra refleja mucha de la radiación que recibe del sol, el denominado albedo, cuya cantidad varía enormemente dependiendo de las condiciones de la superficie: Si está nublado, si el satélite está sobre el mar, si está sobre una zona polar... Esta cantidad recibida de la tierra no es despreciable (Alrededor de un 20 % de la radiación recibida del sol se refleja) y complica enormemente el cálculo, amenazando y poniendo en tela de juicio la validez de la integración.

Como se ha podido ver, el cálculo de este factor es muy complejo, que implica muchos factores no muy sencillos de predecir y simular, de manera parecida a la resistencia atmosférica, pero que tiene un orden mucho menor a esta. Por ello, para simplificar el problema y no alargar el proceso de integración, se ha decidido despreciar el efecto de la presión de radiación del sol, ya que su implementación significaría mucho procesamiento sin una mejora notable de los resultados que justifique su incorporación a los modelos.

Problema perturbado

Hasta ahora se ha visto el planteamiento matemático únicamente del problema de los dos cuerpos sin perturbar. Sin embargo, ya se ha visto que hay presentes muchas más fuerzas además de la gravitatoria del cuerpo central, por lo que la ecuación 2.8 en realidad debe expresarse así

$$\ddot{\mathbf{x}} = -\frac{\mu}{r^3} \dot{\mathbf{r}} + \mathbf{F}_P \quad (2.43)$$

Esta incorporación, aparentemente tan inocente, hace que salvo para casos muy puntuales y teóricos el problema ahora no tenga una solución analítica. Es necesario pues abordarlo de otra manera. Una posibilidad es hacer una integración numérica, calculando todas las fuerzas en cada instante e integrando el paso para obtener la solución en un nuevo punto. Esta es la técnica que usa el programa para resolver el problema y ya se explicará en detalle más adelante.

Sin embargo, existen otro tipo de técnicas que toman otro enfoque: Ya que el problema sin perturbar tiene una solución analítica, conocida y sencilla, se aprovecha esta solución para construir la solución del problema perturbado. De esta rama de técnicas es de la que se ocupa la teoría de perturbaciones, que cuenta con varias herramientas a la hora de afrontar el problema.

Una manera de hacer esto se denomina *método de variación de las constantes*. Si se tiene la solución del problema sin perturbar $\mathbf{y} = \mathbf{y}(\mathbf{k}, t)$, donde \mathbf{k} engloba todas las constantes de integración que han surgido, se busca una solución de la forma $\mathbf{y} = \mathbf{y}(\mathbf{k}(t), t)$, donde ahora las constantes de integración son a su vez funciones del tiempo. Forzando a que esta solución sea solución del problema, puede obtenerse un sistema de ecuaciones que lleva a la obtención de las funciones $\mathbf{k}(t)$ y por tanto a la solución del problema perturbado.

Otro método de afrontarlo es el conocido como *método de Poisson*, heredado de la mecánica analítica, que permite estudiar la evolución del sistema a través de sus integrales primas.

El problema de los dos cuerpos perturbados es uno de los primeros campos a los que se aplicó la teoría de perturbaciones. Su análisis a lo largo de los siglos ha permitido la obtención de varios modelos de evolución del sistema, como las ecuaciones planetarios de Lagrange, que permiten obtener la progresión de los elementos clásicos de la órbita sometidos a un perturbación derivada de un potencial, o las ecuaciones de Gauss, que lo generalizan a una perturbación general expresada en el triedro orbital

$$\begin{aligned}
 \frac{d\Omega}{dt} &= \frac{r \cdot \operatorname{sen}(\omega + \theta)}{h \cdot \operatorname{sen} i} a_{pz} \\
 \frac{d\omega}{dt} &= -\frac{h}{e\mu} \cos \theta a_{px} + \frac{r+p}{eh} \operatorname{sen} \theta a_{py} - \cotg i \frac{r}{h} \operatorname{sen}(\omega + \theta) a_{pz} \\
 \frac{di}{dt} &= \frac{r}{h} \cos(\omega + \theta) a_{pz} \\
 \frac{de}{dt} &= \frac{h}{\mu} \operatorname{sen} \theta a_{px} + \frac{h}{\mu} \left(e \frac{r}{p} + [1 + \frac{r}{p}] \cos \theta \right) a_{py} \\
 \frac{da}{dt} &= \frac{2a^2}{h} (e \operatorname{sen} \theta a_{px} + [1 + e \cos \theta] a_{py}) \\
 \frac{dM}{dt} &= n + \frac{h\sqrt{1-e^2}}{e\mu} \left([\cos \theta - \frac{2er}{p}] a_{px} - [1 + \frac{r}{p}] \operatorname{sen} \theta a_{py} \right)
 \end{aligned}$$

Estas ecuaciones y las de Laplace dan la evolución con el tiempo de los elementos clásicos de la órbita. Sin embargo, para obtener los valores concretos es necesario integrarlas, normalmente numéricamente. Se ha pasado de tener que integrar un sistema de 6 coordenadas a integrar 6 ecuaciones por separado. Y aunque pueda parecer un paso hacia delante, el hecho de usar los elementos clásicos de la órbita como posición hace que haya que introducir más cálculos secundarios en cada paso de integración. Por ello, no se justifica su implementación en el programa y se toma la otra vía, la de la integración numérica directa del sistema de ecuaciones en coordenadas cartesianas.

Integración numérica

Ya hemos visto la solución analítica del problema de los dos cuerpos sin perturbar, pero la introducción de otras fuerzas con distintas formas matemáticas complica el problema bastante, hasta el punto en que resulte imposible hallar una solución sencilla. En este momento, es cuando se justifica la introducción de otro tipo de solución, una solución numérica. La elección del método numérico resulta fundamental a la hora de obtener una buena precisión en la solución. El método no sólo debe ser preciso, sino que además debe ser estable, propiedad que depende tanto del método numérico como del problema específico al que se aplica y del paso de integración que se elija. Por esta importancia, el programa deja elegir entre varios integradores y definir el paso a la hora de plantear las condiciones iniciales.

La ecuación 2.1 ampliada con las fuerzas de perturbación puede escribirse en términos

$$\ddot{\vec{x}} = F(t, \vec{x}, \dot{\vec{x}}) = -\frac{\mu}{r^3} \vec{r} + \vec{F_p} \quad (2.44)$$

Que es una ecuación diferencial de segundo orden. Definiendo un vector

$$\mathbf{y} = \{\vec{x}, \dot{\vec{x}}\} \quad (2.45)$$

puede escribirse

$$\dot{\mathbf{y}} = f(t, \mathbf{y}) \quad (2.46)$$

de manera que ahora se trata de un problema de primer orden, linealizable e integrable numéricamente. La función f puede descomponerse en dos términos según la forma de sus distintas componentes

$$\dot{\mathbf{y}} = \begin{pmatrix} \vec{x} \\ \vec{\dot{x}} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ F_x & F_{\dot{x}} \end{pmatrix} \cdot \begin{pmatrix} \vec{x} \\ \vec{\dot{x}} \end{pmatrix} + \begin{pmatrix} 0 \\ \vec{F} \end{pmatrix} = \mathbf{A} \cdot \mathbf{y} + b$$

Donde en la matriz aparecen los términos de las fuerzas que dependen linealmente de \vec{x} y $\vec{\dot{x}}$ respectivamente, y en b el resto de fuerzas implicadas. El análisis de la matriz \mathbf{A} permite determinar el rango de soluciones donde el método será estable, pero en este caso, al no haber fuerzas lineales, toma la forma $\mathbf{A} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix}$ y todas las fuerzas aparecen en el término independiente.

La integración numérica consiste en sustituir la solución analítica, que tiene forma de ecuación más o menos complicada, por una solución que está definida en una serie de puntos y sólo en ellos. La solución toma la forma de una lista de puntos, separados por lo que se conoce como el paso de integración. Aprovechándose de la continuidad de la solución y sus derivadas, se usa un punto (O varios) \mathbf{y}_n anterior para calcular la solución en el punto nuevo \mathbf{y}_{n+1} . Dependiendo del método que se elija, se necesitan más o menos valores anteriores para hallar esta nueva solución.

La base donde se sustenta todo método de integración numérica es la teoría de series de Taylor. En efecto, puede escribirse una función \mathbf{g} en un instante t como un desarrollo en serie

$$g(t) = g(t_0) + \Delta t \cdot g^I(t_0) + \frac{\Delta t^2}{2!} \cdot g^{II}(t_0) + \frac{\Delta t^3}{3!} g^{III}(t_0) + \dots$$

Donde se ha llamado Δt a la diferencia $\Delta t = t - t_0$ y g^I, g^{II}, \dots representan las sucesivas derivadas de la función g . La función desarrollada debe ser diferenciable (que tenga derivada y esta sea continua) hasta el grado en que se quiera expandir la serie.

Naturalmente, para poder hallar las derivadas con exactitud es necesario conocer la función de antemano, lo que invalida todo el propósito de la integración numérica.

Si se tiene el sistema 2.46, puede escribirse

$$\begin{aligned} y^I &= f(t, y) \\ y^{II} &= f' = \frac{\partial}{\partial t} f + \frac{\partial}{\partial y} f \cdot f \end{aligned}$$

Si no se conoce a función, la dificultad de hallar derivadas de orden alto impide que este método se use, salvo para las aproximaciones más simples, como el conocido *método de Euler*, que se puede expresar como

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \cdot \mathbf{f}(t_n, \mathbf{y}_n) \quad (2.47)$$

Esta complejidad a la hora de calcular las sucesivas derivadas puede solventarse sustituyendo su cálculo con la aproximación de su valor a través de los puntos ya conocidos de la solución. Hay dos vertientes que pueden tomarse aquí: Los métodos de *paso simple* utilizan únicamente el último punto conocido,

y aproximan las distintas derivadas mediante sucesivas evaluaciones de la función f en distintos puntos cercanos a \mathbf{y}_n , mejorando mucho la precisión frente al método de Euler. Otra escuela de métodos son los métodos *multipaso*, que aprovechan el hecho de que ya se conocen varios puntos anteriores de la solución para, en una aplicación de distintas diferencias finitas hacia atrás, combinarlos adecuadamente y aproximar las derivadas de más alto orden.

Ambos enfoques son en principio válidos, y sus diferencias son suficientes como para justificar la implementación de ambos y la necesidad de realizar un estudio más detallado. En el simulador se incorpora un método de cada tipo, ambos de cuarto orden, de manera que el usuario tenga control sobre cual quiere utilizar. Los resultados obtenidos con cada tipo de integrador pueden verse en el capítulo de validación.

Precisión y estabilidad

La precisión de la solución numérica obtenida es fundamental a la hora de valorar la aplicabilidad de los resultados obtenidos. De nada sirve obtener una trayectoria en un proceso de integración, si resulta que su parecido con la realidad es escaso y su uso induce más errores que progreso. Por ello, resulta fundamental estudiar como se comporta el error de la solución obtenida.

A la hora de hacer un estudio detallado, conviene separar el error en tres fuentes principales:

- El error del método: A igualdad del resto de parámetros, distintos métodos producirán resultados distintos. Resulta difícil predecir el comportamiento de un método específico cuando se aplica a un problema determinado. Algunos resultarán más adecuados y precisos cuando el orden de las ecuaciones sea bajo, mientras que otros se beneficiarán de la forma de cierto tipo de soluciones. El método más adecuado que se puede aplicar cambia según el problema y su morfología, y su elección juega un papel fundamental a la hora de obtener una solución precisa. Como norma general, métodos de orden más alto producirán mejores resultados al incluir en su cálculo derivadas más altas, a costa de aumentar el tiempo de computación.
- El error de truncación: El error de truncación es el error que se comete inevitablemente al dividir el tiempo, en realidad continuo, en pequeños paquetes de Δt . Estamos despreciando la evolución que ha tenido el sistema a lo largo de ese corto intervalo y aproximando su estado a aquel que tenía en el instante t_n . Resulta intuitivo y acertado suponer entonces que este error será menos cuanto menor sea el paso de tiempo que tomemos. En efecto, al extraer el método de Euler del desarrollo en serie de la función

$$\mathbf{y}(t) = \mathbf{y}(t_0) + \Delta t \cdot \mathbf{y}^I(t_0) + \frac{\Delta t^2}{2!} \cdot \mathbf{y}^{II}(t_0) + \dots$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \Delta t \cdot \mathbf{f}(t_n, \mathbf{y}_n)$$

estamos cometiendo un error, cuyo orden será $O(\Delta t^2)$. Cuando el paso de tiempo se hace menor, el error de truncación disminuye y se dice que la solución *converge con Δt^2* . Métodos de más alto orden convergen con un mayor exponente, ya que están despreciando menos términos del desarrollo en serie en cada paso.

- El error de redondeo: La implementación informática de cualquier cálculo conlleva cierto error inevitable. Los números en el mundo físico no tienen límite, y a pesar de que $\sqrt{2}$ tiene infinitos decimales, al usarlo en operaciones estaremos usando su valor **exacto**. Sin embargo, un ordenador tiene una cantidad limitada de memoria, una cantidad finita de bits de información, y por tanto, no puede abarcar un concepto como el infinito. Los números en un ordenador se guardan en un número determinado de bits, ya sean 8 (simple precisión) o 16 (doble precisión) y su manejo conlleva una pérdida inevitable de información. Esto es lo que se conoce como error de redondeo. En ciertas aplicaciones, este error no molesta mucho. Al hacer una suma, las dos cadenas de bits operan y el resultado tendrá cierta precisión, que será suficiente para la mayoría de aplicaciones. Sin embargo en un proceso de integración numérica es mucho más importante. Ese pequeño error, insignificante, que se comete en el primer paso, está incluido en el cálculo del segundo paso. Para el tercero, no sólo tenemos el error cometido en el segundo, sino también el acumulado desde el primer paso. Esta cadena sigue aumentando, introduciendo errores poco a poco hasta que llega el momento en que la solución obtenida diverge enormemente de la real. Es necesario pues controlar y tener en cuenta este fenómeno. Puede entenderse intuitivamente que el error final depende del número de pasos dados lo que nos lleva a la conclusión, completamente antiintuitiva, de que el error de redondeo **aumenta cuando se disminuye Δt** .

El comportamiento opuesto de los errores de truncación y de redondeo lleva a un fenómeno curioso. Existe un paso de tiempo óptimo, donde el error final cometido en la solución es mínimo. Por debajo de él, el error de redondeo enturbia la solución y por encima la empaña el de truncación. La elección de Δt resulta pues fundamental a la hora de integrar numéricamente un sistema. Una filosofía de métodos que el simulador no incorpora, llamados *de paso variable*, aprovecha este comportamiento de Δt para elegir dinámicamente en cada paso un paso de tiempo cercano al óptimo, dependiente del estado instantáneo del sistema, para minimizar el error en la solución final.

Se ha mencionado antes el concepto de estabilidad. Es otro apartado muy importante a la hora de elegir el método numérico a usar. La estabilidad es la capacidad de un determinado método de integración de mantener el error acotado a lo largo de los distintos pasos de integración. En un método inestable el error progresará de manera exponencial a lo largo del tiempo y hará inútil la solución obtenida. La estabilidad es una propiedad del problema completo, es decir, depende del sistema de ecuaciones que se quiere integrar y del método de integración que se elige para ello.

Del sistema de ecuaciones se obtienen unos valores λ , una especie de autovalores propios del problema, que caracterizan de alguna manera la forma de las ecuaciones implicadas. Se obtienen al resolver el problema $y' = \lambda y$, y dependiendo de la física implicada, su cálculo puede ser extremadamente complejo.

Cada método de integración tiene lo que se conoce como una *región de estabilidad* propia, que se obtiene al estudiar cada método. Puede demostrarse entonces que, si la combinación $\lambda \cdot \Delta t$ cae dentro de la región de estabilidad, el método será estable para ese problema en concreto.

Por ejemplo, si se supone que se tiene un sistema tal que

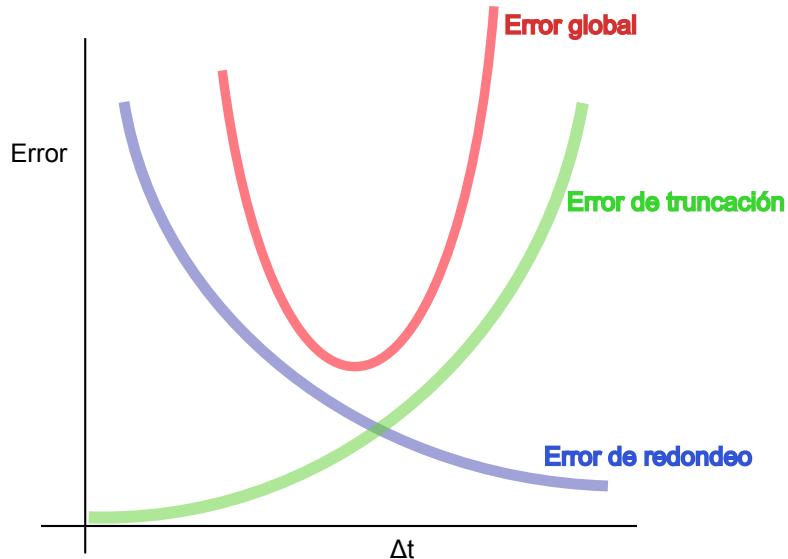


Figura 2.20: Error de redondeo, truncación y global

$$u' = \lambda u$$

al aplicarle el método de Euler se obtiene

$$U^{n+1} = (1 + \Delta t \cdot \lambda)U^n$$

y se dice que el método es *absolutamente estable* cuando se cumple que $|1 + \Delta t \cdot \lambda| \leq 1$

En general, el cálculo de λ dará valores complejos y la región de estabilidad se define en el plano z. La forma y tamaño de la región de estabilidad varía de un método a otro, pero puede verse que, en general, introducirá restricciones adicionales a la elección de Δt .

Runge-Kutta

Esta familia de métodos se caracteriza por ser de paso simple, es decir, por necesitar únicamente el último paso de integración para calcular el siguiente. El aumento de precisión respecto al método de Euler consiste en evaluar las derivadas de orden más alto a través de la evaluación de f en distintos puntos. Uno de los más usados es el de cuarto orden definido por

$$\begin{aligned} y_{n+1} &= y_n + \frac{\Delta t}{6}(k_1 + 2k_2 + 2k_3 + k_4) \\ k_1 &= f(t_n, y_n) \\ k_2 &= f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2}k_1\right) \\ k_3 &= f\left(t_n + \frac{\Delta t}{2}, y_n + \frac{\Delta t}{2}k_2\right) \\ k_4 &= f(t_n + \Delta t, y_n + \Delta t k_3) \end{aligned} \tag{2.48}$$

Adams-Bashforth

Esta familia de métodos se caracterizan por ser multipaso, por usar varias soluciones anteriores para calcular un paso determinado, y por ser explícitos, es decir, porque la solución del nuevo paso depende únicamente de las soluciones en los puntos anteriores, y no de la solución nueva desconocida. El método Adams-Bashforth de cuarto orden consiste en

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\Delta t}{24}(55 f_n - 59 f_{n-1} + 37 f_{n-2} - 9 f_{n-3}) \quad (2.49)$$

Naturalmente este método requiere conocer la solución en cuatro puntos anteriores, por lo que debe iniciarse con otro método de paso simple hasta tener el número requerido. En caso de seleccionarse este método, el programa usa el Runge-Kutta de cuarto orden para hallar los cuatro primeros puntos, antes de pasar a usar el Adams-Bashforth.

Adams-Moulton

Esta es una familia de métodos multipaso implícitos, ya que la solución en el nuevo paso cumple un papel en la ecuación que determina esa misma solución. Se debe disponer antes, por tanto, de una estimación de la nueva solución antes de poder aplicar este método. El Adams-Moulton de cuarto orden consiste en

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{\Delta t}{24}(9 f_{n+1} + 19 f_n - 5 f_{n-21} + f_{n-2}) \quad (2.50)$$

Como estimador para calcular f_{n+1} puede usarse cualquiera de los métodos anteriores. En el caso del programa, se trata del Adams-Bashforth de cuarto orden.

Predictor-Corrector

El uso de un método implícito lleva a obtener resultados más precisos que uno explícito, pero es necesario conocer una estimación de la solución antes de calcular la propia solución, estimación que no siempre tendrá la precisión que deseemos. Los métodos predictor-corrector se basan en un proceso iterativo para mejorar sucesivamente la precisión de estas estimaciones, mediante el uso de ambos tipos de métodos, que se puede resumir en el algoritmo

- Calcular mediante un método predictor explícito la primera estimación \mathbf{y}_{n+1}^0
- Calcular mediante el corrector implícito las sucesivas estimaciones $\mathbf{y}_{n+1}^1, \mathbf{y}_{n+1}^2, \dots, \mathbf{y}_{n+1}^k, \dots$
- Estimar el error de la solución. Si está por debajo de la tolerancia, se da por buena esa estimación. Si no, se repite el segundo paso.

La ventaja de este tipo de métodos es que permite estimar el error de la solución antes de darla por buena, mediante la evaluación de la variación de dos estimaciones sucesivas, que puede expresarse

$$Error \approx \frac{\mathbf{y}_{n+1}^k - \mathbf{y}_{n+1}^{k-1}}{\mathbf{y}_{n+1}^{k-1}} < tolerancia$$

La desventaja es que la convergencia de este proceso es baja y sólo asumible si la primera estimación queda lo suficientemente cerca de la solución, y es necesario imponer restricciones adicionales al paso de integración para asegurarla. Se añade además un nuevo bucle dentro de cada paso del bucle de integración, que puede afectar significativamente al tiempo requerido para completarla. En caso de elegir este método, el programa usa el Adams-Bashforth de cuarto orden como predictor y el Adams-Moulton de igualmente cuarto orden como corrector.

Tema 3

Manual de usuario

El programa se organiza en *escenarios*, el entorno en el que se va a realizar la simulación, compuesto por *parámetros* y *objetos*. Los parámetros siempre tienen un valor configurable, pero los objetos son introducidos por el usuario y personalizados a gusto. La simulación del escenario se lleva a cabo teniendo en cuenta ambos aspectos, aplicando los parámetros a los objetos según sea conveniente.

Funciones generales

La ventana que se recibe al iniciar el programa es la pantalla principal, alrededor de la cual se estructuran el resto de elementos del programa. Desde aquí se puede acceder a todas las funciones necesarias en la simulación.

Una descripción de los distintos elementos puede encontrarse a continuación.

Barra de estado

Proporciona información sobre los distintos procesos que está llevando a cabo el programa. Debido a la linealidad de algunas funciones del código, resulta importante saber cuando ha acabado un proceso para poder comenzar el siguiente. La barra de estado proporciona esta información de manera sencilla, y puede dar una estimación del tiempo restante para finalizar determinada tarea. Cuando una tarea determinada ha finalizado, se muestra en la barra de estado el texto “Completado”, momento en el cual puede lanzarse otro proceso.

Área de visualización

Proporciona información visual de la posición de los elementos presentes en la simulación. Las pestañas de la esquina superior izquierda permiten alternar entre ver la representación en dos o tres dimensiones. Dentro de la representación 3D las acciones

- Click izquierdo + arrastrar permite rotar la representación
- Click derecho + arrastrar permite incrementar o disminuir el zoom de la representación

permiten buscar una posición más favorable a la hora de visualizar los datos.

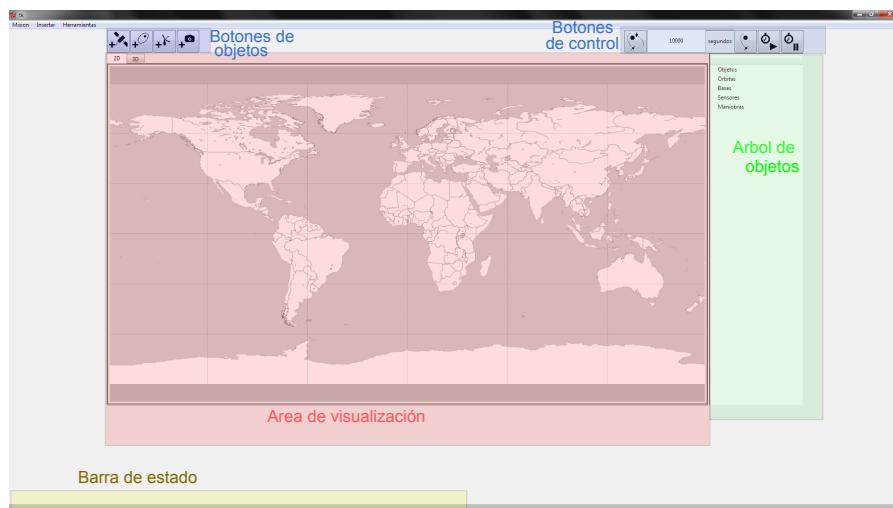


Figura 3.1: Ventana principal

Botones de objetos

Permiten introducir más elementos a la simulación, eligiendo el tipo de objeto que se desea. Su función está duplicada dentro del menú *Insertar* de la barra de herramientas superior para aumentar la claridad. Hay un botón para cada tipo de objeto básico. Sin embargo, si se comete un error al pulsar la categoría equivocada, la ventana siguiente permite cambiar el tipo del objeto que se va a añadir.

Árbol de objetos

Permite conocer de un vistazo qué elementos están formando parte del escenario, clasificados por tipo y referenciados por el nombre que se ha elegido para cada uno. Hacer click derecho sobre un elemento del árbol abre un menú contextual sobre él que permite, entre otras cosas, borrarlo permanentemente del escenario, ocultarlo de las distintas representaciones gráficas y abrir la ventana de configuración del objeto, para cambiar algún parámetro de su definición.

Botones de control

Permiten controlar el flujo de integración de la simulación, haciendo avanzar los objetos en el tiempo en su conjunto o individualmente. La función específica de cada uno se explicará más adelante.

Barra de herramientas

Permite acceder a distintas funciones organizadas en menús temáticos. El menú *Misión* permite acceder a comandos relacionados con el escenario en general. *Nueva misión* permite cargar un escenario vacío desde cero, mientras que *Abrir* permite cargar uno preexistente. *Guardar* y *Guardar como* permiten guardar el escenario en su estado actual en un archivo para su uso posterior. Si es

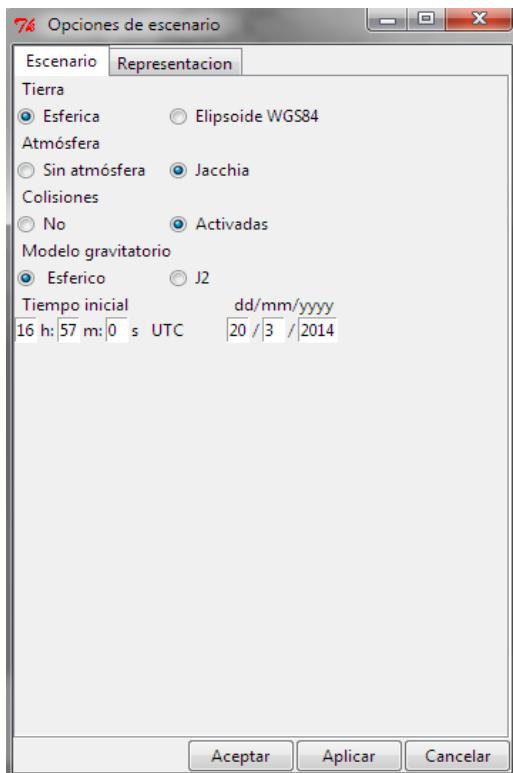


Figura 3.2: Ventana de parámetros de escenario

la primera vez que se guarda el escenario actual, ambas funciones son idénticas. Por último, *Cerrar* hace que el programa se cierre, perdiendo toda información no guardada.

El menú *Insertar* cumple las mismas funciones que los distintos botones de objetos, mientras que el menú *Herramientas* permite el acceso y uso de varias funciones extra.

Organizando la simulación

Ahora que ya se conocen las funciones básicas, vamos a organizar la primera simulación con el programa. El orden en que se realicen los pasos no es importante, siempre que se tenga en cuenta que la integración numérica se llevará a cabo con los valores y objetos que se hayan definido en el momento de lanzarla, por lo que cambiar parámetros a mitad de simulación puede comprometer la validez y consistencia de los resultados.

Configurando los parámetros

Hacer click en *Parámetros*, dentro del menú *Herramientas*, haceemerger una ventana que permite configurar el escenario y los distintos parámetros que se van a usar.

Tierra permite elegir entre un cálculo de tierra esférica o el uso del elipsoide definido por el modelo WGS84. Esto afectará, sobre todo, a los cálculos de los puntos subsatélite y las trazas sobre la proyección bidimensional

Atmósfera permite incorporar a la simulación la perturbación de la atmósfera, definida por el modelo Jacchia 77. La presencia de atmósfera implicará el cálculo en cada paso de la resistencia aerodinámica sobre los satélites, perturbando la órbita que siguen.

Colisiones permite desactivar el choque de los distintos objetos contra la tierra, para realizar un estudio más teórico del problema. Activarlo hará que el proceso de integración se pare cuando se detecte que la posición del satélite está dentro del globo terrestre. Desactivarlo impedirá esto, y permitirá realizar integraciones más agresivas y teóricas.

Modelo gravitatorio alterna el uso de un modelo de gravedad puramente esférico con la introducción de la perturbación J2 del EGM96. Al seleccionar esta última opción, añadirá al modelo de aceleración gravitatoria términos adicionales correspondientes al achatamiento de la tierra.

Tiempo inicial permite elegir con detalle el instante en que comenzará la simulación, parámetro que resulta fundamental a la hora de calcular las posiciones relativas de la tierra y el sol a lo largo de la simulación. Se divide en varios campos, cada uno de los cuales espera un número real.

Puede observarse la existencia de otra pestaña llamada *representación*, que permite alterar varios valores conectados a la representación gráfica del escenario. Tocar estos valores afecta únicamente a la representación y nunca a la simulación numérica, pero relajarlos puede suponer una mejora en el tiempo de cálculo del programa.

El valor contenido bajo la etiqueta *Representar objetos cada* afectará a la frecuencia con la cual se representan los puntos de una trayectoria. Así, un valor de 1 significará que se dibujan todos los puntos de la trayectoria sobre las dos representaciones gráficas, mientras que un valor de 10 indicará que se representa uno de cada diez, saltándose nueve a intervalos regulares (sin embargo, estos nueve puntos siguen en memoria y pueden ser accedidos)

El valor correspondiente a *Representar desde los últimos* indica la cantidad de puntos de un trayectoria que se representan, empezando por el final. Disminuir el valor significará recibir en los gráficos unas trazas más cortas. Los puntos que se quedan fuera de este intervalo todavía están guardados y se volcarán cuando se necesiten.

Debajo de estos parámetros aparecen dos botones. El primero, con la etiqueta *Dibujar los sensores* permite desactivar o activar a voluntad la representación de los distintos conos de visibilidad de los sensores, tanto en la representación tridimensional como en la bidimensional. Desactivarlos reducirá considerablemente los tiempos de proceso cuando hay presentes muchos objetos de este tipo.

El segundo, bajo la etiqueta *Dibujar la máxima visibilidad de los sensores* activa el cálculo y representación de la zona de acceso instantáneo del sensor, el área de la tierra que el satélite ve sin estar obstruido por el horizonte. Activarlo

puede proporcionar información muy valiosa para el diseño de los sensores y la cobertura de un satélite.

Añadiendo elementos

Ahora ya tenemos el escenario configurado a nuestro gusto, pero está vacío. Es hora de añadir los elementos sobre los que tendrá lugar la integración numérica. Los distintos objetos pueden añadirse desde los botones presentes en la ventana principal o desde el menú *Insertar*, y el tipo seleccionado se puede cambiar dentro de la ventana de configuración del objeto. Aunque las maniobras aparecen en el árbol, se crean de manera distinta al resto de objetos desde otro menú, que ya se explicará más adelante. Las unidades que admite el programa para los distintos parámetros vienen a la derecha cuando son necesarias. Si además el parámetro requiere algún formato más complicado que un número real, se indica así mismo a la derecha.

Añadir satélites

Al pulsar el botón con el símbolo del satélite o el elemento *Añadir sólido* del menú *Insertar*, se crea una ventana que permite la definición de un nuevo objeto de tipo sólido, la unidad básica para la simulación. Es a los sólidos a los que se aplican las distintas fuerzas y se hace la integración numérica que determina su posición a lo largo del tiempo.

Los campos *Nombre* y *color* son puramente estéticos, y permiten introducir el nombre con el que el programa se referirá al objeto y el color con el que se dibujarán la trayectoria y la traza en las representaciones tridimensional y bidimensional.

Los campos de velocidad y posición iniciales permiten establecer las condiciones de contorno iniciales del objeto, en el sistema de referencia ECI (Earth Centered Inertial), en unidades de kilómetros y kilómetros por segundo. Pueden introducirse directamente los valores, pero como completar estos campos puede resultar confuso y complicado, hacer click en el botón que aparece a su derecha llevará a otra ventana, que calcula y permite configurar estos valores automáticamente a partir de los elementos clásicos de una órbita elíptica.

Hay que tener especial cuidado al introducir estos campos para respetar el formato que tienen que tener estos dos parámetros: Un vector, englobado por corchetes y con las tres coordenadas separadas por espacios. Es necesario respetar este formato para que el resto de la simulación funcione correctamente.

Los campos de *Integrador* y *Delta-t* permiten controlar el proceso de integración numérica, pudiendo elegir entre Adams-Bashforth de orden 4, Runge-Kutta 4 y predictor-corrector (AB4-AM4). Cuando haya que realizar una integración, el programa reconocerá estos dos valores y llamará al proceso necesario para cada caso.

El campo de *coeficiente balístico* permite introducir un valor que se usará en el cálculo de la resistencia aerodinámica, si se ha seleccionado tenerla en cuenta en los parámetros de la simulación. Se define el coeficiente balístico como una combinación de parámetros de cada satélite, la masa, el área y el coeficiente de resistencia, en la forma

$$CB = \frac{m}{C_D A_f}$$

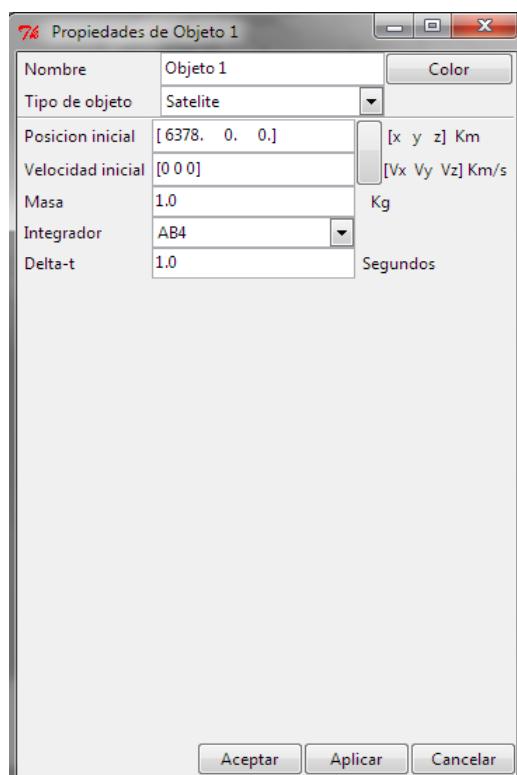


Figura 3.3: Añadir sólido

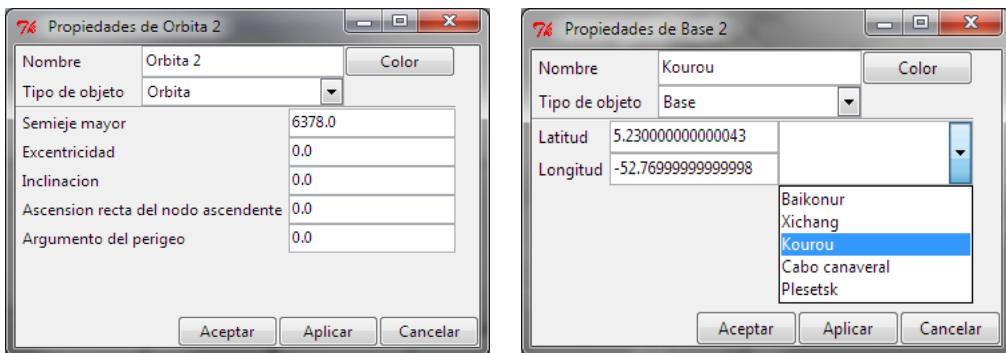


Figura 3.4: Añadir órbitas y bases

La definición de este parámetro permite simplificar mucho el cálculo de la resistencia y reduce la cantidad de información que es necesario conocer antes de realizar la integración.

El coeficiente balístico, definido de esta manera, representa la susceptibilidad del satélite a ser acelerado por el viento. Un satélite grande, con mucha inercia, tendrá valores altos de coeficiente balístico (a partir de 50) mientras que uno pequeño es más susceptible al viento y tendrá valores más bajos, del orden de 10.

Nótese que todos estos campos se aplican únicamente al objeto en particular, por lo que es posible definir varios objetos iguales con distintos integradores para comparar sus resultados, o modificar ligeramente las condiciones iniciales para ver como se propaga esa perturbación.

Añadir órbitas

Al presionar el botón correspondiente se abre la ventana de configuración de la órbita, donde se pueden configurar varios parámetros correspondientes con sus elementos clásicos. El semieje mayor se da en kilómetros y los distintos ángulos en grados. Las órbitas tienen varios usos como elementos auxiliares. Pueden servir como simple mecanismo de visualización para el estudio teórico del problema o pueden servir como punto de partida sobre el que construirlo. Se puede definir la posición de un sólido automáticamente a partir de las órbitas que haya definidas en el problema, ahorrando mucho tiempo si, por ejemplo, varios satélites realizan un vuelo en formación en la misma órbita. También son los elementos esenciales que sirven como punto de partida y definición del cálculo de distintas maniobras entre ellas. Al terminar de definir una órbita se guardan sus elementos clásicos y a continuación se calcula una serie de puntos a lo largo de ella, resultados de la solución analítica del problema de los dos cuerpos, y se dibuja esta trayectoria en la representación tridimensional.

Añadir bases

De la misma manera que el resto de objetos, permite configurar el nombre y el color con el que se representará. La posición de la base viene dada por sus coordenadas geográficas, que se pueden introducir a mano o seleccionar de una pequeña lista que se despliega al hacer click en el pequeño menú de la derecha.

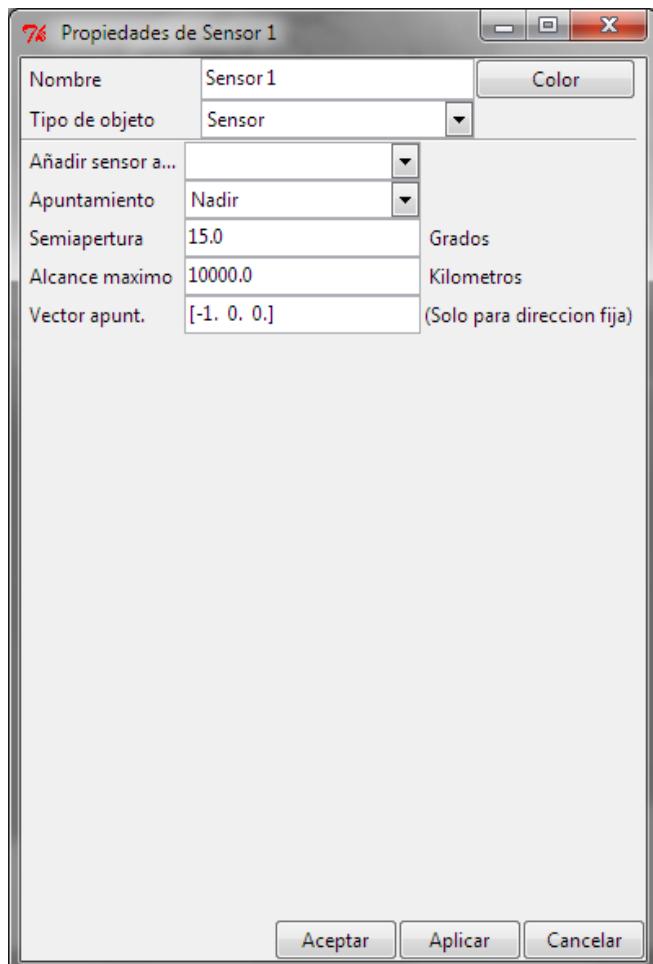


Figura 3.5: Añadir sensor

La posición tridimensional, además de por estas coordenadas geográficas, se verá afectada por el modelo de tierra que se haya elegido en los parámetros de la simulación.

Añadir sensores

Similarmente, al añadir un sensor se pueden configurar varios de sus valores.

Los sensores son objetos un poco especiales ya que no tienen posición propia. En su lugar, al definirlos se añaden a otro objeto de tipo sólido o de tipo base, que llamaremos objeto padre, y se ligan a ellos. Adquirirán su posición a partir de la de su objeto padre y calcularán sus datos a partir de ella. Por ello, es necesario a la hora de definir un sensor haber definido anteriormente el objeto que lo va a soportar. Al hacer click en el botón a la derecha de “Añadir sensor a” se despliega una lista con todos los objetos de las clases adecuadas presentes en el escenario, permitiendo elegir cual actuará de padre del sensor.

El apuntamiento de un sensor es fundamental a la hora de calcular sus actua-

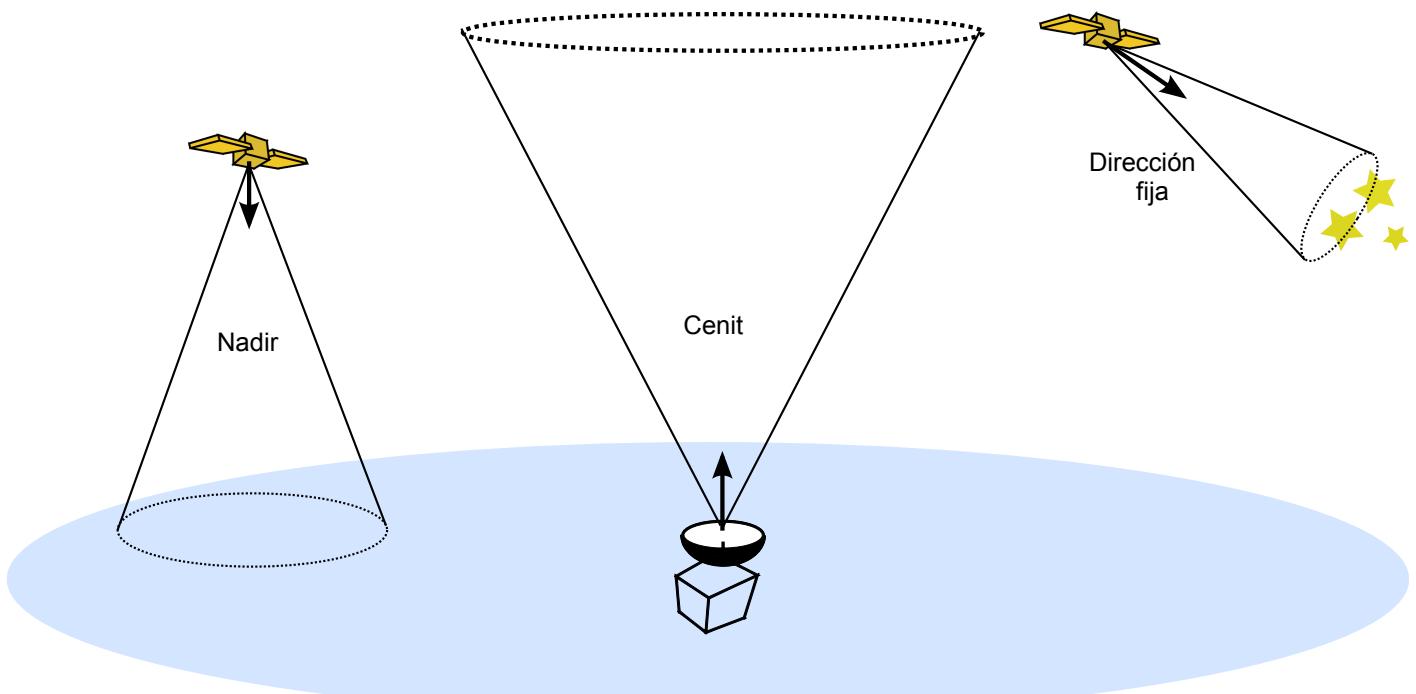


Figura 3.6: Apuntamientos posibles

ciones, por lo que el programa deja elegir entre un apuntamiento fijo a nadir, el punto subsatélite del satélite, un apuntamiento a cenit, la dirección contraria, y apuntamiento a una dirección fija. Esta dirección es un vector del sistema ECI, y puede definirse a través de la entrada del vector de apuntamiento. El apuntamiento será tenido en cuenta a la hora de calcular distintas características del sensor, como el cono de visibilidad.

Por simplicidad, se modelan los sensores como un cono, con el vértice en la posición del sensor y la directriz siguiendo el vector de apuntamiento instantáneo. El cono queda definido a partir de los valores adicionales de la semiapertura, el ángulo que forma una generatriz cualquiera con el vector director, y el alcance máximo, la máxima distancia a la que el sensor es capaz de obtener datos.

Controlando la simulación

El control de la simulación se hace mediante los botones de control de la ventana principal, situados en la esquina superior derecha.

Puede elegirse entre dos modos de hacer avanzar la simulación: Puede avanzar un tiempo determinado o hacerse de manera continua e indefinida. Esta elección se hace mediante el bloque de control de la esquina superior derecha de la pantalla principal.

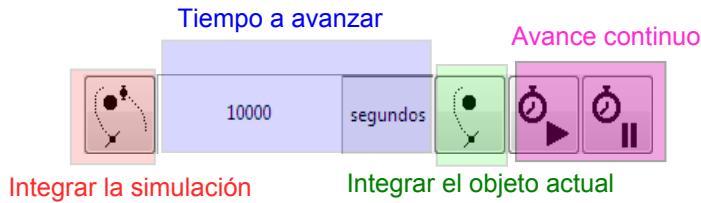


Figura 3.7: Control de la simulación

Avance fijo

El avance fijo es la opción que permite un mayor control del escenario, permitiendo elegir con exactitud cuanto tiempo se quiere integrar. El valor predefinido son 10000 segundos, pero haciendo click sobre el número se puede cambiar al valor deseado.

El primer botón de la izquierda permite integrar todo el escenario. Esta acción lanzará la integración de todos los elementos presentes en el escenario sucesivamente a lo largo del tiempo deseado. El progreso de cada objeto por separado se muestra en la barra de estado inferior, y al terminar la integración se mostrarán las trazas y trayectorias de todos los elementos presentes.

El segundo botón, a la derecha del tiempo a avanzar, permite integrar un único objeto cada vez. Cada objeto lleva una cuenta por separado del tiempo transcurrido desde el instante inicial, por lo que es posible manejar cada uno por separado, con un tiempo de integración diferente. El objeto que se va a integrar puede verse en el árbol de objetos, ya que aparecerá un pequeño símbolo delante de su nombre. Para cambiar el objeto que se va a integrar, basta con hacer doble click sobre el elemento deseado en el árbol para seleccionarlo, cambiando el indicador de posición.

Los únicos objetos que sufren el proceso de integración son los de tipo sólido. Si se hace click sobre el avance único estando seleccionado un objeto de cualquier otro tipo, no tendrá ningún efecto sobre el escenario.

Avance continuo

En lugar de integrar una cantidad fija de tiempo, puede quererse ver la evolución del escenario de manera continua. Los dos últimos botones permiten este control, iniciando y pausando este proceso. Se hace una corta integración seguida de la representación de los datos obtenidos, repitiéndose el proceso hasta que el usuario lo pause. Téngase en cuenta que el dibujado de datos es un proceso lento en el programa y el rendimiento durante esta función puede verse comprometido, por lo que puede resultar conveniente reducir la carga gráfica dentro de los parámetros del programa para acelerar esta operación. Este modo de avance puede resultar útil para conocer la evolución del sistema más detalladamente y al instante, en lugar de tener que hacer un postprocesado de los datos.

Resultados

Ahora que ya hemos introducido los elementos que queríamos analizar, y hemos integrado hasta el punto necesario, vamos a ver como acceder a los re-

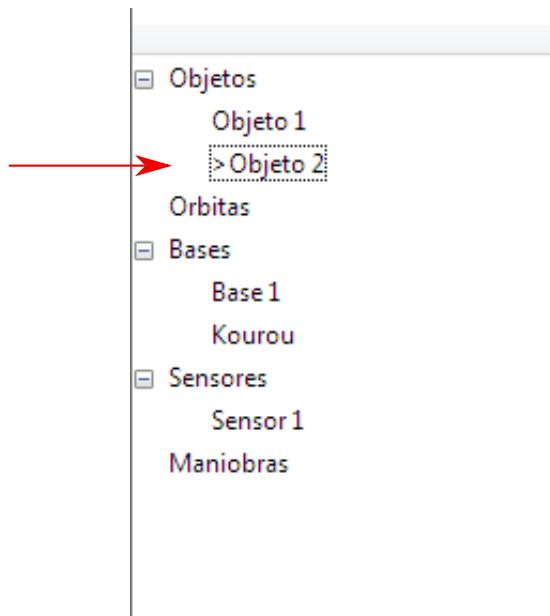


Figura 3.8: Árbol y objeto seleccionado

sultados. En el proceso de integración se genera una cantidad inmensa de datos numéricos de posiciones y velocidades. Además, son datos básicos, fundamentales para el programa, pero poco útiles para el usuario final. Es por ello por lo que hacen falta varias herramientas de procesado y tratamiento de estos datos, para poder analizarlos y generar alguna información que resulte útil para un proceso ingenieril. Se describen aquí las herramientas que incorpora el programa a tal fin.

Como todos a lo largo de nuestra carrera nos acostumbramos a usar ciertos programas y procesos que conocemos, a los que nos acostumbramos y con los que nos sentimos cómodos, resultaría injusto obligar al usuario a usar únicamente las herramientas de análisis que ofrece el programa y limitarle a ellas. Por esto, es necesario disponer de algún método de extraer los datos generados fácilmente para poder usarlos con algún proceso externo. Se han creado y se describen aquí también varias herramientas que permiten hacer esto, para permitir al ingeniero si lo desea usar cómodamente este programa como parte de un proceso más amplio, y no tener que limitarse a él.

Representación gráfica

El más evidente se proporciona sobre las representaciones 2D y 3D. Las trayectorias de los distintos objetos y las trazas que dejan sobre el mapa. Esta información, cualitativa de momento, resulta muy útil para un ingeniero con experiencia en mecánica orbital, ya que permite estimar fácilmente parámetros esenciales como la cobertura, la repetitividad de las trazas, forma de las órbitas, etc.

Sobre el mapa se representan también las posiciones actuales de los distintos objetos mediante unos pequeños iconos. Un pequeño dibujo de un satélite típico,

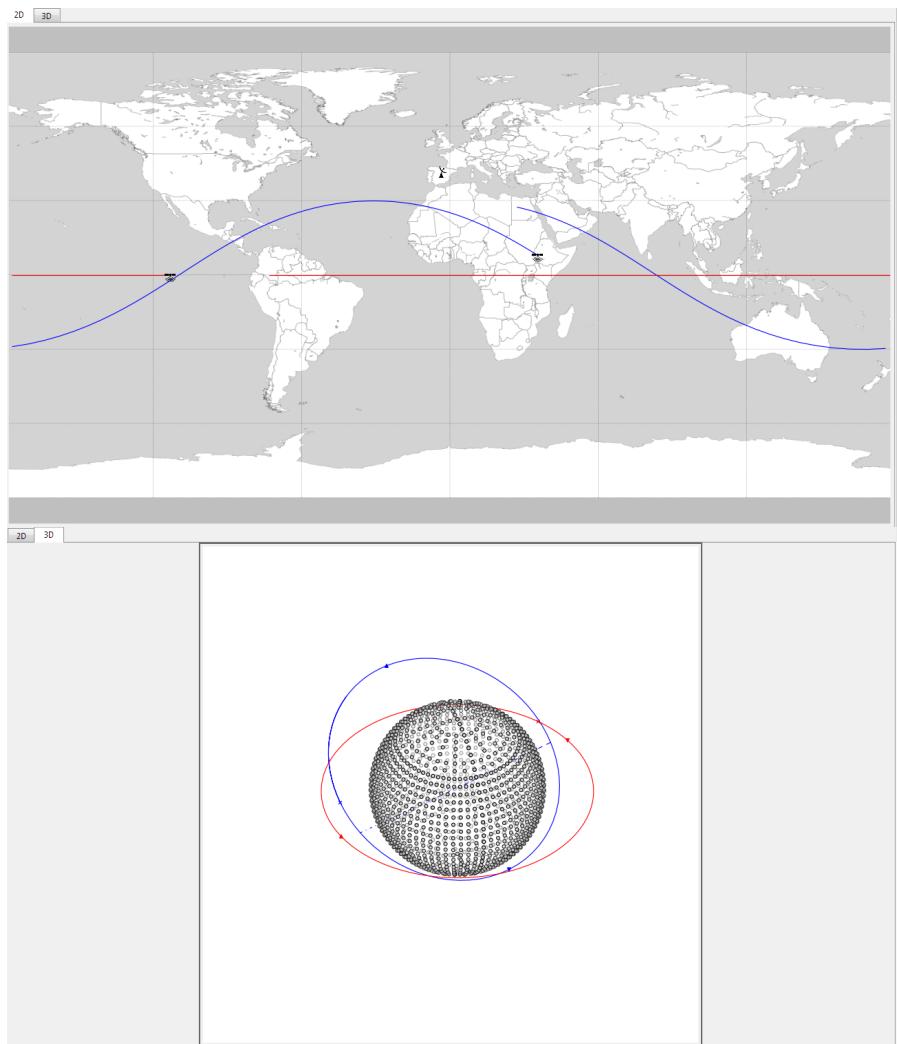


Figura 3.9: Representaciones bidimensional y tridimensional

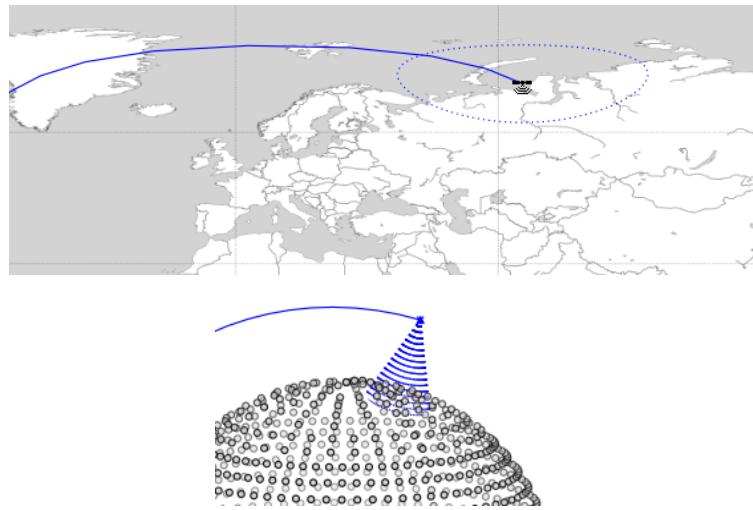


Figura 3.10: Representación del sensor

con paneles solares desplegables marca la posición de un satélite, mientras que un pequeño dibujo de una antena parabólica marca la presencia de una base u observatorio.

Sobre la representación tridimensional se marca la linea de nodos, si la hay, con una linea discontinua, que separa la parte de la trayectoria con coordenada z positiva de la parte con coordenada z negativa. Se marcan además las posiciones del satélite, del apogeo y del perigeo para facilitar la lectura, con una pequeña cruz, un triángulo apuntando hacia arriba y un triángulo apuntando hacia abajo respectivamente.

Los objetos tipo órbita únicamente son representados en tres dimensiones, por carecer de posición instantánea, y se marcan la linea de nodos, apogeo y perigeo de manera similar a las trayectorias de los sólidos. Si hay presente algún sensor, se calcula su posición, vector director y cono de visibilidad en el instante de la representación (teniendo en cuenta el tiempo propio de su objeto padre) y se representa además el cono de visibilidad sobre el gráfico tridimensional y la zona de cobertura sobre el mapa mediante una linea discontinua, si está seleccionado así en los parámetros del programa.

Acceso a datos instantáneos

A través del árbol de objetos se puede acceder a determinada información del estado actual del objeto. Haciendo click derecho, se despliega un menú contextual, uno de cuyos elementos se llama *Información*. Al pulsar sobre él, se abre una ventana que proporciona algunos datos, los fundamentales para cada tipo de objeto, que pueden resultar útiles. Por ejemplo, para un satélite se muestran datos de su posición y velocidad instantáneas, las coordenadas de su punto subsatélite y los elementos clásicos de la órbita prevista que se esperaría que siguiera en ausencia de perturbaciones. Para un sensor, aparece información de su objeto padre, su posición y su vector director en ese instante, entre otras.

Resultados

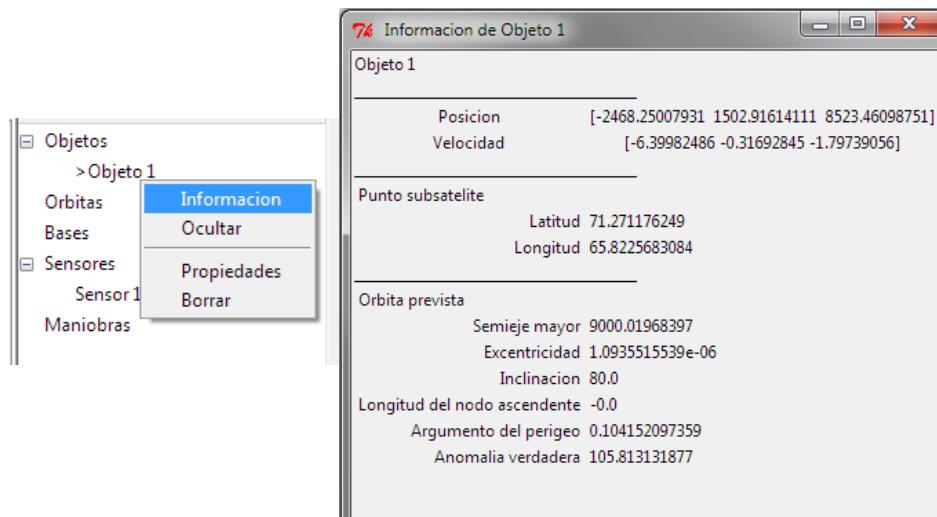


Figura 3.11: Ventana de información



Figura 3.12: Ventana de resultados

Acceso a todos los datos

Dentro del menú *Herramientas*, hacer click sobre el elemento *Resultados* permite conocer una gran cantidad de datos de la simulación. Se abre una venta, llamada ventana de resultados que permite volcar los datos de todos los objetos presentes en el escenario en un archivo, formateados para que su lectura y exportación sean sencillas.

Aparecen listados todos los objetos presentes en la simulación, y una serie de opciones dependientes del tipo de objeto. En la parte superior, debe introducirse la ruta de un archivo de texto, archivo al que se volcarán los resultados. Puede introducirse directamente o usar para ello el botón presente en la parte derecha, que abrirá un diálogo de explorador para seleccionar el archivo. Si ya hay un archivo con el nombre seleccionado, se reemplazará y se perderán los datos anteriores.

Debajo de cada objeto se presentan las distintas posibilidades específicas. Haciendo click sobre cada uno de los botones se puede seleccionar ese dato para volcarlo en el archivo, con un formato adecuado, o ignorarlo, si no interesa cono-cerlo en el escenario actual. Cuando un campo está seleccionado para aparecer

en el archivo de salida, el botón correspondiente aparece hundido.

Las posibilidades de volcado varían según el tipo de objeto en cuestión. Para un sólido aparece, entre otras, información sobre su posición a lo largo del tiempo, sobre su traza o sobre las distintas fuerzas a las que se ve sometido en cada instante. Para una órbita, se pueden volcar sus elementos clásicos o una serie de puntos de su trayectoria.

Los objetos tipo sólido tienen además cuatro botones adicionales, los 4 primeros, que no corresponden a ningún campo de datos, sino a la frecuencia con la que se van a volcar el resto. Así, se puede seleccionar si se quieren conocer los datos cada 1 punto (10^0), 10 (10^1), 100 (10^2) o 1000 (10^3), de manera que el archivo final y la cantidad de información a procesar sea menor. Puede resultar útil subir esta frecuencia si se está interesado únicamente en la evolución a largo plazo.

Al hacer click en el botón aceptar, se creará (o reemplazará) el archivo deseado con los datos seleccionados para cada objeto, donde cada fila del archivo corresponderá con un instante del tiempo. La descripción de cada columna de datos se adjunta al principio de cada objeto, y se separan los objetos por medio de líneas discontinuas.

Herramientas

El programa dispone además de algunas herramientas útiles a la hora de analizar la misión de un satélite. Deben usarse después de haber introducido los elementos necesarios y haber integrado el escenario hasta el punto deseado, accediendo a ellas a través del menú *Herramientas*. Todas estas herramientas no generan nuevos datos en el escenario, sino que usan los que ya están calculados, transformándolos y analizándolos para conseguir información útil.

Asistente de visibilidad

Sirve para comprobar si un objeto ve a otro a lo largo de su trayectoria. Se define la visibilidad como la existencia o no de una línea de visión directa entre dos objetos, es decir, si la linea recta imaginaria que los une puede trazarse completamente o queda eclipsada por algún otro cuerpo, normalmente la tierra. El cálculo de este parámetro es muy importante en determinadas aplicaciones, como constelaciones de satélites volando en formación y dependientes del intercambio de información, o misiones de observación de la tierra con un objetivo claro. Una vez que se tienen las trayectorias deseadas calculadas y los objetos específicos definidos e integrados, hacer click en el elemento del menú herramientas *Asistente de visibilidad* abre una nueva ventana que hará de menú, hub central y controlador de los cálculos.

Dentro de esta ventana debe elegirse un archivo de manera parecida a la ventana de resultados: Introduciendo la ruta directamente o pulsando el botón de la derecha, que abre un diálogo del sistema operativo para seleccionar una ruta y nombre de archivo. La salida se hace en forma de texto plano, por lo que se recomienda un archivo de formato *.txt*

Los dos campos inferiores, llamados *Objeto 1* y *Objeto 2* permiten elegir los dos objetos que se quieren analizar de entre los presentes en el escenario. Pueden ser sólidos, bases o sensores debidamente definidos e integrados. En el segundo

Herramientas

Tiempo (s)	Posicion (km)			Velocidad (km/s)		
	X	Y	Z	Vx	Vy	Vz
0.000000	-808.293780	6549.978164	1565.730463	-4.676249	-1.956180	5
100.000000	-1269.768847	6313.003911	2130.137809	-4.543382	-2.778609	5
200.000000	-1715.038277	5995.453050	2667.358434	-4.352522	-3.565655	5
300.000000	-2138.420571	5601.389052	3170.538528	-4.106106	-4.307236	4
400.000000	-2534.511821	5135.838791	3633.255442	-3.807284	-4.993853	4
500.000000	-2898.257493	4604.744579	4049.604542	-3.459886	-5.616715	3
600.000000	-3225.017551	4014.887529	4414.275297	-3.068363	-6.167856	3
700.000000	-3510.626161	3373.799983	4722.619572	-2.637737	-6.640234	2
800.000000	-3751.445168	2689.668193	4970.711249	-2.173530	-7.027826	2
900.000000	-3944.410640	1971.226563	5155.396370	-1.681690	-7.325704	1
1000.000000	-4087.071884	1227.644899	5274.333172	-1.168520	-7.530097	0
1100.000000	-4177.622423	468.410157	5326.021496	-0.640589	-7.638440	0
1200.000000	-4214.922563	-296.795692	5309.821218	-0.104648	-7.649402	-
1300.000000	-4198.513269	-1058.220086	5225.959516	0.432456	-7.562902	-
1400.000000	-4128.621222	-1806.164898	5075.526892	0.963869	-7.380108	-
1500.000000	-4006.155019	-2531.110470	4860.462100	1.482822	-7.103418	-
1600.000000	-3832.692621	-3223.836998	4583.526191	1.982711	-6.736423	-
1700.000000	-3610.460251	-3875.541747	4248.266101	2.457188	-6.283864	-
1800.000000	-3342.303071	-4477.950585	3858.968294	2.900236	-5.751561	-
1900.000000	-3031.648055	-5023.422447	3420.603120	3.306248	-5.146341	-
2000.000000	-2682.459587	-5505.045446	2938.760651	3.670094	-4.475946	-
2100.000000	-2299.188373	-5916.723452	2419.578855	3.987187	-3.748930	-
2200.000000	-1886.714372	-6253.252123	1869.665067	4.253537	-2.974555	-
2300.000000	-1450.284476	-6510.383478	1296.011764	4.465804	-2.162667	-
2400.000000	-995.445750	-6684.878282	705.907739	4.621330	-1.323573	-
2500.000000	-527.975093	-6774.545672	106.845787	4.718181	-0.467912	-
2600.000000	-53.806180	-6778.269571	-493.571939	4.755159	0.393481	-
2700.000000	421.045385	-6696.021656	-1087.729730	4.731823	1.249710	-
2800.000000	890.557796	-6528.860756	-1668.094114	4.648492	2.089950	-
2900.000000	1348.778727	-6278.918747	-2227.308471	4.506235	2.903587	-
3000.000000	1789.900052	-5949.373154	-2758.285322	4.306862	3.680344	-
3100.000000	2208.330700	-5544.406813	-3254.295200	4.052900	4.410413	-
3200.000000	2598.766774	-5069.155115	-3709.051019	3.747561	5.084570	-
3300.000000	2956.258091	-4529.641475	-4116.786935	3.394699	5.694296	-
3400.000000	3276.270337	-3932.701797	-4472.330755	2.998768	6.231877	-
3500.000000	3554.742097	-3285.898868	-4771.169023	2.564764	6.690506	-
3600.000000	3788.136061	-2597.427666	-5009.503972	2.098162	7.064359	-
3700.000000	3973.483776	-1876.012746	-5184.301667	1.604851	7.348679	-
3800.000000	4108.423393	-1130.798906	-5293.330716	1.091062	7.539828	-
3900.000000	4191.229925	-371.236445	-5335.191052	0.563290	7.635339	-
4000.000000	4220.837621	393.037598	-5309.332416	0.028210	7.633950	0
4100.000000	4196.854153	1152.320800	-5216.062248	-0.507400	7.535618	1
4200.000000	4119.566396	1896.967807	-5056.542865	-1.036747	7.341525	1
4300.000000	3989.937703	2617.512396	-4832.777890	-1.553109	7.054069	2
4400.000000	3809.596646	3304.787602	-4547.588051	-2.049917	6.676832	3
4500.000000	3580.817326	3950.042415	-4204.576590	-2.520844	6.214542	3
4600.000000	3306.491450	4545.053553	-3808.084666	-2.959878	5.673015	4
4700.000000	2990.092474	5082.230861	-3363.137237	-3.361407	5.059084	4
4800.000000	2635.632231	5554.714963	-2875.380077	-3.720289	4.380518	5
4900.000000	2247.610543	5956.465864	-2351.008664	-4.031916	3.645919	5
5000.000000	1830.958432	6282.341311	-1796.689811	-4.292282	2.864619	5
5100.000000	1390.975626	6528.163834	-1219.477025	-4.498027	2.046562	5
5200.000000	933.263147	6690.775533	-626.720644	-4.646492	1.202173	5
5300.000000	463.651834	6768.079834	-25.973918	-4.735745	0.342232	6
5400.000000	-11.872278	6759.069594	575.103759	-4.764617	-0.522276	5
5500.000000	-487.244747	6663.841127	1168.845197	-4.732711	-1.380292	5
5600.000000	-956.400676	6483.593916	1747.674043	-4.640415	-2.220834	5
5700.000000	-1413.352791	6220.615957	2304.202383	-4.488893	-3.033138	5
5800.000000	-1852.268610	5878.254897	2831.326010	-4.280073	-3.806799	5
5900.000000	-2267.545661	5460.875314	3322.316089	-4.016623	-4.531905	4
6000.000000	-2653.883745	4973.802703	3770.905964	-3.701913	-5.199171	4
6100.000000	-3006.353279	4423.254884	4171.371973	-3.339976	-5.800054	3
6200.000000	-3320.458784	3816.261771	4518.607151	-2.935453	-6.326873	3
6300.000000	-3592.196696	3160.574563	4808.186866	-2.493531	-6.772899	2
6400.000000	-3818.106705	2464.565601	5036.425505	-2.019875	-7.132452	1

Figura 3.13: Archivo de resultados

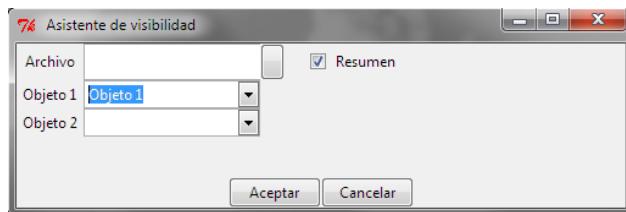


Figura 3.14: Ventana de visibilidad

objeto puede elegirse además el sol, que permitirá calcular el tiempo en que un objeto está iluminado para, por ejemplo, calcular los eclipses de un satélite o los amaneceres de una posición geográfica. El orden en que se elijan es indiferente, ya el que programa siempre hará los cálculos del elemento más cercano al más lejano, para disminuir los errores provocados por objetos cercanos a la superficie de la tierra.

La pequeña opción de resumen está activada por defecto. Si se desactiva, vuela al archivo elegido información detallada de las posiciones de los dos objetos en cada instante y calcula si se ven entre ellos o no. Si se deja activada, vuela únicamente los períodos en que los dos objetos se ven, marcando entrada, salida y duración de cada período. En cálculo interno es el mismo pero la forma del archivo de salida cambia para adaptarse a las necesidades del usuario.

Al hacer click en el botón aceptar, se lanza el análisis, calculando las posiciones de los dos objetos y comprobando después si la línea de visión está obstruida por la tierra. Si uno de los dos objetos es un sensor, comprueba además que la línea de visión esté dentro del cono de cobertura. El análisis se hace calculando las posiciones en instantes enteros, es decir, con un paso de tiempo de 1 segundo (y por tanto esta es la precisión del análisis). Si el paso de tiempo del análisis no coincide con los instantes específicos en que cada objeto tiene definida su trayectoria, se interpola linealmente entre los dos instantes más cercanos, inferior y superior, y las posiciones correspondientes a esos instantes.

Asistente de maniobras

Otro dato que resulta muy útil calcular en las primeras fases de un análisis de misión es el dimensionamiento, basta en un primer paso, de la masa final de un satélite. Resulta difícil dar un dato preciso, ya que es un proceso iterativo resultado de la interacción entre todos los subsistemas del satélite. Pero uno de los parámetros más críticos es siempre el dimensionamiento de la masa de combustible, proceso para el cual es fundamental el cálculo de las distintas maniobras que tendrá que hacer el satélite a lo largo de su vida útil. Ya sean maniobras al comienzo de su vida, para colocarse en la órbita objetivo, o maniobras de mantenimiento que se harán en momentos concretos de la vida en servicio, la masa de combustible suele ser el factor que limita y pone fecha de caducidad a las misiones espaciales comunes. Tanto para definir la misión como para dimensionar la masa de combustible y el motor, resulta fundamental conocer desde el comienzo los impulsos que habrá que realizar a lo largo de la vida del satélite y tener una estimación buena. Para ello, el programa cuenta con una herramienta que facilita el cálculo de estas maniobras orbitales y sus resultados.

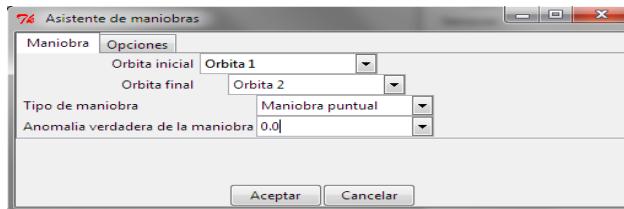


Figura 3.15: Ventana de maniobras

Al hacer click sobre el elemento *Asistente de maniobras* del menú herramientas se abre una ventana de diálogo que actúa como controlador central. Es necesario, antes de poder realizar algún cálculo, haber definido en el escenario las órbitas que van a estar implicadas y entre las cuales se hará la maniobra.

En esta ventana se permite seleccionar dos órbitas de entre las presentes en el escenario mediante los campos de *órbita inicial* y *órbita final*. Como en muchos tipos de maniobra es necesario hacer cálculos intermedios antes de poder ofrecer las distintas posibilidades, una vez que se tengan las órbitas no se mostrará nada más hasta que se seleccione el tipo de maniobra deseada. De momento, el programa sólo ofrece la posibilidad de calcular maniobras puntuales e instantáneas, pero se contempla en el futuro añadir otro tipo de maniobras y cálculos más realistas.

El tipo de maniobra llamado *Maniobra puntual* se define como una transferencia básica entre dos órbitas elípticas. Por ello, para que sea posible es necesario que la órbita inicial y la final se corten en al menos un punto. Al seleccionar este tipo de maniobra, el programa calcula los puntos de corte (puntos entre los que las órbitas se acercan más de un kilómetro) y los presenta en forma de la anomalía verdadera de la órbita inicial. Se puede elegir entonces el punto exacto donde se quiere realizar la maniobra.

El tipo llamado *Impulso puntual* se define de igual manera, como un impulso entre dos órbitas elípticas que se cortan, pero cambia el método de definición. En lugar de definir la maniobra mediante dos órbitas, se define ahora mediante la órbita inicial y el Δv de la maniobra, por lo que sólo se hará caso del campo de órbita inicial. Al seleccionar este tipo de maniobra, se permite introducir la anomalía verdadera donde se lleva a cabo el impulso y las componentes del Δv , en un sistema de referencia típico de estas aplicaciones

- Un eje en la dirección de la velocidad instantánea, llamado posigrado/antiposigrado
- Un eje en la dirección perpendicular al plano de la órbita, llamado normal/antinormal
- Un eje perpendicular a ambos, en la dirección del radio de giro instantáneo de la trayectoria, llamado radial/antiradial (aunque no siempre corresponde con el radiovector desde la tierra)

Un valor positivo corresponderá a la parte positiva del eje (en dirección de la velocidad, en el sentido de la velocidad angular de la órbita, hacia la tierra) y un valor negativo al sentido contrario o anti. La razón del uso de este sistema de ejes radica en que permite separar las maniobras según los efectos que vayan a tener

Herramientas

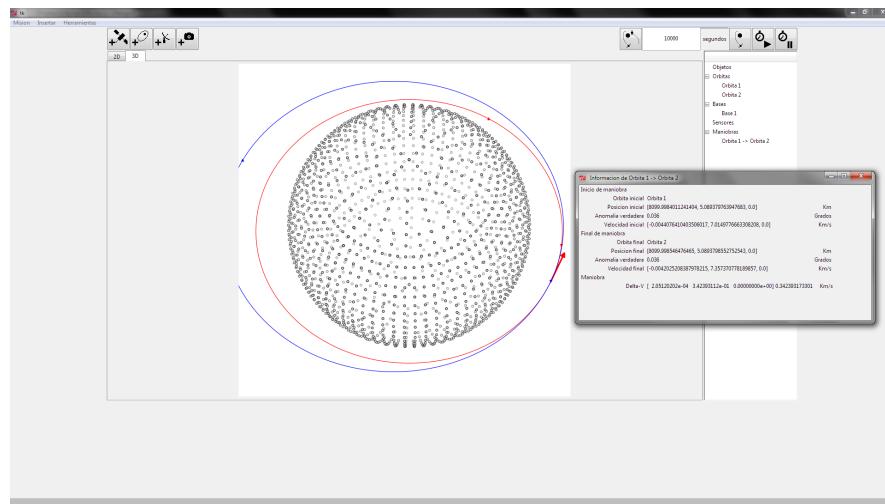


Figura 3.16: Resultados de la maniobra

en los elementos clásicos, y permite un diseño mucho más sencillo, intuitivo y productivo.

Bajo la pestaña *Opciones* se presentan parámetros que afectarán al cálculo de la maniobra y a su interacción con el escenario.

Una vez se han seleccionado todos los parámetros necesarios, hacer click en el botón aceptar calculará la maniobra elegida con todos los parámetros necesarios para definirla. Se despliega una ventana con información sobre ella y, si se ha seleccionado así, se añaden los objetos creados en el cálculo al escenario global, para su uso y revisión posteriores.

Tema 4

El entorno

En este capítulo va a tratarse el diseño y funcionamiento interno del software, explicando las funciones más importantes y haciendo énfasis en la estructura de procesos, de manera que pueda ser comprendido y complementado fácilmente por cualquier usuario que lo desee. Ejemplos específicos de la implementación del código aquí descrita pueden verse en el anexo B.

Objetivo

En el capítulo 3 se ha presentado la apariencia y uso de la interfaz gráfica diseñada para el programa, con las distintas herramientas debidamente localizadas y agrupadas, con muchas funciones complejas escondidas detrás de simples botones o asumidas en los procesos. Este hecho presenta una ventaja clara, el programa es amigable para el usuario y no requiere ni grandes conocimientos del problema ni grandes conocimientos del código, siendo rápido de usar y entender. Sin embargo, el hecho de añadir una interfaz gráfica hace que el usuario que lo deseé no vea el funcionamiento del código, ni sepa como circula el flujo de información entre las distintas funciones.

Aunque la interfaz gráfica sea suficiente para el uso que le pueda dar un usuario básico al simulador, un usuario avanzado querrá destripar el código, verificar su implementación e incluso modificarlo a su gusto. Este capítulo busca dar a conocer todas las funciones del programa a nivel de código para que todo usuario que quiera introducir modificaciones, o simplemente verificar el funcionamiento, pueda hacerlo con comodidad, sirviendo al propósito secundario de rendir cuentas de todo el trabajo realizado a lo largo de este proyecto. Además, una vez que se ha cerrado la interfaz gráfica, el entorno de Python sigue recordando las variables usadas y el progreso obtenido, por lo que el usuario conocedor de los métodos adecuados puede acceder directamente a la información que deseé.

Consideraciones iniciales

Antes de hablar del funcionamiento interno del programa es necesario hablar de su característica principal. Está escrito en Python. Por tanto, es recomendable conocer y estar familiarizado con este lenguaje de programación antes de lanzarse a modificar aspectos del código. Aspectos tan inherentes al lenguaje

como las clases o los métodos son fundamentales a la hora de programar y, de alguna manera, *moldean* la estructura a su alrededor, siendo a veces incongruentes, redundantes o inefficientes si se plantean los problemas en otro lenguaje. En esta sección se van a intentar desnudar todas las funciones del lenguaje en sí, pudiendo encontrarse los ejemplos concretos de código en el anexo B, pero la influencia indirecta de Python sin duda seguirá presente.

Otro aspecto a considerar, sobre el que el autor solicita algo de indulgencia por parte del lector, es la poca experiencia que tengo en el ámbito de la programación. La carrera de Ingeniería Aeronáutica es una carrera con un fuerte énfasis en el aspecto teórico, pero pocas ocasiones en las que adquirir experiencia práctica, y la manera de mejorar en este aspecto es de forma autodidacta. Y aunque se ha adquirido mucha experiencia a lo largo de este proyecto, el mayor que he realizado hasta la fecha, el código y la estructura de éste sin duda serán los de un novato de la programación y un novato de Python, que cualquier entendido de la materia encontrará toscos. De nuevo, pido indulgencia sobre este aspecto y, quiero creer, que si se viese el código en orden cronológico, se vería un importante avance y una progresiva adaptación y mejora del código.

El lenguaje

Ya se ha dicho que se va a intentar mantener el lenguaje aparte en este capítulo, explicando la estructura subyacente en sí, pero es necesario tener en mente algunos conceptos básicos de éste antes de poder hacer frente a este capítulo o al anexo B.

Python es un lenguaje de programación dinámico: Las variables se asignan al vuelo, cambian de clase al vuelo y ocupan espacio al vuelo, según el flujo de datos lo requiera. Por ello, el usuario que este acostumbrado a lenguajes mucho más estructurados, como Fortran, sin duda encontrará este aspecto caótico. Este dinamismo puede jugar a favor, no siendo necesario tener perfectamente milimetrados los tamaños de las variables, o pudiendo crear objetos según se necesiten (Como se hace en el programa a la hora de crear los distintos objetos de la simulación), otorgando flexibilidad y dinamismo en el flujo de información. Pero esto también tiene sus aspectos negativos. Esta flexibilidad debe ser tenida en cuenta, y las distintas variantes previstas de antemano e implementadas. Si no se tiene cuidado, puede perderse información en el camino o introducir cambios que hagan que sea imposible continuar el proceso. Debe prestarse mucha atención a las declaraciones que se hagan, de manera que el dinamismo se tenga bajo control.

Un programa no es más que una secuencia de comandos que se declaran, se ejecutan secuencialmente y dan un resultado, proceso que tiene un objetivo claro y definido. Con esta definición, cualquier código que de resultado es un programa, por básico que sea. Sin embargo, según crece la ambición del proyecto, crecen las líneas del código, hasta el punto en que la ampliación y el mantenimiento del mismo resulta inviable. Llega entonces el momento de implementar una herramienta muy útil: Las funciones. Las funciones son pequeñas cajas negras, con una entrada y una salida definidas, que se comunican entre sí por medio de variables. Estas variables pueden ser locales, que sólo cada función específica ve y controla, o globales, que se comparten entre ellas. Python permite la creación de este último tipo de variables, y permite obligar a una función

a acceder, leer y modificar esa variable global directamente, sin la necesidad de que el programa general, el *pegamento* que une todas las funciones tenga que recibir, almacenar y distribuir toda esa información. Cada función sabe a qué información tiene que acceder y donde dejar su resultado, de manera que se permite descargar esta tarea del programa principal, que se dedica sólo a coordinar las distintas funciones. Vamos a aprovechar esta propiedad del lenguaje para crear en el simulador algunas variables globales, que almacenen toda la información necesaria, a las que las funciones accederán cuando lo necesiten.

Como ya se explicó en la introducción de esta tesis, Python, al incorporar la filosofía de orientación a objetos, divide su información en distintos paquetes, definidos por las clases. En la declaración de una variable se le asigna una clase, y esto determina los distintos apartados de información que contendrá. El propio lenguaje ya tiene algunas clases predefinidas: Una variable tipo *float* contendrá información de un número real, una *string* contendrá una secuencia de caracteres, una *list* agrupará varias variables dentro de si misma, etc. Sin embargo, parte de la potencia y versatilidad del lenguaje viene dada por que permite la creación de clases personalizadas. Dentro de una clase, la información se divide en dos tipos: Los *atributos* y los *métodos*. Los atributos son las unidades básicas de información, y cada uno será una variable de la clase que queramos. Los métodos son funciones, secuencias de código, que se aplican a la clase en particular, y dan resultados propios de la clase en particular. La potencia de este sistema recae en que los atributos son variables globales dentro de cada clase, de manera que todos los métodos podrán acceder a ellos y modificarlos según lo necesiten, pero locales a cada variable de esa clase, de manera que se puedan declarar dos variables de la misma clase, cada una con unos atributos diferentes, pero cuyos métodos sean el mismo y den el mismo resultado. Por poner un ejemplo práctico, que se espera resulte aclaratorio, en el simulador se pueden crear variables de tipo sólido. Uno de los atributos será el integrador que se quiere usar, de manera que podamos definir un sólido con un Adams-Bashforth y otro idéntico, pero con un Runge-Kutta. De esta manera, al ejecutar el método *integrar()*, propio de la clase sólido, cada variable se integra teniendo en cuenta el integrador que se quiere usar en cada caso, produciendo resultados distintos.

Este principio se usa en el programa para dar cierta uniformidad a las entradas y los resultados, al tiempo que se mantiene la flexibilidad deseada y el orden necesario para que el código sea lo suficientemente claro.

Hay que mencionar la manera sencilla que tiene Python de acceder a toda esta información: *nombre-de-la-variable.atributo-o-método*. Así, siguiendo el ejemplo anterior, para una variable llamada Objeto1 de tipo sólido, *Objeto1.integrador* dará acceso a leer o modificar el integrador que se quiere usar, mientras que *Objeto1.integrar()* lanzará la integración con el integrador que este indicado en ese momento en sus atributos.

Estructura básica

El código está dividido a lo largo de varios archivos, agrupado temáticamente, de manera que el acceso y mantenimiento de distintas partes sea sencillo y asequible. *gui.py* contiene todas las funciones necesarias para el funcionamiento de la interfaz gráfica y la cooperación entre variables. *integracion.py* contiene

toda la información de los sólidos, así como varias funciones para representar distintas trayectorias. *orbita.py* define el resto de clases de objetos, incluyendo algunas funciones de análisis sencillas. *mision.py* contiene el escenario general, así como algunas funciones de análisis generalistas. *cuaternios.py* permite la incorporación de estos elementos a los cálculos y *atmosfera.py* contiene los cálculos de la atmósfera que, por su complejidad, se han separado del resto de las fuerzas.

Dos variables resultan fundamentales en el entorno. El escenario y los objetos. Ambas son de tipo global, por lo que todas las funciones que lo necesitan pueden acceder a ellas cuando lo requieran.

La variable que contiene la información del entorno se llama *escenario* y es del tipo *prescenario*. Dentro de sus atributos se guardan distintos parámetros generales necesarios, como la fecha inicial o las perturbaciones a aplicar. Los atributos más básicos son, entre otros:

- *escenario.reptierra*, que permite elegir entre una tierra esférica o elipsoidal a la hora de calcular los radios
- *escenario.modeloatmosfera*, que permite elegir entre usar o no usar el modelo de densidad Jacchia explicado en el capítulo 2
- *escenario.colisiones* permite activar las colisiones, que cortaran la integración si la trayectoria se interseca con la tierra
- *escenario.modelogravedad*, que permite cambiar entre un modelo gravitatorio esférico o incluir el efecto de J2
- *escenario.tiempo*, que guarda la fecha inicial global del escenario en forma de una lista de 6 valores, ordenados de mayor cantidad de tiempo a menor

$$(\text{yyyy} , \text{mm} , \text{dd} , \text{hh} , \text{mm} , \text{ss})$$

Todos estos parámetros se guardan, además de en la variable, en el archivo *config.cfg*, de manera que se recuerdan de una sesión para otra.

La otra variable básica es la que contiene todos los objetos del entorno, llamada *objetos* y del tipo *diccionario*. Un diccionario es un tipo de variable natural de Python, que permite guardar en el mismo sitio una lista de variables, a cada una de las cuales se le hace corresponder un nombre al declararla de manera que, sabiendo el nombre, se puede acceder a toda la información de la variable fácilmente. Se usa este sistema por la facilidad de manejo, mantenimiento y acceso que presenta frente a otros. Así, si se ha definido un objeto llamado NuevoObjeto, sea del tipo que sea, *objetos[“NuevoObjeto”]* proporciona acceso a los atributos específicos del objeto y a sus métodos de clase.

Crear un objeto nuevo es tan fácil como crear un objeto de la clase que se desee y luego incluirlo en el diccionario de objetos usando su nombre como referencia. El programa usa esta técnica, nombrando a los distintos objetos con su clase seguida del primer número natural que no esté usado. Este nombre se puede naturalmente cambiar a posteriori.

Objetos

Cada clase de objeto definida tiene sus propios atributos y métodos, pensados para que cada uno de ellos pueda actuar de manera autónoma. Vamos a explicar

las funciones más importantes de cada clase, y el tipo de atributos que esperan para poder funcionar correctamente.

La clase **sólido** es la base sobre la que se construye el proceso de integración. Un sólido viene definido por su posición (*solido.pos*), su velocidad (*solido.vel*) y su tiempo propio (*solido.t*), además de varios parámetros más que recogen sus propiedades. Cada uno de estos tres atributos es una matriz bidimensional, con las filas correspondiendo a distintos instantes de tiempo y las columnas a distintas coordenadas. Otros dos parámetros fundamentales en la integración son *solido.int* y *solido.At*, que contienen el integrador que se va a usar y el paso de tiempo deseado. En *solido.tmax* se guarda el límite de tiempo hasta el que se quiere integrar, medido desde el instante inicial. Así, si el tiempo inicial definido del escenario son las 12:00:00 y un sólido definido tiene un atributo *tmax* con valor 120, la próxima vez que se llame a la integración esta se parará a las 12:00:02 cuando el tiempo local del objeto alcance este valor, y si se quiere seguir el proceso de integración será necesario definir otro valor de *tmax*.

Al lanzar el proceso de integración, se lanza un bucle que, con la información que tiene en ese momento, calcula en cada paso la nueva posición, velocidad y tiempo, y actualiza los atributos correspondientes con la nueva información. Este bucle continua hasta que alcanza condiciones de parada, es decir, se alcanza el tiempo objetivo o se choca contra la tierra (Si las colisiones están activadas)

Cálculo de fuerzas. El cálculo de la resultante se hace en el método *solido.sumafuerzas*. Como alguno de los métodos numéricos requieren poder calcular las fuerzas en los puntos de la trayectoria pero también en puntos fuera de esta, se usan la posición y velocidad como parámetros de entrada, para poder otorgar esta flexibilidad exigida. Tanto las entradas como la salida deben ser vectores de tres coordenadas, para que el resto de las operaciones se realicen correctamente. Se calculan las fuerzas gravitatoria y aerodinámica, se suman en coordenadas ECI y se devuelve el vector resultante.

<code>solido.sumafuerzas</code>	
Entradas	\vec{x}, \vec{v}
Salidas	\vec{F}
Parámetros necesarios	escenario.modeloatmosfera sólido.propiedades.balístico escenario.modelogravedad

Cálculo de f: El cálculo del término de la derecha de la ecuación 2.46 se hace en *solido.f*. Con las fuerzas calculadas, sólo quedaría ordenar los términos. Sin embargo, muchos integradores requieren no sólo calcular $f(\mathbf{y}^n, t^n)$, sino que también necesitan calcular el resultado en otros puntos de la trayectoria o en puntos y tiempos intermedios. Por ello se crean las entradas, para poder flexibilizar el cálculo. Paso es un número entero que indica el punto de la trayectoria deseado, y si es negativo empezando por el final. Así *paso* = -1 hará que se

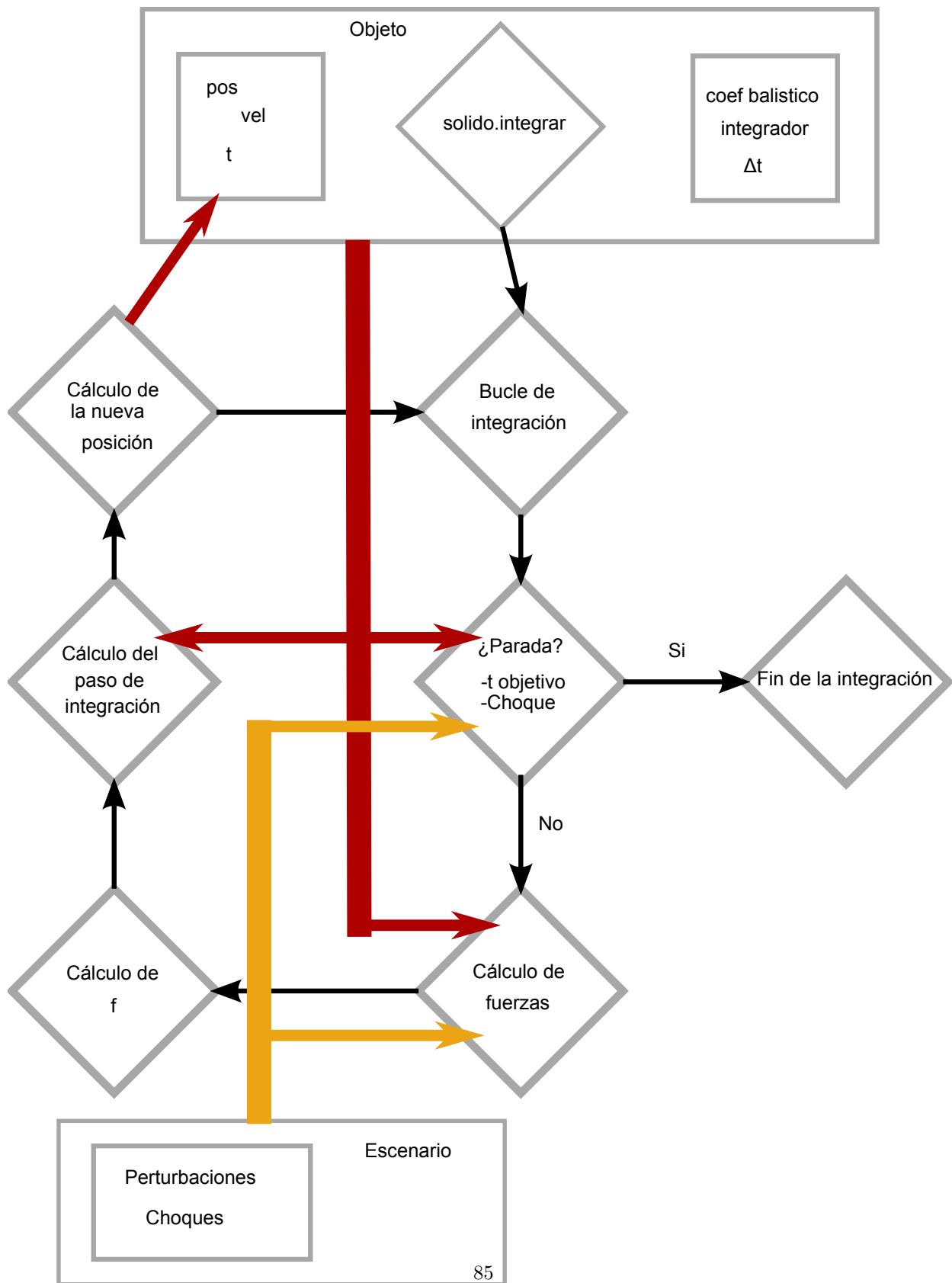


Figura 4.1: Flujo del bucle de integración

usen \mathbf{y}^n , t^n y $\text{paso} = -3 \mathbf{y}^{n-2}, t^{n-2}$. Punto y tiempo permiten introducir además alguna desviación adicional sobre estos puntos, como las requeridas cuando se usa un Runge-Kutta.

solido.f

Entradas	paso, tiempo, punto
Salidas	\mathbf{f}
Parámetros necesarios	-

Cálculo del paso de integración. Varios métodos se ocupan de este paso, dependiendo del integrador que se esté usando para el sólido. Cada uno de ellos tiene definidas en su interior las llamadas necesarias a \mathbf{f} y la combinación correcta que debe hacer entre ellas. Al final, devuelven el incremento de \mathbf{y} (no hay que olvidar que es un vector compuesto por la posición y velocidad) que hay que añadir a la última posición para hallar la nueva. Dentro de esta categoría, *solido.rungekutta4*, *solido.adamsbashforth4* y *solido.predictorcorrector4* tienen la misma estructura y comportamiento.

solido.[integrador]

Entradas	Δt
Salidas	$\Delta \mathbf{y}$
Parámetros necesarios	-

Cálculo de la nueva posición. El proceso que se encarga de esto es el propio *solido.integrar*. Una vez que recibe el paso de integración, se suma a la última posición y se añade a las matrices de posición y velocidad del sólido. Sin embargo, este método también hace de coordinador del resto del proceso de integración, hace las llamadas necesarias y controla el flujo. Se crea un bucle temporal, con el paso de tiempo entre puntos, y en cada paso se calcula la nueva posición mediante el método correspondiente. El bucle progresará comprobando las condiciones de parada en cada paso, hasta que se recibe un positivo y se detiene la integración.

solido.integrar

Entradas	-
Salidas	<i>solido.pos</i> , <i>solido.vel</i>
Parámetros necesarios	<i>solido.integrador</i> <i>solido.At</i>

Por último, el proceso de integración finaliza cuando se alcanzan condiciones de parada. A cada paso se comprueba si el tiempo alcanzado ha superado al objetivo, o si el radio de la posición es menor que el radio de la tierra en esas coordenadas. Si alguna de estas dos condiciones se cumple, se interrumpe el bucle de integración y se puede realizar otra tarea.

solido.parada

Entradas	-
Salidas	True/False
Parámetros necesarios	solido.tmax escenario.reptierra

La clase **orbita** es una clase auxiliar que se usa para poder realizar varios cálculos intermedios. Una órbita está definida por sus elementos clásicos, que se almacenan en forma de vector en el atributo *orbita.parametros*, y sobre los cuales se realizan el resto de cálculos. Es importante el orden ya que los procesos buscan en la dirección específica, que corresponde a

$$(a, e, i, \Omega, \omega, \theta)$$

Nótese que se usa la anomalía verdadera, frente a la excéntrica, la media o el tiempo de paso por el perigeo. El semieje mayor se guarda en kilómetros, y los distintos ángulos en grados. La excentricidad debe ser además menor de 1., para que corresponda a una órbita elíptica.

Una vez que se tiene la órbita definida, puede accederse a varios métodos útiles. El primero es *orbita.par2xv*, que permite transformar los elementos clásicos en posición y velocidad de los ejes de referencia, de acuerdo con las ecuaciones teóricas del capítulo 2. Accede a los parámetros de la órbita definidos y devuelve dos vectores tridimensionales con las coordenadas de la posición y velocidad en el sistema ECI J2000, en unidades de kilómetros y kilómetros por segundo.

orbita.par2xv

Entradas	-
Salidas	\vec{x}, \vec{v}
Parámetros necesarios	orbita.parametros

El proceso contrario también es posible mediante la función *orbita.xv2par*, que define la órbita a partir de una posición y velocidad. Se llama a la función con dos vectores tridimensionales en el sistema de referencia y unidades anteriores, y el proceso define los elementos clásicos de la órbita a partir de ellos, mediante el algoritmo descrito en el capítulo 2.

orbita.xv2par

Entradas	\vec{x}, \vec{v}
Salidas	orbita.parametros
Parámetros necesarios	-

orbita.orbita calcula, en las coordenadas de referencia, los puntos de la órbita. Predeterminadamente calcula 200 puntos (con la anomalía verdadera distribuida uniformemente entre ellos), pero este valor se puede cambiar como se deseé. Devuelve una matriz bidimensional, de forma idéntica a las que devolvía el proceso de integración de los sólidos.

`orbita.orbita`

Entradas	-	
Salidas	orbita.pos	
Parámetros necesarios	orbita.parametros	

Otra función interesante es la que permite calcular si un punto geométrico pertenece a una órbita. A partir de un vector tridimensional correspondiente a una posición espacial, se divide la longitud del arco en 1000 puntos y se comprueba la proximidad de cada punto con el objetivo, con una resolución de 1 Km. Si la distancia es menor de este valor, se devuelve además de un valor lógico *True* la anomalía verdadera del punto dentro de la órbita.

`orbita.xenorbita`

Entradas	\vec{x}	
Salidas	True/False, ϑ	
Parámetros necesarios	orbita.parametros	

También hay definidas varias funciones pequeñas pero igualmente importantes, como *orbita.periodo*, que calcula el periodo de la órbita mediante la tercera ley de Kepler, u *orbita.plano*, que devuelve un vector unitario perpendicular al plano de la órbita.

La clase **sensor** presenta algunas funciones interesantes de cálculo geométrico. Como ya se ha explicado, se modelan los sensores como objetos auxiliares, unidos a otro objeto padre cuyo nombre se define en el campo *sensor.objeto*. El tipo de apuntamiento definido se guarda en *sensor.actitud* y, en caso de que sea una dirección fija, en *sensor.vector*. El nombre del objeto padre debe ser el mismo que se ha definido en el escenario general, y el apuntamiento debe ser “Nadir”, “Cenit” o “Direccion fija”.

Como es un objeto dependiente de su padre, su posición y tiempo siempre van a estar ligadas a él. *sensor.actualizar* accede al objeto padre dentro de la variable global y copia su posición, actualizando las propiedades del sensor en ese instante a partir de ella.

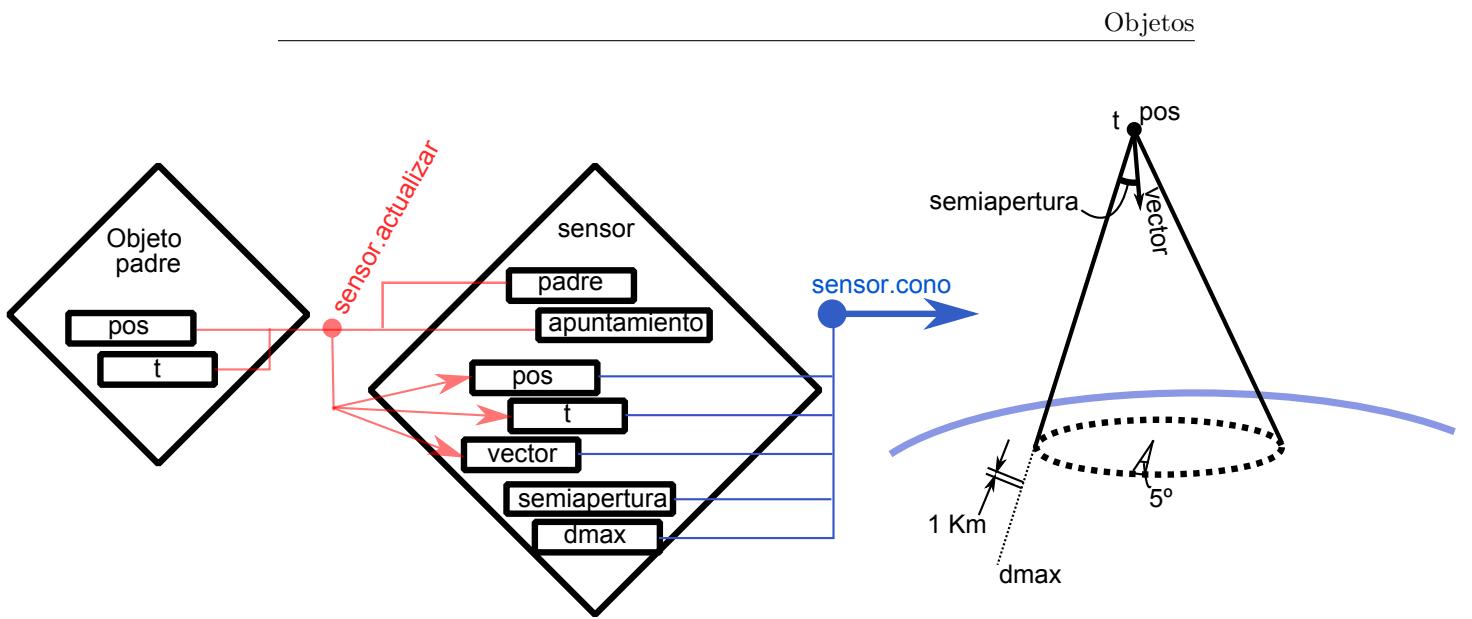


Figura 4.2: Estructura del sensor

sensor.actualizar

Entradas	Objetos (global)
Salidas	sensor.pos
	sensor.t
Parámetros necesarios	sensor.vector
	sensor.actitud

Las propiedades del cono de visibilidad con el que se modelan los sensores se definen en los atributos *sensor.semiangulo* y *sensor.dmax*, en unidades de grados y kilómetros.

sensor.cono calcula los puntos del extremo del cono de visibilidad del sensor. Lanza un rayo que forma un ángulo correspondiente a la semiapertura con el vector director, y calcula una serie de puntos separados por 1 kilómetro siguiendo esta linea. Hace lo mismo para distintos rayos del perímetro del cono, girando alrededor del vector director en intervalos de 5 grados. Se obtiene una lista de puntos que son, o bien la intersección entre cada uno de los rayos del cono y la tierra, o bien los puntos finales de cada uno de los rayos del cono cuando se llega al alcance máximo. Esta lista de puntos toma la misma forma de matriz bidimensional definida anteriormente, con cada fila correspondiente a un punto distinto y cada columna a una coordenada cartesiana del radiovector.

sensor.cono

Entradas	-
Salidas	<i>p</i>
Parámetros necesarios	escenario.reptierra

Puede darse el caso, sobre todo para satélites en órbitas más altas, de que la apertura del sensor sea lo suficientemente grande como para que abarque toda

la tierra. La zona de cobertura no queda entonces limitada por el ángulo del sensor, sino por el horizonte de la tierra en cada punto. *sensor.traza* se encarga de buscar este horizonte y *sensor.dibujartraza* de representarlo en unos ejes. Para el cálculo, se parte del punto subsatélite del sensor y se calcula el punto más al este que ve, buscando condiciones de tangencia entre los vectores implicados. A continuación, se hace un barrido bidimensional, imitando un método de rejilla calculando para cada latitud el punto más alejado en la horizontal que queda dentro del horizonte. Se recorren así todas las latitudes y longitudes, con una resolución espacial de 0.5 grados, de manera que se dibuja una curva cerrada, en el sentido contrario a las agujas del reloj. Se devuelve la lista de los puntos, en la forma de matriz bidimensional ya conocida.

sensor.traza

Entradas	-	
Salidas	p	
Parámetros necesarios		escenario.reptierra

Herramientas

Hasta ahora hemos definido funciones que permiten, a partir de ciertos parámetros iniciales, crear información referente a los distintos objetos y a su evolución en el tiempo. Sin embargo, para alcanzar el máximo potencial de todos estos datos, es necesario crear un conjunto de herramientas que permitan analizarlos, ordenarlos y procesarlos, de forma que se transformen en todo tipo de información útil para el usuario.

En primer lugar, vamos a hablar de las herramientas de visualización de los datos. Los valores numéricos son muy adecuados cuando se quiere realizar cálculos detallados y precisos, pero muchas veces la representación de esos datos es mucho más valiosa e intuitiva para el usuario. Por ello, el programa cuenta con funciones apropiadas para ello.

Una vez que se tiene calculada, la trayectoria de un sólido viene dada por sus coordenadas en el sistema de referencia. Y aunque estas coordenadas, sin más análisis, puedan resultar útiles para algunas aplicaciones, en la mayoría de los casos convendrá situarlas respecto a la tierra. Para ello, se definen un par de funciones que permitan la transformación a coordenadas geográficas, referenciadas a la tierra. *xtogeo* permite hallar la proyección de un punto sobre la tierra, resolviendo la ecuación 2.29 cuando sea necesario.

xtogeo

Entradas	\vec{x}, t	
Salidas	latitud, longitud	
Parámetros necesarios		escenario.reptierra escenario.tiempo

La función *trayectoriageo* incorpora esta función para poder hallar la traza de una trayectoria completa. Además, “rompe” la traza en varios segmentos, de manera que cada paso entre las longitudes de -180 y 180 sea una linea diferente. Así se permite obtener una representación más limpia, o incluso controlar cuantos historial de trazas se quiere representar.

trayectoriageo

Entradas	pos, t
Salidas	trazas, número de trazas
Parámetros necesarios	-

También se incorpora el proceso inverso, de manera que se pueda hallar la posición tridimensional de objetos sobre la superficie de la tierra en un instante dado. Se calcula el radio correspondiente a las coordenadas geográficas y se escribe la posición en forma vectorial, teniendo en cuenta el giro de la tierra.

geotox

Entradas	latitud, longitud, tiempo
Salidas	\vec{x}
Parámetros necesarios	escenario.reptierra escenario.tiempo

Una vez se tienen las proyecciones deseadas, *dibujartraza* permite representar la traza sobre un plano bidimensional y *dibujarorbita* sobre unos ejes tridimensionales. Aunque ambas están pensadas para integrarse directamente en la interfaz gráfica diseñada, puede extraerse la salida si se introduce como argumento en ambas un objeto de **matplotlib** adecuado, ejes2D y ejes3D respectivamente. Además de la representación básica, incorporan varios procesos para dibujar rasgos de las trayectorias, como la posición del apogeo o del nodo ascendente.

dibujartraza

Entradas	x, t, color, ejes
Salidas	Plot 2D de x
Parámetros necesarios	-

dibujarorbita

Entradas	x, t, color, ejes
Salidas	Plot 3D de x
Parámetros necesarios	-

Además de las encargadas de hallar las distintas representaciones, se definen muchas más funciones que se ocupan de diversos aspectos de la simulación, como controlar y actualizar los objetos o hacer interactuar los distintos modelos con el proceso. Y aunque todas son importantes, muchas se refieren a aspectos del funcionamiento interno del programa, por los que se van a explicar aquellas más interesantes de cara al uso que el usuario final puede realizar.

Para transformar una fecha en su fecha juliana se usa la función *juliana*, dentro de *mision.py*. Devuelve un número real correspondiente al día juliano requerido, con la fracción de día transcurrida en los decimales. La entrada es una lista de 6 valores, correspondiente a la fecha y hora ordenadas de mayor unidad a menor

$$(yyyy, mm, dd, hh, mm, ss)$$

juliana

Entradas	t
Salidas	JD
Parámetros necesarios	-

La fecha juliana influye en la posición relativa de los distintos cuerpos celestes. Por ejemplo, *girotierra* calcula la rotación de la tierra respecto al sistema ECI y permite geolocalizar los satélites. Su entrada son los segundos transcurridos desde el comienzo de la simulación, ya que incorpora automáticamente la fecha inicial a los cálculos, y devuelve el ángulo que hay que girar desde el eje x de la referencia inercial para hallar el punto de latitud 0 y longitud 0 sobre la tierra.

girotierra

Entradas	t
Salidas	Ángulo
Parámetros necesarios	escenario.tiempo

La fecha también influye en la posición relativa del sol en los ejes inerciales. *posicionsol* se encarga de, dada una fecha juliana, devolver el vector de posición del sol en ese instante en la referencia J2000. El algoritmo que se usa para este cálculo está descrito en el capítulo 2.

posicionsol

Entradas	JD
Salidas	\vec{x}_{sol}
Parámetros necesarios	-

interseccionorbitas es una herramienta potente que permite comprobar si dos órbitas distintas se cruzan en el espacio. Su entrada son dos objetos de tipo órbita debidamente definidos y su salida son dos listas separadas, cada una de las cuales contiene la anomalía verdadera de los puntos de cruce, si los hay, referidos a cada una de las órbitas. Si las órbitas están contenidas en planos distintos, se comprueban únicamente los puntos de la recta de intersección entre ellos, para ahorrar tiempo de cálculo. Si son coplanarias, recorre la anomalía verdadera buscando cambios de signo en la resta entre radios.

interseccionorbitas

Entradas	órbita 1, órbita 2
Salidas	[θ en órbita 1], [θ en órbita 2]
Parámetros necesarios	-

Para realizar los distintos giros implicados en el problema, se usan cuaternios combinados con las operaciones debidas. La función *ang2cua* es capaz de, dado un vector y un ángulo (en grados), devolver el cuaternion que representa un giro del ángulo alrededor del vector. La función *giro* lleva esta operación un paso más allá, devolviendo el resultado de girar el vector \vec{u} alrededor de \vec{v} un ángulo ϑ .

giro

Entradas	$\vec{u}, \vec{v}, \vartheta$
Salidas	\vec{u}'
Parámetros necesarios	-

En algunos momentos es necesario calcular si dos posiciones del espacio se “ven” entre ellas, es decir, si se puede trazar una linea recta que las une sin que se interseque con nada, o si por el contrario la línea de visión está interrumpida. Esta función puede resultar muy útil a la hora de calcular los pasos de un satélite por encima de una posición geográfica o los satélites que provoca la sombra de la tierra. De realizar estos cálculos se encarga la función *visibilidad* que, dadas dos posiciones (*pos1* y *pos2*) en forma de vector calcula si la linea que las une interseca a la tierra.

Para ello, calcula un vector director entre las posiciones y elige de entre las dos la más cercana a la tierra, en un intento de minimizar los errores. Desde ahí lanza un rayo siguiendo el vector y evalúa en mil puntos a lo largo de la linea el radio de la posición, comparándolo al de la tierra en ese punto. Si ninguno de los radios es menor que los de la tierra, se considera que hay visibilidad y devuelve un valor **True**. En caso contrario, devuelve un **False**.

Incluye además un par de evaluaciones que cortan el bucle si se alcanzan determinadas condiciones en las que se considera que la tierra ya no puede estar en medio, con el fin de ahorrar en tiempo de computación.

visibilidad

Entradas	$\vec{x_1}, \vec{x_2}$
Salidas	True/False
Parámetros necesarios	escenario.reptierra

Interfaz gráfica

Se han explicado varias de las funciones implicadas en la simulación, de manera que, si el usuario lo desea, pueda ampliarlas, modificarlas, o usarlas como un paquete de funciones independientes en sus propios programas. Todas ellas, una vez se han creado todos los parámetros necesarios, son capaces de funcionar autónomamente sin problemas. Sin embargo, si no se tiene software anterior, el proceso de recopilar, unificar y combinar todas las funciones necesarias puede resultar largo y tedioso. Por ello, aprovechando que la interfaz gráfica del programa ya está definida, se va a explicar por encima la estructura de esta, para el usuario que, aunque quiera ampliar las posibilidades de la simulación, quiera aprovechar la distribución de ventanas ya creada.

La interfaz gráfica se ha creado usando el módulo de Tkinter, cuyo uso está muy extendido entre la comunidad de Python. Este módulo favorece la creación de interfaces gráficas con una estructura altamente jerarquizada, donde los elementos de la pantalla se dividen hasta los elementos más básicos, y se crea una fuerte relación de padres-hijos entre los distintos niveles de información. Se crean ventanas, marcos y regiones con la distribución que se desea, que a su vez incorporan elementos llamados *widgets* que pueden variar entre una simple etiqueta que muestre información como a una botón desplegable que de a elegir entre varias opciones.

El proceso de creación de los *widgets* tiene dos fases: En primer lugar es necesario crearlos, definiendo sus ascendencias, propiedades y su manera de interactuar con el usuario. En segundo lugar, es necesario distribuirlos por la geometría, asignarles un lugar dentro de las ventanas. A la hora de afrontar este último paso, el módulo permite dos posibilidades bien diferenciadas. Por un lado, aplicarle un método *.grid* a un elemento dividirá el elemento padre en varias filas y columnas, permitiendo seleccionar el posicionamiento exacto en la fila y columna que queramos. Por otra parte, se puede aplicar en su lugar un método *.pack*, que “lanzará” al elemento en una dirección hasta los límites de la ventana. Ambos métodos tienen sus usos, siendo *.grid* muy útil cuando se tienen muchos elementos parecidos y regulares cerca, y resultando *.pack* más adecuado cuando se tienen pocos elementos y una geometría más irregular. Es necesario tener en mente, sin embargo, que la geometría de un mismo parente sólo puede ser manejada por uno de los dos a la vez, por lo que es conveniente decidir con rigor cual se quiere.

Varias son las opciones de interacción software-usuario que ofrece este módulo. Una *label*, una etiqueta, permite mostrar información para que el usuario la reciba. Un botón es una manera automatizada de implementar el lanzamiento de una función, sin tener que hacer la llamada vía, por ejemplo, línea de comandos. Un *radiobutton* permite seleccionar o deseleccionar una opción determinada, y un *checkboxbutton* permite elegir al usuario un parámetro, desde una

lista cerrada de valores admitidos. Todos estos *widgets* tienen métodos y funcionamiento distintos, y conviene estar familiarizado con ellos antes de modificar el código. Merecen especial mención los objetos tipo *entry*, la manera más flexible que tiene el módulo de que el usuario introduzca información y que permite al usuario escribir libremente en un campo. Sin embargo, al recuperar la información esta se guarda en una variable tipo *string*, y si se quiere operar con ella numéricamente (si, por ejemplo, la entrada corresponde a una coordenada de la posición de un sólido) es necesario convertirla antes al tipo de variable adecuado, un *float*. Como este proceso puede ser una fuente importante de errores, puede resultar conveniente introducir protecciones que impidan al usuario introducir información del tipo incorrecto. Por falta de recursos, se ha evitado introducir en el programa este tipo de protecciones, a la espera de que el tipo de usuarios de este software, de alto nivel y preparación técnica, pueda filtrar esta clase de errores por si mismo.

La base del programa es un objeto de tipo *Tk*, intrínseco de Tkinter, llamado **root** sobre el cual se empiezan a definir el resto de objetos, con su geometría calculada y modificada para que se ajuste a la resolución de la pantalla del ordenador que esté corriendo el programa. Dentro de este objeto se crea un marco llamado *framebase*, que contiene todos los elementos del programa excepto la barra de herramientas superior y la barra de estado inferior. Dentro del marco, se definen otros cuatro marcos, que separan las dos zonas de botones, la zona de dibujo y el árbol de objetos.

La barra de estado es un objeto de la clase creada *barrastatus*. Permite escribir y mostrar un texto, borrarlo o, al utilizar el método *completado*, marcar el término de otra función escribiendo un texto y haciéndolo desaparecer poco después.

En la barra de menús se definen varios elementos, agrupando distintas funciones por grupos funcionales y creando distintas *cascadas* de elementos, cada uno de los grupos de funciones. Cada elemento dentro de estos grupos lleva asociada una función, que controla lo que sucede cuando se hace click sobre dicho elemento.

La zona del árbol se agrupa en un marco llamado *areaarbol*, que a su vez maneja dos objetos distintos. Por un lado, el árbol en si, un objeto predefinido de Tkinter que se maneja y mantiene con la función *actualizararbol*. Por otro lado, se define una nueva clase llamada *menuderechoarbol*, que controla el pequeño menú emergente que se crea cuando se hace click derecho en esta zona, cuyas funciones se definen en los métodos de la clase. Por ejemplo, *menuderechoarbol.informacion* controla la pequeña ventana que muestra información sobre el objeto deseado (distinta para cada tipo de objeto), y *menuderechoarbol.propiedades* abre la ventana de definición del objeto, por si se ha cometido algún error al crearlo.

El área de dibujo se engloba en un marco llamado *areadibujo*, dentro del cual se crea un objeto de tipo *notebook*, un objeto de Tkinter que permite cambiar con facilidad entre varios marcos distintos. A continuación, se crea un marco para cada representación, *frame2D* y *frame3D*, donde se definen sendos objetos

tipo *canvas* con sendos ejes de dibujo, *ejes2D* y *ejes3D*. Los *canvas* son objetos predefinidos de Tkinter, aunque es necesario modificarlos ligeramente para que se adapten bien a las librerías de matplotlib.

Los botones se definen en una de las dos zonas de botones, dependiendo de su función. A la hora de crear un botón hay que proporcionarle un *handler*, una función a la que llamar cuando el usuario pulse sobre él. Cada botón tiene por supuesto un handler distinto, pero con todos se ha intentado seguir una regla de denominación parecida al nombre del botón para facilitar su mantenimiento. Además, Tkinter permite asignar un archivo de imagen a cada botón, de manera que aparezca en lugar de texto plano en la representación del botón en la ventana. Todos los archivos se encuentran en la subcarpeta *./graficos/iconos* y se guardan en formato *.gif*, con un nombre que hace referencia a su función.

Cuando se hace click en algún elemento que requiere que se abra otra ventana esta se crea con su propia geometría y elementos, pero siempre supeditada a root. Cada una de estas ventanas tiene geometrías diferentes con sus propios marcos y *widgets*.

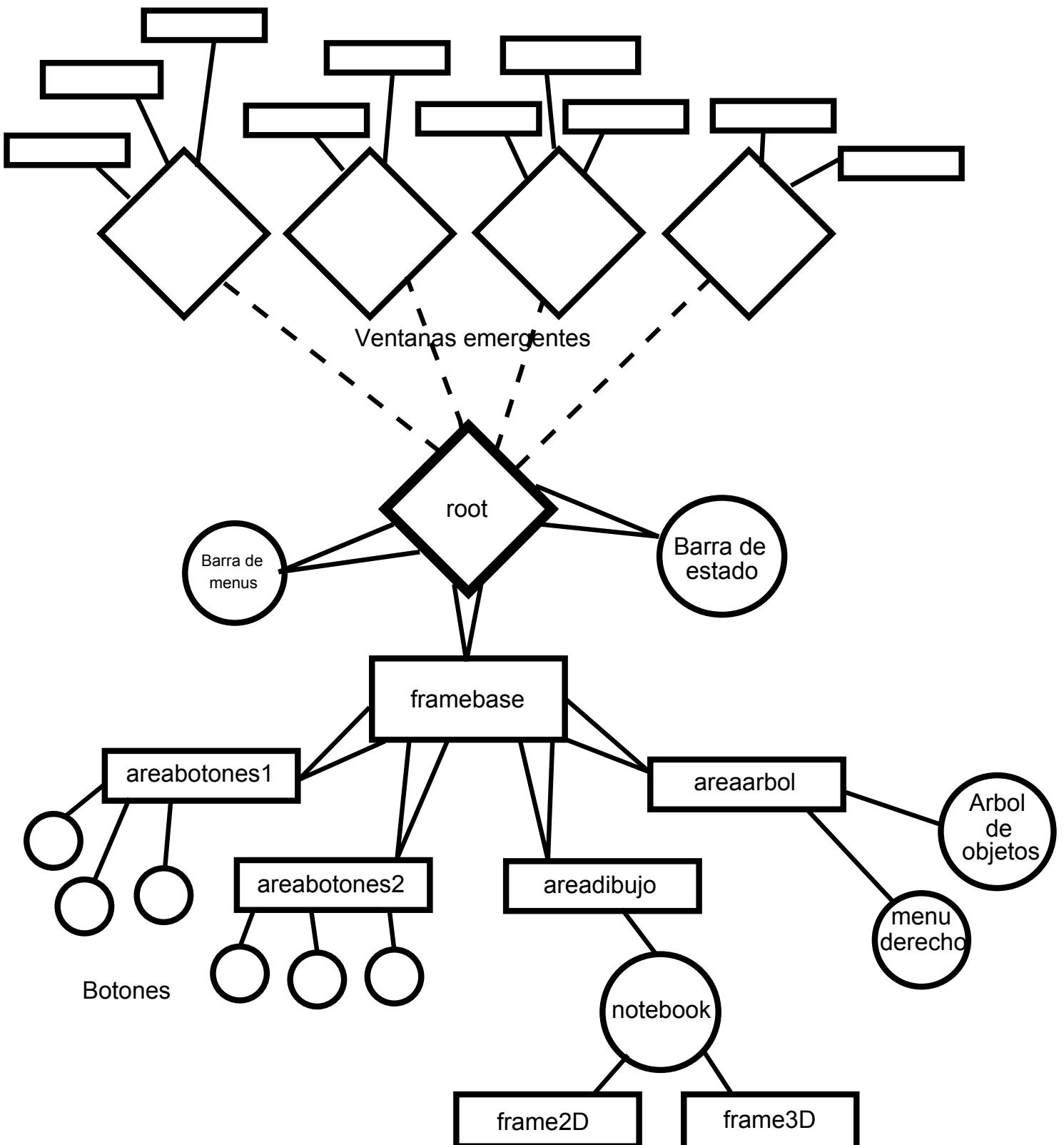


Figura 4.3: Estructura de root

La ventana de definición de los distintos objetos se controla desde la función *ventanapropiedadessolido*, que crea una nueva ventana llamada *ventanaprop*. Como esta es la misma para todos los tipos de objeto contemplados en el simulador, es necesario disponer de algún mecanismo flexible que cambie según el objeto con el que se la llame. Algunos widgets son comunes para todos los objetos, como los botones inferiores o los campos de personalización del nombre y el color. La adaptación del resto de la ventana viene dada por el botón desplegable *selectiposolido* y la función *cambiotiposolido*, que crean un marco llamado *framesolido* con los campos necesarios para cada tipo de objeto. Es este marco *framesolido* el que permite la flexibilización de la ventana y el que contiene los widgets específicos de cada objeto.

Al llamar a la ventana se hace una copia local del objeto con el que se la ha llamado y se guarda bajo el nombre *objetotemp*, realizándose sobre esta copia los cambios antes de pasarlos al objeto global. Esta técnica permite mayor flexibilidad, facilita que se cambie la clase del objeto dentro de la propia ventana más fácilmente y permite descartar los cambios si el usuario ha cometido un error, sin perder la información original. Es conveniente recalcar que la llamada a esta función se hace cuando el objeto ya está creado (tarea de la que se ocupan otras funciones, dependiendo de la clase que se desee) y que el objeto permanecerá en el escenario aún haciendo click en el botón cancelar. Cuando se hace click en aplicar o aceptar, la ventana reconoce el tipo de objeto que se está usando en ese momento y recoge y ordena la información en consecuencia (transformándola al tipo adecuado si es necesario) guardándola en el objeto temporal local. Si no detecta ningún fallo en este proceso, vuelca el objeto local sobre el global y permite continuar al programa, teniendo especial cuidado de modificar los objetos índice necesarios si ha habido algún cambio de nombre.

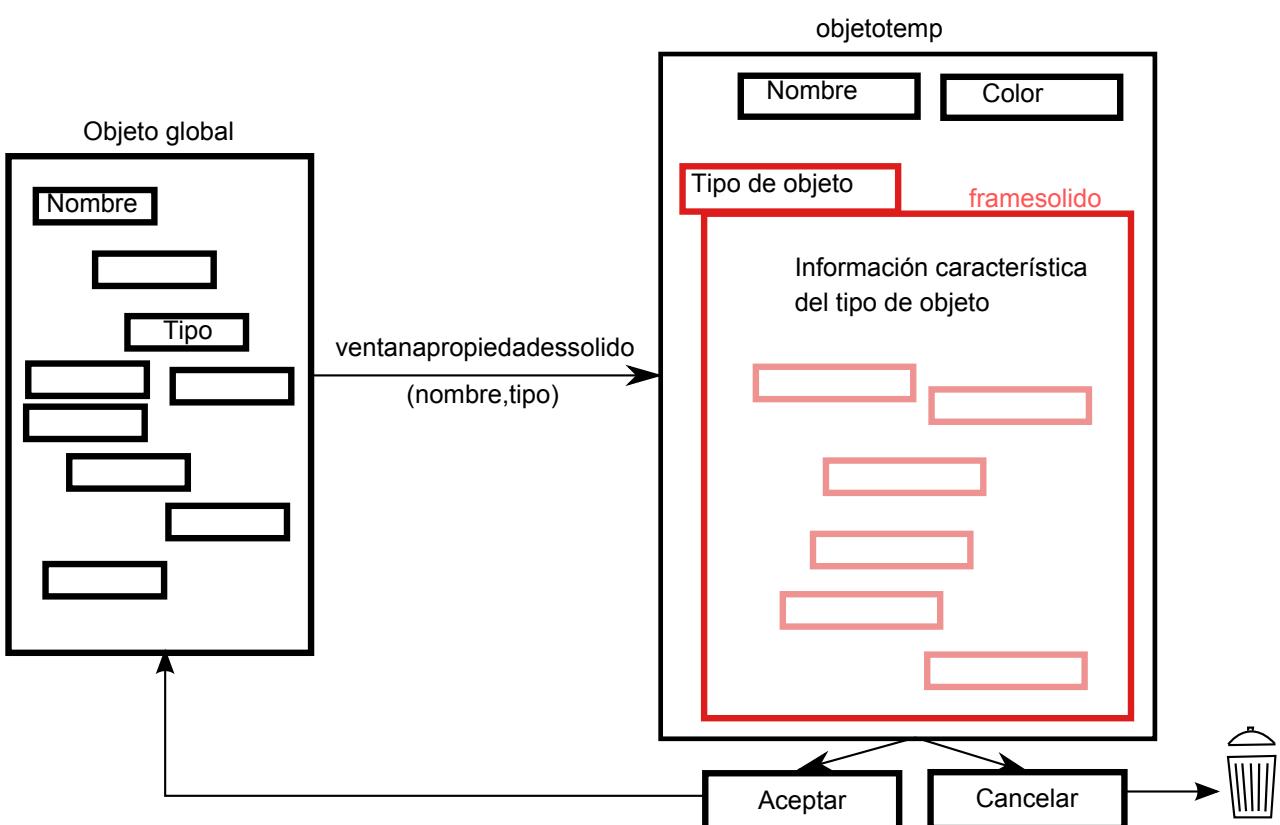


Figura 4.4: Ventana de propiedades del objeto

La ventana de opciones del escenario la maneja *ventanaopciones*, teniendo acceso a la variable global *escenario*. Dentro se define un notebook, para separar las opciones de la simulación de las puramente gráficas. Dentro del notebook, *frameescenario* y *framerep* contienen al resto de widgets necesarios para controlar cada uno de los parámetros por separado. Para cada parámetro se sigue una estructura similar, con una etiqueta especificando el parámetro y unos botones que controlan la variable del escenario, botones que sólo deben dejar elegir un caso a la vez.

Por la estructura que exige Tkinter para esta clase de construcción, se definen un tipo especial de variables que controlan cada valor específico y que permiten a los botones cambiar automáticamente su valor. Estas variables actúan entonces como controladores y los cambios que el usuario haga en la pantalla quedan reflejados en su valor. Para acceder a este valor se usa el método *[variable].get*

Al hacer click en *aceptar* o *aplicar* se acceden a todas estas variables locales, se lee su valor y se copia en el objeto global escenario. Además, cuando termina este proceso se guardan los valores del escenario en el archivo *config.cfg*, para que su valor se mantenga de una sesión a otra y sea más sencillo para el usuario usar el programa.

De la escritura de *config.cfg* se encarga el proceso *guardarconfig*, que toma los valores del escenario y los escribe en el archivo siguiendo un formato determinado. Se guarda un nombre descriptivo de la variable, por si es necesario modificarlo a mano, seguido del valor que toma el atributo del escenario, colocando la secuencia “=“ (*igual espacio*, sin las comillas) entre ambos campos.

Del proceso contrario, la lectura inicial de *config.cfg* se encarga *cargarconfig*. Lee el archivo por líneas, separa el atributo de su nombre y guarda su valor, convirtiéndolo al tipo que sea necesario (en el archivo se guarda texto plano, es necesario convertir, por ejemplo, los números, para que sean números y no una secuencia de texto). En la actualidad, es fundamental mantener el orden y separación entre los campos para que la carga funcione correctamente. En un futuro, se pretende dotar de más “inteligencia” al intérprete y flexibilizar un poco este aspecto, haciéndolo menos susceptible a errores. Por supuesto, al tratarse de un archivo de texto plano, el usuario puede acceder a este archivo con cualquier aplicación de edición de texto, pero se recomienda tener especial cuidado si se quiere modificar.

ventanaop

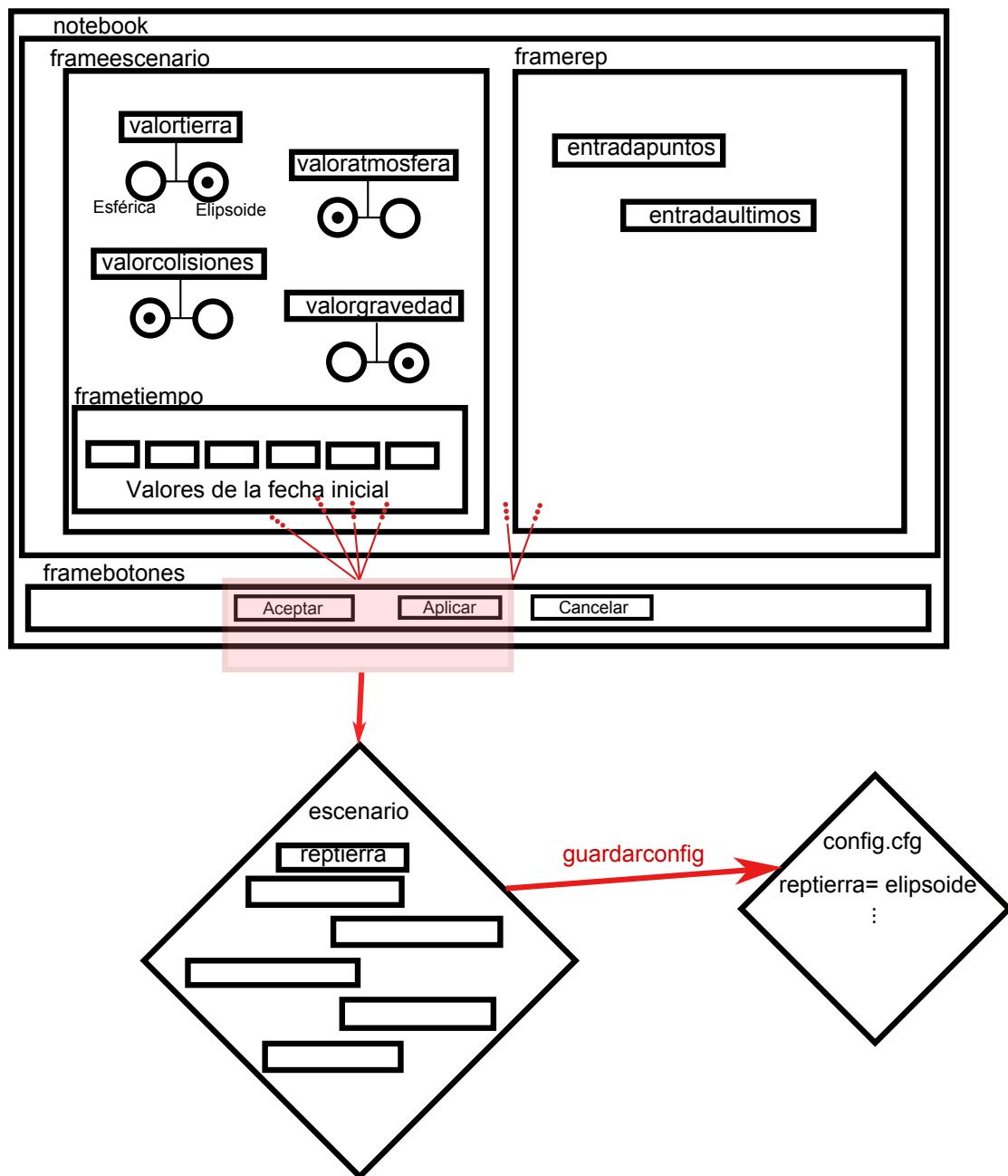
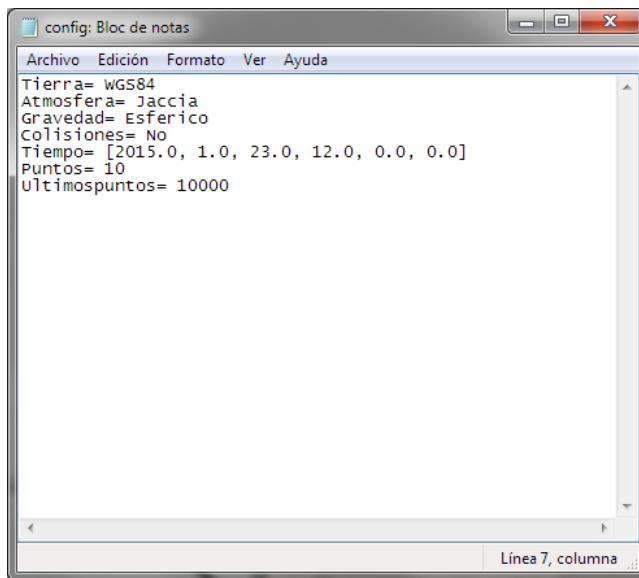


Figura 4.5: Estructura de la ventana de opciones

Figura 4.6: *config.cfg*

La ventana de visibilidad es una clase separada de tipo *ventanaasisvisibilidad*. Se deja la estructura de la ventana al método *framevis* y los cálculos posteriores a *visibilidad* y *visibilidadresumen*, dependiendo de cual este seleccionado en el valor de *resumen* (1 para la versión resumida y 0 para la completa).

Sea cual sea el método que se lanza, se leen los dos objetos que el usuario ha seleccionado en *obj1* y *obj2* y se crea un bucle de tiempo que se inicia en el instante inicial (en la fecha definida en el escenario) y va hasta el último instante que se tenga integrado con pasos de 1 segundo.. Si ambos objetos son de tipo sólido, el bucle avanza hasta el mínimo de los dos tiempos. Si sólo uno de ellos lo es, avanza hasta su último valor de tiempo, y si ninguno es de este tipo, avanza una cantidad fija de 86400 segundos, un día completo.

En cada paso del bucle se calculan las posiciones de los dos objetos accediendo, cuando corresponda a sus variables de tiempo y posición. Se busca entre la de tiempo el instante más adecuado y se extrae entonces de la posición la correspondiente a ese momento. Si el tiempo del bucle no coincide exactamente con los intervalos donde está definido el objeto, se hace una interpolación lineal entre los dos instantes más próximos, calculando una posición intermedia ponderada.

Una vez que se tienen las posiciones de ambos objetos, se lanza la función *visibilidad*, definida en *mision.py* y se guarda el resultado de la evaluación. Si se ha elegido la versión completa se guarda en el archivo elegido una linea con las posiciones de los objetos y el resultado de la evaluación de *visibilidad*, repitiendo el proceso para cada valor temporal.

Si se ha elegido la versión resumida, se guarda únicamente el valor del tiempo en que se detecta la entrada en visibilidad (entendida como un cambio de falso a verdadero en el valor de *visibilidad*) y la salida (un cambio de verdadero a falso). Si en el intervalo de tiempo procesado hay más de un ciclo de este tipo, se guarda cada uno en una linea diferente.

Cuando el bucle ha alcanzado el tiempo máximo, se cierra el archivo y el

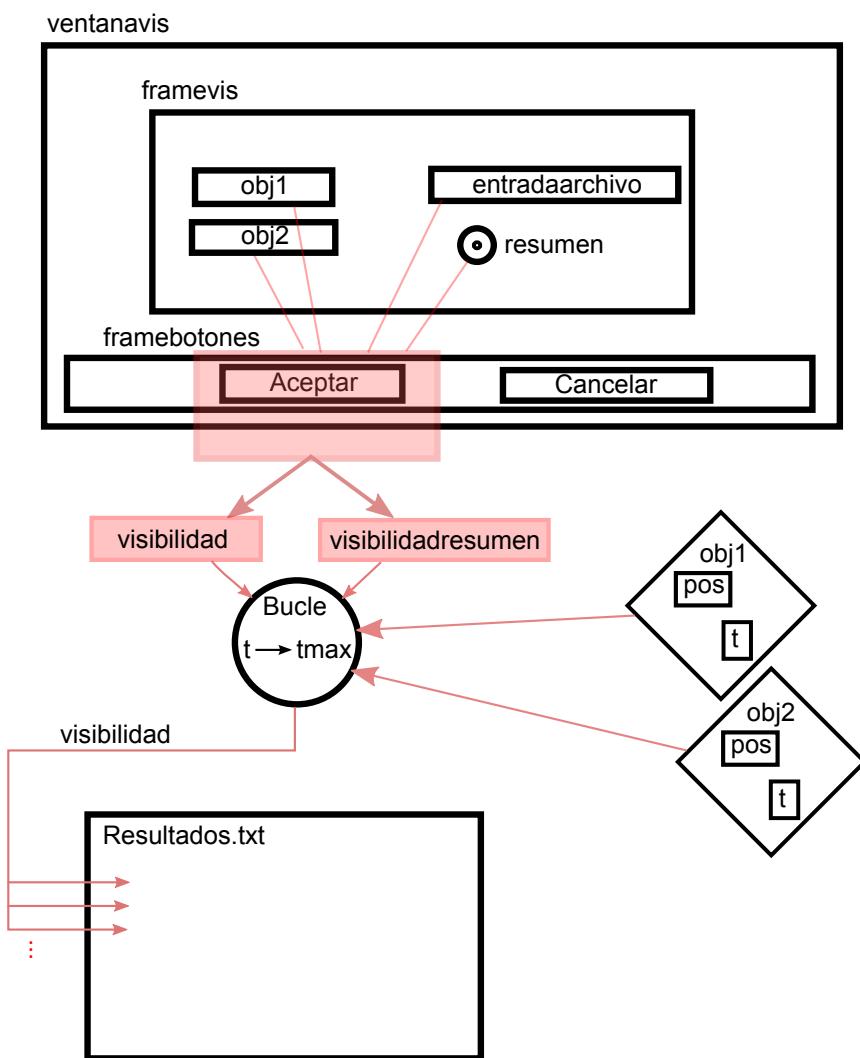


Figura 4.7: Estructura de la ventana de visibilidad

usuario puede acceder a los resultados en la forma en que haya elegido.

De manera parecida, se define la ventana del asistente de maniobras como un objeto de clase *ventanaasisismaniobras*. Dentro de ella, como se hacia con la ventana de definición de los objetos, un marco llamado *frametipo* se adapta a la clase de maniobra que se elige con *entradatipo* y *cambiotipo*. Los cálculos se hacen dentro del método *maniobra*, que comprueba el tipo de maniobra y dirige la función por el camino adecuado.

Una vez que el usuario ha elegido las dos órbitas involucradas, guardadas en *entradaorbita1* y *entradaorbita2*, al hacerse click en la selección del tipo de maniobra se lanza el método *cambiotipo*, que introduce en *frametipo* los campos necesarios que el usuario debe llenar antes de calcular la maniobra. Esta información dependerá del tipo que haya seleccionado.

Para una maniobra puntual entre dos órbitas, *manpunt* calcula los posibles puntos de corte entre ellas. Una llamada a *interseccionorbitas*, ya explicada, devuelve aquellos puntos donde las órbitas se acercan a menos de 1 Km en forma de la anomalía verdadera de la primera órbita. Esta lista de valores se muestra en *entradapunto*, una lista desplegable con los valores ya calculados para que el usuario pueda elegir el punto donde desea calcular la maniobra.

Para un impulso puntual, se despliega un marco, sobre el que se permite al usuario introducir valores para la anomalía verdadera inicial y las componentes del impulso propulsivo, en el sistema de referencia orbital. Los vectores del sistema de referencia, junto con el resto de la maniobra, se calculan al hacer click en el botón aceptar, teniendo en cuenta la anomalía verdadera introducida por el usuario.

Al hacer click en aceptar, se recoge la información de los campos elegidos por el usuario, se calculan las posiciones y velocidades de las órbitas implicadas y se crea un objeto de tipo *maniobra* para contener los resultados. Si además el usuario ha seleccionado conservar los objetos al final del cálculo, esta maniobra se guarda en la variable global *objetos* para permitir su acceso posterior y, si es necesario, se crean además los objetos tipo órbita implicados.

En el objeto *maniobra* se guarda toda la información que el usuario necesita, como las dos órbitas implicadas, el punto donde se realiza, descrito en ambas órbitas o el Δv en coordenadas cartesianas.

ventanaman

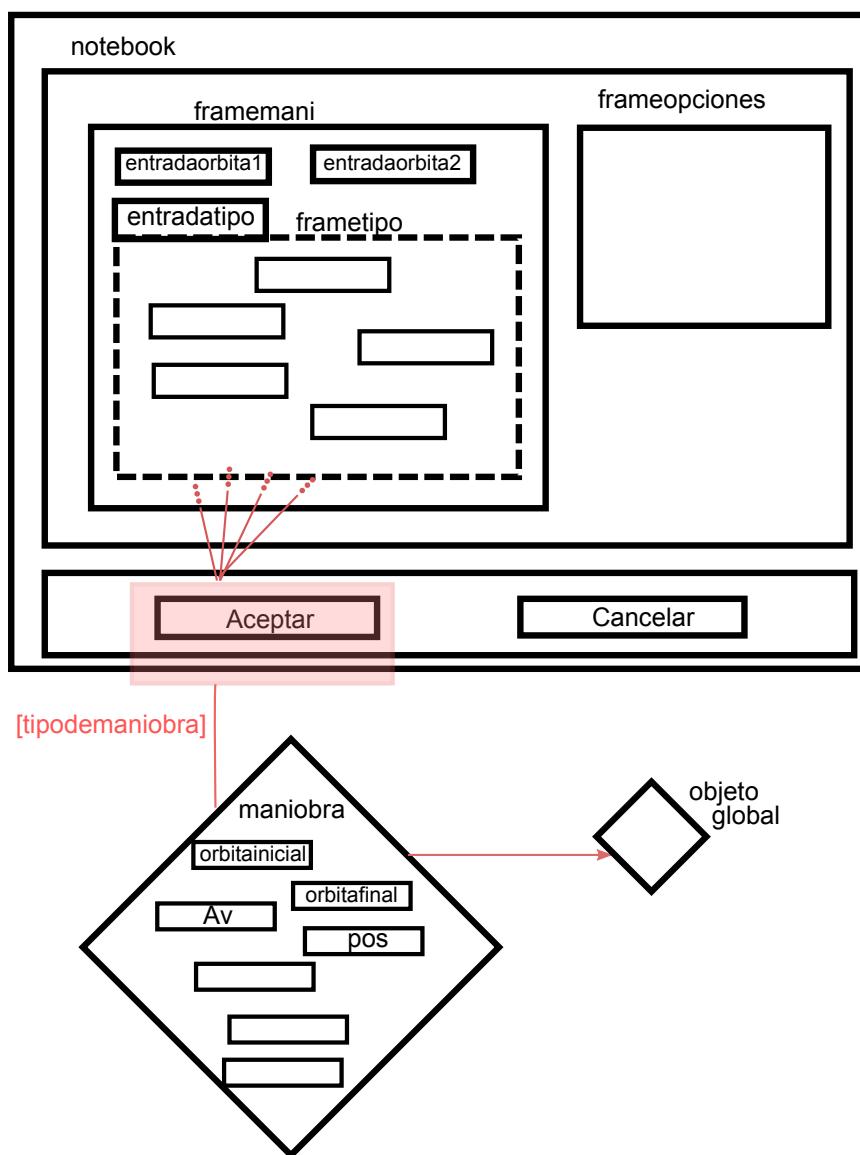


Figura 4.8: Estructura de la ventana de maniobras

La ventana de resultados requiere de elementos un poco más complejos. Dentro de uno de sus marcos, el llamado *frameobjetos*, se reparten los objetos en filas, cada una de una clase especial con los posibles parámetros del objeto. Estas filas, a la hora de escribir el archivo, devuelven los campos que el usuario desea escribir y modifican el archivo de salida en consecuencia. La función que se encarga de todos estos procesos se denomina *ventanaresultados*

Al crearse la ventana la función accede a los objetos globales y crea una serie de filas, cada una asociada a un objeto global. La información que está dentro de cada una de estas filas cambia según el tipo de objeto que sea, y todas contienen una serie de botones que permiten al usuario elegir que valores volcar al archivo de resultados. Cada botón tiene asociada una variable distinta, pero se reúnen y agrupan todas al final en una lista llamada *listaresultados*, a la que se accede a la hora de determinar los deseos del usuario.

El archivo se especifica mediante su ruta en *archivo*, y al hacer click en aceptar se creará, o sustituirá si ya hay otro con el mismo nombre.

Al hacer click en aceptar se crean dos bucles anidados. Por un lado se recorren todos los objetos globales. Por otro, se recorren cuando procede todos los instantes de tiempo en que se ha calculado algo. Para cada instante y objeto se comprueba la *listaresultados* correspondiente y se guardan en el archivo aquellos campos que el usuario ha seleccionado, realizando los cálculos intermedios que sean necesarios en cada paso.

Al terminar todos los bucles, se cierra el archivo y el usuario puede acceder al archivo plano de texto, que se ha formateado adecuadamente, y ver, analizar o exportar los datos como considere oportuno.

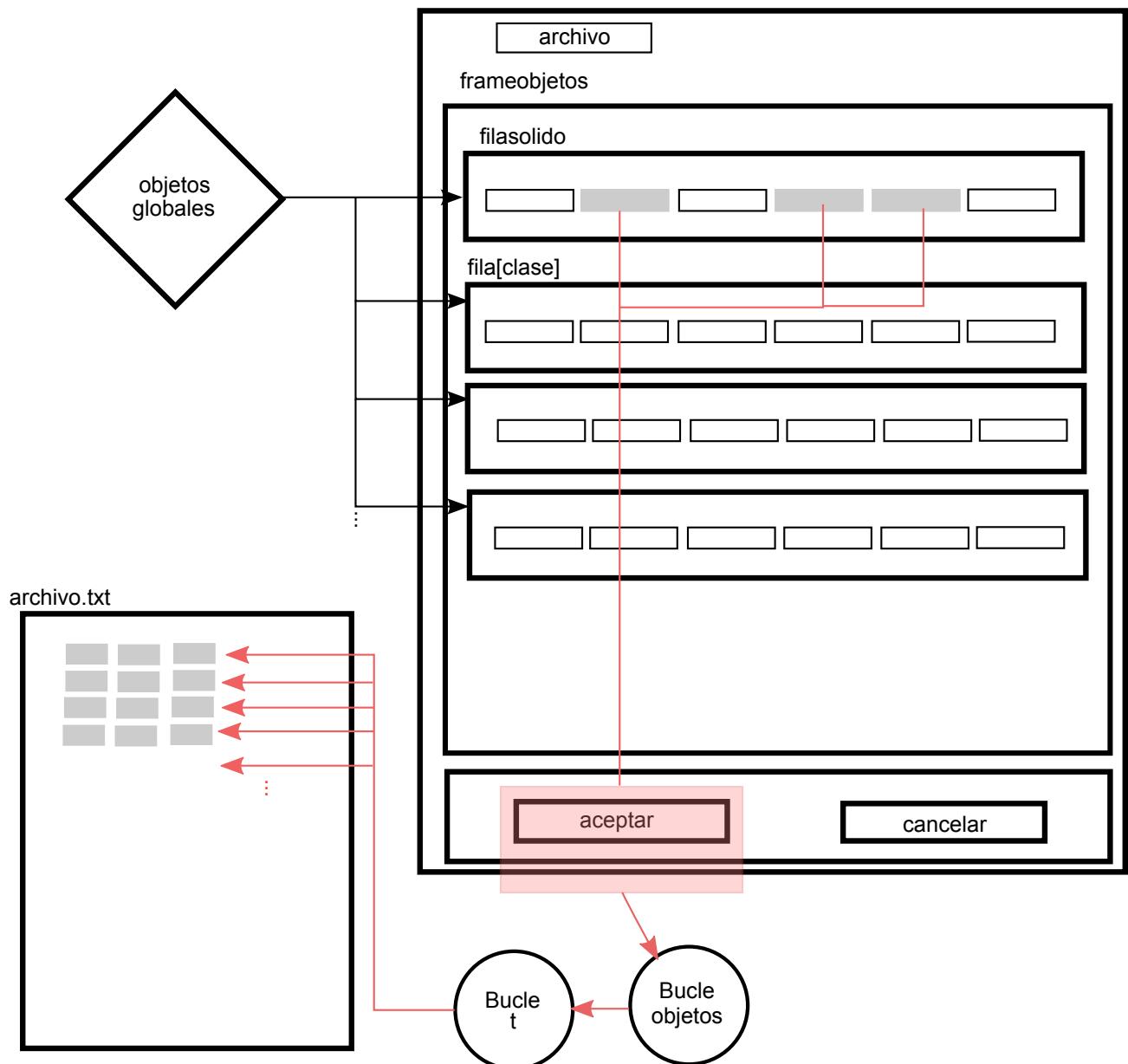


Figura 4.9: Estructura de la ventana de resultados

Hay por supuesto muchos aspectos de la interfaz que no se han descrito. Cada texto que aparece se ha tenido que definir y colocar mediante una etiqueta, y cada botón ha sido asociado a un *handler* distinto que interactúa de una manera única con el resto de elementos. Se han omitido de esta sección por considerarse poco interesantes, repetitivos y, en general, sencillos de rastrear. Se ha intentado seguir una nominación y estructura regulares a la hora de definirlos y ordenarlos, de manera que el usuario que lo desee pueda modificarlos o añadir los suyos propios. En las definiciones geométricas se han dejado espacios para poder introducir elementos auxiliares sin mucho compromiso, y se han intentado separar los elementos en grupos con objetivos parecidos.

En general, la interfaz gráfica es un mecanismo complejo, con muchas capas de objetos entrelazadas de distintas formas. Por ello se recomienda que el usuario que quiera modificar alguna parte de la estructura esté muy familiarizado con el módulo de Tkinter y sus aplicaciones antes de emprender el proyecto. Referencias útiles en este aspecto pueden encontrarse en la bibliografía del final.

Tema 5

Validación

En este capítulo va a tratarse la verificación y validación de todos los elementos implicados en la simulación y de su conjunto, abordando una de las partes más importantes de la construcción de todo simulador.

Objetivo

Una vez que se ha construido un simulador, se espera tener una serie de datos, normalmente numéricos, que sean la respuesta del sistema simulado a una serie de entradas. Sin embargo, estos datos son el resultado de un proceso puramente teórico y su validez está todavía en entredicho. No se pueden, o no se deben, usar estos datos de salida sin comprobar antes si se acercan o se alejan de la realidad, de manera consistente, para tener cierta seguridad de que podemos fiarnos de ellos a la hora de usarlos en un proyecto.

Como normalmente se hace una simulación para obtener la solución a un sistema para el que no se tiene respuesta analítica o experimental, resulta difícil comparar nuestra solución directamente con la realidad sin lentes y caros experimentos, y hay que buscar otra estrategia.

Una posible solución es usar varios simuladores independientes, introducir el mismo problema y comparar los resultados entre las soluciones que aporta cada uno de ellos, confiando en que en la multitud esté la respuesta correcta. Este proceso, aunque beneficioso, es largo y costoso, por lo que debe reducirse en la medida de lo posible.

La otra ruta más común consiste en elegir un problema para el que sí se tenga la solución. Se compara la simulación con ella y se extrae la validez del sistema hasta otros problemas. Esta es la vía más rápida, pero no siempre se tiene acceso a una solución de un problema lo suficientemente complejo como para resultar útil en este proceso.

El proceso de validación, se elija la ruta que se elija, consistirá en comparar la solución numérica con una solución patrón, viendo cuánto se desvía y comprobando que este error está dentro de las tolerancias del proyecto.

Como añadido a la validación de los modelos implementados se puede hablar también de un proceso de validación del software, comprobando que cumple los requisitos y objetivos que se le habían planteado al inicio del proyecto, y evaluando el programa desde el punto de vista del usuario final. Para obtener resultados

valiosos en este apartado, suele ser necesario un proceso largo y exhaustivo de análisis crítico y *feedback* de los clientes, normalmente mediante *beta-testers* o grupos de expertos.

Métodos de validación

Se plantean cuatro posibles métodos para realizar la validación de este programa.

1. Verificación del código. La manera más sencilla, rápida e intuitiva de comprobar que el simulador ha implementado correctamente los modelos definidos es, efectivamente, comprobar directamente que los modelos se han implementado bien. Este proceso se hace conociendo los modelos teóricos que se han implementado y navegando el código para inspeccionarlo y comprobar que la lógica y estructura del programa corresponde con la del modelo teórico. Este proceso no es ni mucho menos suficiente para asegurar la validez de la simulación, pero desde luego es necesario para abordar futuros pasos. El autor del programa ya ha realizado numerosas veces este método, pero se invita a que el lector haga una lectura crítica y compruebe por si mismo las funciones del simulador. Para ello, se ha intentado diseñar el código de manera clara y simple, con funciones modulares autodescriptivas y autocontenidoas, de manera que su inspección resulte fácil. Como el programa se distribuye con su código abierto, ayudado por las ecuaciones presentadas en el capítulo 2 cualquiera puede realizar esta tarea.
2. Validación del código frente a la solución analítica. Si el problema tiene una solución analítica sencilla y conocida, resulta muy fácil comparar los resultados numéricos obtenidos en el simulador con esta solución. Sin embargo, el rango de problemas que cumplen estos requerimientos es alarmantemente bajo, por lo que su aplicación a la validación de este programa no es posible más que en los procesos más sencillos.
3. Validación del código frente a la solución obtenida en otro simulador. Si otro programa resuelve la misma gama de problemas y ya tiene sus soluciones verificadas y validadas, resulta relativamente fácil comprobar la similitud entre las dos soluciones y extrapolarla a la similitud con la realidad. Elegir con criterio el programa que se va a usar como patrón resulta fundamental, y resulta conveniente elegir paquetes de software robustos, con muchos kilómetros encima y cuyo uso esté extendido en la industria. En este proyecto va a usarse como fuente de comparaciones el software STK en su versión 10, distribuido por Analytical Graphics, Inc.¹(AGI), un programa con mucho historial de servicio en la industria y prestaciones más que probadas.
4. Validación de la solución frente a datos reales. Qué mejor manera de estimar cuanto se acerca nuestra solución a la realidad que ver cuanto se acerca nuestra solución a la realidad. Aunque este método pueda resultar el más adecuado, no siempre es fácil o posible obtener datos de una misión

¹www.agi.com/products/stk

Validación del estado inicial

real y, cuando se tienen, no resulta nada fácil discriminar entre los efectos que se quieren verificar del resto de efectos y del ruido.

Todo programa que quiera competir en el mercado y contar con algo de rigor se someterá normalmente a los cuatro procesos, mediante análisis extensivos e intensivos de todos los aspectos y funciones del simulador y sus distintas interacciones. En este trabajo hemos intentado, dentro de la medida de los pocos recursos de que se dispone, de hacer esto mismo, usando las cuatro técnicas cuando sea posible y separando los procesos antes de analizar la función completa. Naturalmente, todas las pruebas que se van a realizar son reproducibles por el usuario y se invita al lector a que someta al programa a todos aquellos procesos que considere adecuados.

Validación del estado inicial

Antes de ver como se comporta la integración numérica del programa, vamos a comprobar que todos los elementos y sistemas de referencia están bien definidos, y que las conversiones entre ellos funcionan correctamente.

La NASA publica con regularidad los datos de la órbita que sigue la estación espacial internacional, además de su posición en distintos sistemas de referencia. Vamos a usarlos para comprobar que nuestro ECI está bien definido.

Órbita ISS J2000						NASA
23/1/2015 12:00:00						
Semieje mayor	Excentricidad	Inclinación	Longitud nodo asc	Argumento perigeo	Anomalía verdadera	
6789.96481 Km	0.0011196	51.746º	86.254º	37.759º	339.336º	

Introduciendo estos datos en el programa, se obtiene

Posición ISS J2000					
	Simulador	NASA		STK	
Posición Km	-808.29378	-808.30168		-808.300178	
	6549.97816383	6549.98438		6549.984541	
	1565.73046259	1565.70111		1565.700474	
Velocidad Km/s	-4.6762487	-4.67623009		-4.676235	
	-1.95617957	-1.956160859		-1.956159	
	5.75617466	5.756198415		5.756193	
Módulo error pos / % contra módulo		0.0310262	0.00045 %	0.0313196	0.00046 %
Módulo error vel / % contra módulo		3.55135·10 ⁻⁵	0.00046 %	3.07761·10 ⁻⁵	0.00040 %

La posición de este punto sobre la tierra en el instante señalado es

Traza ISS		
	Simulador	STK
Latitud	13.427183591	13.414
Longitud	154.674416216	154.741
Error latitud		0.013183591
Error longitud		0.066583784
Distancia error sobre la tierra		7.349 Km

Validación de la integración a corto plazo

Una vez que se ha comprobado que el sistema de cálculo de órbitas y los distintos sistemas de referencia funcionan correctamente, vamos a ver cómo se comporta la integración numérica del simulador. Para ello se sigue el mismo procedimiento, se elige una órbita (La de la ISS) y se integra en el tiempo la misma cantidad mediante el simulador y el STK para hallar el punto final y calcular sus diferencias. Además al hacerse estos primeros test sin perturbaciones, va a calcularse también la solución analítica que proporcionan las ecuaciones de mecánica orbital.

Posición ISS J2000			
Integrador: Adams-Bashforth 4	Paso de integración: 1. segundo		
Tiempo de integración: 90 minutos	Perturbaciones: Ninguna		
	Simulador	STK	Analítica
Posición Km	-11.872329	-12.082321	-11.89909952
	6759.06956	6759.032671	6759.049063
	575.10382	575.322834	575.1360026
Velocidad Km/s	-4.764617	-4.764604	-4.764616
	-0.522276	-0.522615	-0.5223207
	5.986831	5.986813	5.9868277
Módulo error pos		0.3056543	0.04661
Módulo error vel		0.000339	$4.48328 \cdot 10^{-5}$

Posición ISS J2000			
Integrador: Runge-Kutta 4	Paso de integración: 1. segundo		
Tiempo de integración: 90 minutos	Perturbaciones: Ninguna		
	Simulador	STK	Analítica
Posición Km	120.219219	-12.082321	-11.89909952
	6817.208853	6759.032671	6759.049063
	412.746093	575.322834	575.1360026
Velocidad Km/s	-4.745979	-4.764604	-4.764616
	-0.283243	-0.522615	-0.5223207
	5.983051	5.986813	5.9868277
Módulo error pos		217.5301	217.2747
Módulo error vel		0.2401249	0.239832

Posición ISS J2000			
Integrador: Predictor-Corrector	Paso de integración: 1. segundo		
Tiempo de integración: 90 minutos	Perturbaciones: Ninguna		
	Simulador	STK	Analítica
Posición Km	146.599039	-12.082321	-11.89909952
	6828.529375	6759.032671	6759.049063
	380.297794	575.322834	575.1360026
Velocidad Km/s	-4.741370	-4.764604	-4.764616
	-0.236034	-0.522615	-0.5223207
	5.981129	5.986813	5.9868277
Módulo error pos		260.853	260.597584
Módulo error vel		0.287577	0.287285

Puede comprobarse que el integrador que mejor se comporta es el Adams-Bashforth de cuarto orden, alcanzando con él una gran precisión.

Puede surgir la duda de cómo cambia el comportamiento frente a distintos pasos de integración. Se repiten las pruebas, usando el integrador Adams-Bashforth y variando el Δt

Δt	Posición Km	Velocidad Km/s	Error pos con STK	Error vel con STK
0.1	1044.1713	-4.5612	1700.18	1.872172
	6853.5761	1.3356		
	-753.5961	5.8834		
0.5	-12.023336	-4.764616	0.06672655	$6.9231 \cdot 10^{-5}$
	6759.039837	-0.522547		
	575.292472	5.986808		
1.0	-11.872329	-4.764617	0.3056543	$3.3972 \cdot 10^{-4}$
	6759.06956	-0.522276		
	575.10382	5.986831		
1.5	-11.620536	-4.764617	0.710934	$7.93 \cdot 10^{-4}$
	6759.119175	-0.521824		
	574.789261	5.986869		
2.0	-11.268124	-4.764618	1.27889	$1.42 \cdot 10^{-3}$
	6759.188559	-0.521193		
	574.348998	5.986923		
5.0	-7.0404	-4.7646	8.0948	$9.02 \cdot 10^{-3}$
	6760.0180	-0.513617		
	569.0671	5.9875		
10.0	8.0503	-4.7645	32.422	0.0361
	6762.9297	-0.4866		
	550.2094	5.9897		

Puede comprobarse que el mejor resultado se consigue usando pasos de tiempo menores que 1. Sin embargo, esto aumenta significativamente el tiempo requerido de computación y alarga el proceso. Usar pasos muy pequeños, además de disparar el tiempo requerido para hacer la integración, introducen mucho error de redondeo y dan lugar a resultados malos. Usando pasos mayores que 1 se consigue reducir el tiempo de integración manteniendo todavía una buena precisión en los cálculos, por lo que para grandes integraciones puede resultar una estrategia rentable. Eligiendo usar un paso de tiempo de 1, se alcanza un compromiso adecuado entre precisión y tiempo de cálculo.

Validación de la integración a largo plazo

Que el proceso de integración sea preciso a corto plazo no quiere decir que lo sea a largo. El problema de mecánica orbital es inestable en el sentido de *Lyapunov*: Perturbaciones en el estado inicial, por pequeñas que sean, se propagan irremediablemente en el tiempo hasta hacerse tan grandes como queramos. Esto hace que la integración a muy largo plazo sea inviable sin implementar ciertas técnicas y trucos que regularicen los resultados en cada paso.

Vamos a integrar el mismo problema, pero esta vez durante 7 días, y comprobar como ha evolucionado la precisión.

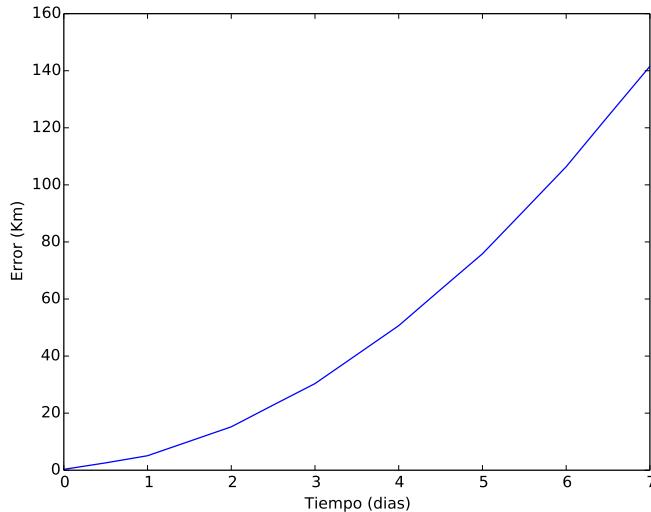


Figura 5.1: Evolución del error en la integración

Posición ISS J2000			
Integrador: Adams-Bashforth 4		Paso de integración: 1. segundo	
Tiempo de integración: 7 días		Perturbaciones: Ninguna	
	Simulador	STK	Analítica
Posición Km	3333.9964 -3810.4834 -4535.2616	3387.0907 -3692.4967 -4592.6811	3385.2832 -3696.5798 -4590.7318
Velocidad Km/s	2.9170 6.3275 -3.1674	2.8381 6.4156 -3.0603	2.8409 6.4127 -3.06408
Módulo error pos		141.5516	136.6795
Módulo error vel		0.15955	0.15403

Puede verse que el error ha aumentado significativamente. Representando la evolución del módulo de la diferencia de la solución con la que proporciona el STK, con datos tomados cada 12 horas de integración obtenemos la figura 5.1

Esta variación tan fuerte es debida al carácter inestable ya explicado con anterioridad: Pequeñas variaciones en la posición inicial llevan a órbitas ligeramente diferentes y con movimientos medios distintos, de manera que, con el tiempo, el error no estará acotado, por pequeño que sea el error inicial.

A primera vista puede parecer éste un resultado malo, pero no hay que olvidar el objetivo del programa: Proporcionar una ayuda de cálculo para las primeras fases del análisis de misión de un proyecto de mecánica orbital, para generar datos suficientes con los que avanzar. Teniendo esto en cuenta, esta diferencia no resulta tan escandalosa, ya que el refinamiento posterior que se hará de la misión y sus parámetros, de los modelos y del software introducirá variaciones que harán que se superen con creces estos 150Km.

Si además cambiamos estas posiciones a los elementos clásicos de la órbita

	a	e	i	Ω	ω	ϑ
Simulador	6790.1333	0.001129	51.7458	86.2537	37.6941	200.48
STK	6789.7206	0.001154	51.746	86.2540	38.2167	201.15

Se ve con mucha más claridad que la diferencia entre los parámetros es mucho menor que el orden de magnitud típico del cambio que van a introducir las distintas perturbaciones.

Así pues, se puede considerar que el programa cumple los requerimientos de precisión necesarios para cumplir su objetivo con fiabilidad.

Validación de las perturbaciones

Siguiendo el mismo principio, una vez que se ha verificado que el código se ajusta a los modelos planteados, es hora de comparar resultados numéricos y ver cuanto se acercan al patrón. Con la integración numérica ya validada, vamos a poner el foco en las distintas perturbaciones que se pueden añadir a la simulación.

En primer lugar vamos a comprobar la atracción gravitatoria debida a J2. Esta perturbación, al derivar de un potencial, admite una solución semianalítica añadida al problema esférico simplificado. Sin embargo, en la simulación no se hace uso de esta solución semianalítica, sino que se incorpora la fuerza de perturbación, derivada del potencial, directamente a la suma de fuerzas. No obstante, podemos usar la solución semianalítica para saber *qué esperar* del proceso de integración de la perturbación. Los efectos más importantes son los siguientes:

- Precesión de la linea de nodos: La conservación del momento cinético hace que el plano de la órbita cambie lentamente, girando alrededor del eje de polos en dirección oeste para órbitas posigradas y este en órbitas retrógradas, efectivamente cambiando la longitud en la que se encuentra el nodo ascendente.
- Rotación del perigeo: De igual manera, la posición del perigeo dentro del plano de la órbita cambia, variando de nuevo la posición de la órbita pero no su forma.

Los parámetros que más influyen en estos dos movimientos son la excentricidad y la inclinación de la órbita. Eligiendo correctamente estos dos valores, puede controlarse el movimiento que produce la perturbación e incluso aprovecharse. Este es el principio fundamental en que se basan las órbitas heliosíncronas, las *Molniya* o las *Frozen*, cuyas propiedades hacen que su explotación resulta muy interesante para determinadas aplicaciones.

Volviendo al tema de la validación, va a elegirse una órbita que no cumpla ninguna de estas características mencionadas anteriormente, para poder ver la magnitud del cambio que provoca la perturbación. Además, errores de integración hacen que esta perturbación se comporte mal en órbitas de baja excentricidad (no nula), por lo que se va a usar una órbita con una excentricidad algo mayor que la de la ISS.

Se eligen los siguientes parámetros para realizar la prueba

Elementos iniciales

a	e	i	Ω	ω	ϑ
7370	0.05	47	86	37	156

Y se integra el problema durante 1 día obteniendo

Elementos clásicos	Integrador: Adams-Bashforth 4	Paso de integración: 1. segundo
J2000	Tiempo de integración: 1 día	Perturbaciones: J2
		a e i Ω ω ϑ
Simulador	7361.96	0.0506
STK	7370	0.05
		46.9690 81.8450 40.4915 63.771
		47 81.881 41.003 59.354

Se ve que la evolución de los distintos parámetros en el simulador y en el STK son similares, dando una órbita resultado muy parecida. Destaca mucho el error en el semieje mayor, debido posiblemente a errores en la alineación de las fuerzas, pero el usuario versado en mecánica orbital debería ser capaz de descartar este comportamiento en su análisis.

La validación de la fuerza aerodinámica es mucho más complicada. La cantidad de parámetros que influyen en el resultado final y su alta variabilidad hace que el parecido de una misión real con los datos obtenidos de un modelo simple como el que incorpora el programa sea cuanto menos cuestionable. Por tanto, no va a ser posible hacer una validación exhaustiva de este punto. Sin embargo, vamos a comprobar que los resultados obtenidos son de la forma que cabría esperar y tienen los efectos que les corresponden, dentro de un orden de magnitud parecido a la realidad. Los efectos que se pueden esperar de la resistencia aerodinámica son

- En órbitas con cierta excentricidad, la fuerza aerodinámica actúa mucho más duramente en el perigeo que en el apogeo de la órbita. Esto provoca que la posición del primero no se modifique mientras la del último cae. La órbita pierde energía pero la posición de su pericentro no cambia, por lo que la órbita también pierde excentricidad: Se circulariza. Este sistema se usa en misiones interplanetarias cuando el objetivo tiene atmósfera, para pasar de la órbita inicial hiperbólica con que se llega a una órbita elíptica adecuada para realizar la misión sin consumir grandes cantidades de combustible.
- En órbitas con baja excentricidad, u órbitas ya circularizadas, la resistencia tiene la misma fuerza tanto en el apogeo como el perigeo, por lo que su efecto es continuo. Se produce entonces una pérdida gradual de altitud hasta que se alcanza un punto crítico y el objeto se precipita contra la superficie. Es el conocido fenómeno de la reentrada. Durante la vida útil del satélite este efecto suele ser negativo, pues pone en peligro la continuidad de la misión (La ISS tiene que realizar de vez en cuando maniobras para recuperar la altitud perdida), pero, una vez que el satélite ha dejado de funcionar, puede jugar a favor, pues permite la retirada del satélite

de la órbita sin gastar combustible, permitiendo a otro ocupar su lugar y reduciendo la basura espacial.

La NASA, además de los datos orbitales de la ISS, publica algunos datos de las propiedades físicas de la estación, que van a resultar muy útiles para este punto. Vamos a usar los datos correspondientes al 23 de Enero de 2015, los que hemos estado usando hasta ahora, con los añadidos

$$\text{Área} = 17222,26 \text{ ft}^2 \simeq 1600 \text{ m}^2$$

$$C_D = 2,0$$

$$\text{Masa} = 950073,3 \text{ lb} \simeq 430946 \text{ Kg}$$

$$\text{Flujo solar mensual } F_{10,7} = 132,9$$

Y mide el *decay* de la órbita en el cambio de revoluciones que sufre la ISS en un día, revoluciones/día·día.

Con estos parámetros obtenemos un coeficiente balístico

$$CB = \frac{m}{C_D A_f} \simeq 135 \frac{\text{Kg}}{\text{m}^2}$$

Hay que mencionar que el flujo solar mensual está un poco por encima del medio que se usa en el programa, lo que llevará a una temperatura exoesférica mayor y a un ligero sobredimensionamiento de la resistencia aerodinámica en el simulador.

Definimos un objeto con la órbita de la ISS y un coeficiente balístico de 135, e integraremos el problema durante un día teniendo en cuenta la atmósfera. Si nos fijamos en los resultados del semieje mayor

Posición ISS J2000		
Integrador: Adams-Bashforth 4		Paso de integración: 1. segundo
Tiempo de integración: 1 día		Perturbaciones: Atmósfera
	Simulador	NASA
Semieje mayor inicial Km	6789.999534	-
Movimiento medio inicial rev/dia	15.516656472	-
Semieje mayor final	6789.895692	-
Movimiento medio final	15.517012432	-
Ratio de declive orbital rev/dia^2	$3,5595 \cdot 10^{-4}$	$1,6717 \cdot 10^{-4}$

Puede verse que, aunque ambos valores no coincidan, si son del mismo orden de magnitud. Analizando el resto de elementos orbitales de la simulación, se ve también que la órbita evoluciona como se espera de esta perturbación, manteniendo su forma y posición y perdiendo altitud lentamente. Se concluye por tanto, que, para un primer análisis con el grado de precisión que se busca, la implementación de la perturbación atmosférica resulta adecuada.

Tema 6

Conclusiones

Objetivos

Hay muchas maneras de valorar el éxito de un proyecto. Se puede sopesar el rendimiento económico que genera frente a los gastos incurridos. Se pueden analizar los cambios que introduce y los avances que produce. Incluso un proyecto en principio fallido puede ser muy beneficiosos si genera conocimiento y experiencia para el futuro. Pero sin duda una de las maneras más objetivas de analizar el buen término de un proyecto es comprobar que cumple los objetivos que se habían planteado al comenzar.

Al principio del proyecto se enfocó un objetivo más o menos claro: Crear un simulador que permitiera al ingeniero o equipo de ingenieros afrontar los primeros pasos de un proyecto de análisis de misión de órbita baja terrestre, generando datos suficientes para poder empezar a caracterizar los subsistemas del satélite.

Se ha creado y desarrollado un conjunto de funciones que cumplen esta tarea, integrando los aspectos necesarios de la simulación dentro de un mismo proceso y proporcionando herramientas de cálculo adicionales que permitan la explotación y análisis de los datos obtenidos mediante la integración numérica. Se han dado métodos para diseñar la órbita que ocupará la misión, teniendo en cuenta varios factores. Se han integrado herramientas que permiten generar datos preliminares para distintos subsistemas, como los tiempos de eclipse para el sistema de potencia o los tiempos de visibilidad para el sistema de comunicaciones, y que permiten extraer estos datos para su tratamiento externo posterior.

Se han creado las distintas funciones de manera que resulte fácil acceder a ellas, comprenderlas y modificarlas en caso de que se requiera ampliar el rango del simulador. Se ha diseñado modularmente para que cada fragmento sea lo más autónomo posible y tenga sus competencias perfectamente delimitadas. Se ha creado el código también de manera que cada función pueda ser extraída de su conjunto y usada aparte o integrada en otros programas con los mínimos cambios posibles. Se ha distribuido el código de manera abierta para que cualquiera pueda usarlo, verificarlo y modificarlo según sus necesidades.

Se ha creado una interfaz gráfica sencilla y amigable, con secuencias de pulsado cortas y intuitivas, que proporciona un punto de encuentro entre las distintas funciones del programa y una manera rápida de visualizar preliminarmente los

datos obtenidos de la simulación.

Se ha alcanzado una gran precisión en la integración a corto plazo y una aceptable en la integración a largo plazo. El parecido de los resultados con la realidad o con los obtenidos con otro software líder de la industria hacen que los datos obtenidos del simulador sean perfectamente adecuados y fiables para afrontar la etapa de diseño objetivo y proporcionar unas primeras cifras sobre las que desarrollar el resto del proyecto.

En resumen, se han cumplido los objetivos que se habían planteado en un principio con soltura, por lo que el simulador resulta adecuado para la función que se había diseñado.

El camino

Se hace camino al andar. Y sin duda he andado mucho a lo largo de este proyecto. Un comienzo lento, poco familiarizado con el lenguaje, pasando por un replanteamiento del objetivo del programa hasta llegar a lo que es ahora han caracterizado el desarrollo de este proyecto. He pasado muchas horas tratando de hacer un código simple y depurado, a la vez que lo suficientemente complejo y flexible como era necesario. He pasado aún más horas arreglando y rediseñando ese código, cuando se llegaba a un callejón sin salida, o cuando no cumplía los requisitos que le exigía. Y, tras mucho pelear, creo que he conseguido un programa integral y completo, que cumple el objetivo propuesto. He tenido que revisar y refrescar muchos conocimientos anteriores sobre ecuaciones y modelos físicos antes siquiera de plantear su implementación en el programa. He tenido que lidiar con escasa información de algunos apartados y desarrollar ecuaciones a partir de ella. Y he tenido que trabajar en la integración de distintos modelos, métodos y ecuaciones para que se comunicasen entre ellas y se adaptasen a las necesidades del usuario.

He tenido que enfrentarme con el aprendizaje del lenguaje de programación según lo iba necesitando, y revisar lo ya escrito para adaptarlo a este nuevo conocimiento adquirido. En el transcurso del proyecto me he dado cuenta, comprendido y afrontado los problemas especiales de emprender un proyecto de este calibre, de las necesidades de la estructura y planificación del código y de las ventajas de una estructura modular y definida. He adquirido conocimiento de la filosofía de programación orientada a objetos y de sus métodos y virtudes, así como de técnicas de programación generalistas y del mucho espacio de mejora que tengo en este aspecto. He comprendido lo que significa el proceso de desarrollo de un software determinado, y llegado a imaginar la magnitud de un programa más especializado y completo.

De este modo, no sólo he aprendido mucho sobre aspectos técnicos y específicos del programa, sino que también he aprendido mucho sobre mí mismo, sobre mis límites y sobre la motivación a la hora de trabajar en un proyecto tan largo y exigente. He tenido que lidiar con problemas, con inspiración y rachas de bloqueo, y con motivación y desmoralización a lo largo de todo este tiempo. He aprendido a pelear con fuentes poco comunes y poco explicativas a la hora de adquirir información, y con problemas poco documentados. He aprendido sobre la necesidad de ampliar constantemente horizontes y conocimientos, con un aprendizaje continuo y abierto y con no descartar fuentes por no cumplir con todas las expectativas.

He aprendido mucho como persona y futuro ingeniero a lo largo de este proyecto, y creo por tanto, a pesar de las dificultades y gracias a ellas, que su ejecución sin duda resultará muy valiosa a la hora de afrontar mi futuro en la profesión y la vida.

Ad astra per aspera.

Aspectos a mejorar

Resulta difícil dar por terminado un proyecto y darse por satisfecho con los resultados obtenidos. Siempre hay algo que añadir, algún punto que mejorar, algún proceso que pulir. Sin embargo, es necesario ser realista y conformarse con un punto adecuado que busque el equilibrio entre prestaciones y coste, entre *input* y *output*. Y aunque este punto se alcance, y considero que en este proyecto se alcanzado un punto adecuado, es muy conveniente ser consciente de las limitaciones que implica parar en el camino y tener diseñada la hoja de ruta que permitiría, en un futuro, avanzar aún más. Por ello, voy a señalar aquellos puntos del programa cuya implementación o mejora resultaría prioritaria antes de emprender otras vías.

- Mejorar la velocidad de las distintas funciones. El proceso de integración en la actualidad resulta lento y pesado, y puede dar lugar a un análisis demasiado encorsetado, poco interactivo. Resultaría de máxima prioridad reducir este tiempo al mínimo posible, haciendo que la integración fluya mucho mejor y facilitando el análisis. Las causas de este retardo pueden ser muchas, pero principalmente se barajan dos: La manera en que Python realiza operaciones numéricas y la manera en que se almacenan y actualizan los resultados de esas operaciones. Python es un lenguaje generalista, que resulta muy adecuado para grandes aplicaciones que tocan muchos puntos, pero no es el más efectivo cuando se trata de realizar grandes cantidades de operaciones de cálculo. Lenguajes como Fortran resultan mucho más eficientes en este ámbito, aunque carezcan de potencia en otros. Este problema resulta difícil de abordar. Afortunadamente, otro de los puntos en que Python destaca es en su integrabilidad: Es posible, no tan fácilmente, incorporar al programa código escrito en otro lenguaje con un proceso especial que permite al intérprete de Python comprenderlo y usarlo en sus procesos. Sería incluso posible, en teoría, escribir las funciones más demandantes en código de bajo nivel (más cercano al lenguaje máquina y con menos intermediarios) y reducir el número de procesos que se llevan a cabo para cada sentencia. Cualquiera de estas soluciones, sin duda, reduciría el tiempo de integración y haría la experiencia más fluida. Otra posible implementación tiene que ver con como maneja el lenguaje los objetos. A medida que un objeto se hace más y más grande, las distintas funciones que se aplican sobre él se hacen más y más lentas y pesadas, retrasando notablemente los procesos. Implementar alguna manera en que la información ya integrada se guardase en otro punto, descargando de la memoria los grandes arrays de datos sin duda ayudaría a, si no mejorar, si que evitar retrasar el proceso de integración como sucede en la actualidad.

- Mejorar la precisión. No como fin en si mismo, aunque obtener cálculos más precisos nunca resulte molesto. La precisión que se ha obtenido es adecuada para la función del programa. Sino como medio, mejorar la precisión de los cálculos sin duda permitirá incrementar el paso de tiempo que se puede dar sin comprometer la validez de los resultados, efectivamente reduciendo el tiempo final requerido para la integración. Este punto es más difícil de tratar. Existen métodos numéricos que reducen los errores en la propagación del problema de mecánica orbital, como el método de *Störmer-Cowell* o la regularización de *Sterling* que sin duda el programa aprovecharía. Alguna manera de implementar una solución semianalítica también resultaría adecuada. Sin embargo, habría que estudiar la implementabilidad de estos métodos, y si su añadido mejoraría este punto lo suficiente como para justificar el tiempo y coste requeridos.
- Con la mejora del tiempo de integración viene un proceso de diseño más fluido, flexible y con más posibilidades. La manera prioritaria de implementar esto sería incluir en el diseño de órbitas algunos tipos especiales, como heliosíncronas o *Frozen*, que facilitasen el diseño rápido de algunos tipos de misión. Tener la posibilidad de diseñar y caracterizar maniobras más complejas, como un *rendezvous* o una transferencia a una órbita interplanetaria sin duda ampliará las aplicaciones para las que el programa puede ser útil.
- Introducir más herramientas de análisis. Obtener rápidamente, por ejemplo, los ángulos que forma el sol con una cara del satélite o los ángulos de un paso del objeto vistos desde la tierra permitiría profundizar en los datos con facilidad y ampliar las posibilidades al alcance del usuario del programa sin interferir en el resto de procesos.
- Mejorar las salidas y entradas del programa. Permitir al usuario elegir, por ejemplo, con qué formato quiere obtener las salidas numéricas o qué precisión requiere. Mejorar el sistema de lectura de archivos, para hacerlo más “inteligente” y flexible y mejorar su respuesta a errores en los archivos. Introducir un sistema de importación de datos desde archivos externos, para facilitar la portabilidad con otros programas. En definitiva, mejorar las vías con las que el programa se comunica con el exterior, para aumentar su versatilidad.
- Migrar los cálculos gráficos. Cambiar el sistema de representación gráfica con un módulo más adecuado para la representación en tiempo real. Un módulo propio permitiría, además de más independencia, mayor control sobre lo que se muestra en pantalla respecto a las librerías de matplotlib, mejorando en muchos aspectos de rapidez y fluidez y permitiendo hacer representaciones más ambiciosas y completas y mucho más dinámicas. Poder crear y destruir elementos al vuelo sin tener que tocar el resto o poder implementar texturas complejas o cambios de cámara sin duda mejoraría la experiencia del usuario. Esta tarea requiere sin embargo de muchos más conocimientos informáticos y experiencia de los que dispongo en la actualidad, pero sin duda sería un punto importante y necesario del desarrollo futuro.

Y muchos más aspectos se quedan en el tintero, en el saco de ideas, a la espera de tener los recursos y conocimientos necesarios para llevarlos a cabo. No por pocas razones el desarrollo de software resulta en tiempos de desarrollo tan largos, con equipos tan grandes y numerosos retrasos, y sin duda a lo largo del proyecto he desarrollado un renovado respeto por el campo. Resulta difícil dar por terminado un proyecto en el que se ha trabajado tanto tiempo, cuando quedan *tantas* cosas por hacer. Pero hay que ser realistas y considerar los recursos disponibles. Estoy muy contento con los resultados obtenidos hasta la fecha, cuyos avances son suficientes como para justificar el objetivo que se planteó al comienzo y la independencia y capacidades del programa en este punto. Sin embargo, la presentación de este proyecto no tiene por qué, y no va a, significar un punto y final y carpetazo al desarrollo del simulador. He sufrido y aprendido mucho a lo largo de este tiempo, pero sin duda en el futuro seguiré trabajando para mejorar y ampliar el software, explorando todas las posibilidades que se me ocurran.

Índice de figuras

2.1.	Días solar y sidéreo	15
2.2.	Coordenadas cartesianas frente a cilíndricas	21
2.3.	Coordenadas geográficas	22
2.4.	Coordenadas geográficas sobre el mapa	23
2.5.	Distorsión de la proyección cilíndrica	23
2.6.	Coordenadas celestiales	24
2.7.	Coordenadas eclípticas	25
2.8.	Problema de los dos cuerpos	27
2.9.	Elementos de la órbita elíptica	30
2.10.	Elementos clásicos de la órbita	33
2.11.	Triedro orbital	34
2.12.	Triángulo de velocidades	35
2.13.	Transferencia de Hohmann	35
2.14.	Tierra esférica	40
2.15.	Sección del elipsoide de referencia	42
2.16.	Geoide	43
2.17.	Potencial gravitatorio	44
2.18.	Geometría del potencial de un cuerpo general	45
2.19.	Armónicos esféricos	48
2.20.	Error de redondeo, truncación y global	58
3.1.	Ventana principal	62
3.2.	Ventana de parámetros de escenario	63
3.3.	Añadir sólido	66
3.4.	Añadir órbitas y bases	67
3.5.	Añadir sensor	68
3.6.	Apuntamientos posibles	69
3.7.	Control de la simulación	70
3.8.	Árbol y objeto seleccionado	71
3.9.	Representaciones bidimensional y tridimensional	72
3.10.	Representación del sensor	73
3.11.	Ventana de información	74
3.12.	Ventana de resultados	74
3.13.	Archivo de resultados	76
3.14.	Ventana de visibilidad	77
3.15.	Ventana de maniobras	78
3.16.	Resultados de la maniobra	79

ÍNDICE DE FIGURAS

4.1.	Flujo del bucle de integración	85
4.2.	Estructura del sensor	89
4.3.	Estructura de root	97
4.4.	Ventana de propiedades del objeto	99
4.5.	Estructura de la ventana de opciones	101
4.6.	<i>config.cfg</i>	102
4.7.	Estructura de la ventana de visibilidad	103
4.8.	Estructura de la ventana de maniobras	105
4.9.	Estructura de la ventana de resultados	107
5.1.	Evolución del error en la integración	114

Bibliografía

- [Tkinter] «Tkinter Documentation and References.» URL <https://wiki.python.org/moin/TkInter>.
- [Jacchia77] HUESTIS, David L. «Jacchia 1977 Atmospheric Model.» National Aeronautics and Space Administration. URL <http://nssdcftp.gsfc.nasa.gov/models/atmospheric/jacchia/jacchia-77/>.
- [matplotlib] HUNTER, J. D. «Matplotlib: A 2D graphics environment.» URL <http://matplotlib.org/>.
- [Dinámica] GARCÍA DE JALÓN, Javier, y Eduardo BAYO. *Kinematic and Dynamic Simulation of Multibody Systems*. Springer-Verlag, 1994.
- [NumPy] JONES, Eric, Travis OLIPHANT, Pearu PETERSON, *et al.* «SciPy: Open source scientific tools for Python.», 2001-. URL <http://www.scipy.org/>.
- [Gravedad] MASSACHUSETTS INSTITUTE OF TECHNOLOGY. «The Earth's Gravitational Field.» URL <http://ocw.mit.edu/courses/earth-atmospheric-and-planetary-sciences/12-201-essentials-of-geophysics-fall-2004/lecture-notes/>.
- [ISS] NATIONAL AERONAUTICS AND SPACE ADMINISTRATION. «ISS Trajectory Data.» URL <http://spaceflight.nasa.gov/reldata/elements/>.
- [EGM96] —. «Spherical Harmonic Coefficients.» URL <ftp://cddis.gsfc.nasa.gov/pub/egm96/>.
- [Sol] PLATAFORMA SOLAR DE ALMERÍA. «Algoritmo para el cálculo de la posición solar.» URL <http://www.psa.es/sdg/sunpos.htm>.
- [Python] PYTHON SOFTWARE FOUNDATION. «Python programming Language.», 2013. Version 2.7, URL <https://www.python.org/>.
- [TkDocs] ROSEMAN, Mark. «TkDocs, Documentation for Tkinter.» URL <http://www.tkdocs.com/index.html>.
- [Juliana] UNITED STATES NAVAL OBSERVATORY. «Converting Between Julian Dates and Gregorian Calendar Dates.» URL http://aa.usno.navy.mil/faq/docs/JD_Formula.php.

BIBLIOGRAFÍA

[Resistencia] VALLADO, David A., y David FINKLEMAN. «A Critical Assessment of Satellite Drag and Atmospheric Density Modeling.»

Apéndice A

Datos de validación

En este anexo se presentan varios datos obtenidos mediante diferentes fuentes, datos que se usan durante el capítulo de verificación para validar la salida del simulador. Los datos se presentan tal y como se obtienen, sin tratamiento, para que el lector tenga un fácil acceso a ellos y si lo desea pueda repetir cualquiera de las pruebas o ampliarlas a su gusto.

Los datos están ordenados por su fuente y el tipo de propagación que se hace. En los casos en que el archivo de datos sea excesivamente largo, se seleccionan los puntos más relevantes para reflejarlos.

Se presentan cuatro fuentes de datos. Por un lado, la NASA publica con regularidad datos sobre la posición de la estación espacial internacional (ISS) y su órbita. Los datos aquí presentados y los que se han usado en el capítulo de validación son los correspondientes al 23 de Enero de 2015, 12:00:00 UTC.

Los datos obtenidos del STK provienen del software publicado por Analytical Graphics, Inc. Se usa una licencia de estudiante que, aunque tiene sus prestaciones muy limitadas, resulta suficiente para obtener los datos básicos con que comparar. Una vez creado el escenario en la fecha correcta, se añade un satélite y se accede a distintos datos mediante la herramienta de análisis *Report and Graph Manager* y se reflejan aquí tal como se obtienen.

Los datos originados en el simulador presentado en este trabajo se obtienen mediante la herramienta *resultados*, una vez se han introducido los objetos necesarios e integrado el escenario hasta el punto deseado.

Los datos correspondientes a la solución analítica se obtienen ayudándose de las funciones de cálculo de un objeto tipo *orbita* del simulador, una vez se ha validado su correcto funcionamiento.

Datos NASA

Posición ISS inicial

ISS TRAJECTORY DATA

Lift off time (UTC) : N/A
 Area (sq ft) : 17222.26
 Drag Coefficient (Cd) : 2.00
 Monthly MSFC 50% solar flux (F10.7-jansky) : 132.9
 Monthly MSFC 50% earth geomagnetic index (Kp) : 2.186
 ET - UTC (sec) : 67.18
 UT1 - UTC (sec) : 0.00

130

Maneuvers contained within the current ephemeris are as follows:

IMPULSIVE TIG (GMT)	M50 DVx(FPS)	LVLH DVx(FPS)	DVmag(FPS)
IMPULSIVE TIG (MET)	M50 DVy(FPS)	LVLH DVy(FPS)	Invar Sph HA
DT	M50 DVz(FPS)	LVLH DVz(FPS)	Invar Sph HP
028/18:42:08.283	1.2	-1.9	1.9
N/A	0.9	0.3	220.3
000/00:04:12.566	-1.2	-0.0	216.7

Coasting Arc #1 (Orbit 552)

Vector Time (GMT): 2015/023/12:00:00.000
Vector Time (MET): N/A
Weight (LBS) : 950073.3

M50 Cartesian

X =	-727392.70
Y =	6558570.91 meter
Z =	1569431.26
XDOT =	-4669.795875
YDOT =	-1903.904340 meter/sec
ZDOT =	5778.898520

M50 Keplerian

A =	6789964.81	meter
E =	.0011196	
I =	52.02376	
Wp =	37.73357	
RA =	85.62907	deg
TA =	339.33603	
MA =	339.38127	
Ha =	220.652	n.mi
Hp =	217.689	

M50 Cartesian

X =	-2386458.99
Y =	21517621.08 feet
Z =	5149052.70
XDOT =	-15320.852608
YDOT =	-6246.405316 feet/sec
ZDOT =	18959.640815

J2K Cartesian

X =	-808301.68
Y =	6549984.38 meter
Z =	1565701.11
XDOT =	-4676.230090
YDOT =	-1956.160859 meter/sec
ZDOT =	5756.198415

TDR Cartesian

TDR Cartesian

X =	-19583286.76	X =	-5968985.80
Y =	9239834.20	feet	meter
Z =	5131889.62		
XDOT =	-2119.045564	XDOT =	-645.885088
YDOT =	-14991.702956	feet/sec	meter/sec
ZDOT =	18862.830949		

The mean element set is posted at the UTC for which position is just north of the next ascending node relative to the above vector time

132

TWO LINE MEAN ELEMENT SET

ISS
 1 25544U 98067A 15023.56127426 .00016717 00000-0 10270-3 0 9001
 2 25544 51.6451 86.1253 0006010 294.3336 65.7188 15.53554402 5538

Satellite: ISS
 Catalog Number: 25544
 Epoch time: 15023.56127426 = yrday.fracday
 Element set: 900
 Inclination: 51.6451 deg
 RA of node: 86.1253 deg
 Eccentricity: .0006010

Arg of perigee: 294.3336 deg
Mean anomaly: 65.7188 deg
Mean motion: 15.53554402 rev/day
Decay rate: 1.67170E-04 rev/day^2
Epoch rev: 553
Checksum: 283

Datos STK

Integración a corto plazo

Satellite–Satellite1: J2000 Position & Velocity

Time (UTCG)	x (km)	y (km)	z (km)	vx (km/sec)	vy (km/sec)	vz (km/sec)
23 Jan 2015 12:00:00.000	-808.300178	6549.984541	1565.700474	-4.676235	-1.956159	5.756193
23 Jan 2015 12:01:40.000	-1269.771684	6312.995418	2130.105777	-4.543366	-2.778593	5.519890
23 Jan 2015 12:03:20.000	-1715.039479	5995.445897	2667.328584	-4.352505	-3.565644	5.213122
23 Jan 2015 12:05:00.000	-2138.419950	5601.382731	3170.510974	-4.106086	-4.307230	4.839808
23 Jan 2015 12:06:40.000	-2534.509165	5135.832803	3633.230207	-3.807263	-4.993852	4.404723
23 Jan 2015 12:08:20.000	-2898.252561	4604.738448	4049.581558	-3.459862	-5.616719	3.913441
23 Jan 2015 12:10:00.000	-3225.010080	4014.880816	4414.254399	-3.068337	-6.167864	3.372258
23 Jan 2015 12:11:40.000	-3510.615863	3373.792296	4722.600502	-2.637708	-6.640245	2.788112
23 Jan 2015 12:13:20.000	-3751.431728	2689.659202	4970.693651	-2.173497	-7.027840	2.168488
23 Jan 2015 12:15:00.000	-3944.393725	1971.216024	5155.379799	-1.681654	-7.325720	1.521327
23 Jan 2015 12:16:40.000	-4087.051141	1227.632663	5274.317098	-1.168480	-7.530115	0.854916
23 Jan 2015 12:18:20.000	-4177.597491	468.396194	5326.005316	-0.640545	-7.638457	0.177781
23 Jan 2015 12:20:00.000	-4214.893077	-296.811281	5309.804269	-0.104601	-7.649417	-0.501422
23 Jan 2015 12:21:40.000	-4198.478872	-1058.237048	5225.941088	0.432506	-7.562914	-1.174021
23 Jan 2015 12:23:20.000	-4128.581577	-1806.182821	5075.506257	0.963923	-7.380115	-1.831440
23 Jan 2015 12:25:00.000	-4006.109824	-2531.128770	4860.438528	1.482878	-7.103418	-2.465307
23 Jan 2015 12:26:40.000	-3832.641625	-3223.854919	4583.498980	1.982770	-6.736415	-3.067565
23 Jan 2015 12:28:20.000	-3610.403270	-3875.558361	4248.234603	2.457249	-6.283846	-3.630569
23 Jan 2015 12:30:00.000	-3342.240009	-4477.964797	3858.931944	2.900297	-5.751531	-4.147189
23 Jan 2015 12:31:40.000	-3031.578922	-5023.433013	3420.561464	3.306308	-5.146298	-4.610892
23 Jan 2015 12:33:20.000	-2682.384513	-5505.050989	2938.713375	3.670152	-4.475888	-5.015827
23 Jan 2015 12:35:00.000	-2299.107627	-5916.722496	2419.525809	3.987242	-3.748857	-5.356898
23 Jan 2015 12:36:40.000	-1886.628375	-6253.243119	1869.606288	4.253587	-2.974466	-5.629823
23 Jan 2015 12:38:20.000	-1450.193808	-6510.364847	1295.947492	4.465847	-2.162563	-5.831189
23 Jan 2015 12:40:00.000	-995.351162	-6684.848459	705.838433	4.621365	-1.323454	-5.958488

23	Jan	2015	12:41:40.000	-527.877505	-6774.503150	106.772127	4.718206	-0.467778	-6.010151
23	Jan	2015	12:43:20.000	-53.706678	-6778.212953	-493.649049	4.755172	0.393629	-5.985563
23	Jan	2015	12:45:00.000	421.145553	-6695.949702	-1087.809168	4.731823	1.249869	-5.885068
23	Jan	2015	12:46:40.000	890.657238	-6528.772433	-1668.174558	4.648477	2.090118	-5.709964
23	Jan	2015	12:48:20.000	1348.875923	-6278.813276	-2227.388414	4.506204	2.903762	-5.462486
23	Jan	2015	12:50:00.000	1789.993378	-5949.250050	-2758.363106	4.306815	3.680522	-5.145778
23	Jan	2015	12:51:40.000	2208.418456	-5544.265921	-3254.369043	4.052836	4.410590	-4.763849
23	Jan	2015	12:53:20.000	2598.847220	-5068.996640	-3709.119057	3.747479	5.084743	-4.321529
23	Jan	2015	12:55:00.000	2956.329476	-4529.466000	-4116.847263	3.394600	5.694461	-3.824404
23	Jan	2015	12:56:40.000	3276.330941	-3932.510298	-4472.381474	2.998652	6.232032	-3.278749
23	Jan	2015	12:58:20.000	3554.790270	-3285.692712	-4771.208285	2.564632	6.690644	-2.691450
23	Jan	2015	13:00:00.000	3788.170259	-2597.208605	-5009.530028	2.098015	7.064478	-2.069922
23	Jan	2015	13:01:40.000	3973.502604	-1875.782898	-5184.312915	1.604691	7.348775	-1.422012
23	Jan	2015	13:03:20.000	4108.425637	-1130.560726	-5293.325741	1.090891	7.539898	-0.755908
23	Jan	2015	13:05:00.000	4191.214589	-370.992687	-5335.168670	0.563110	7.635381	-0.080033
23	Jan	2015	13:06:40.000	4220.803953	393.283926	-5309.291704	0.028023	7.633960	0.597058
23	Jan	2015	13:08:20.000	4196.801669	1152.566486	-5216.002580	-0.507590	7.535595	1.266784
23	Jan	2015	13:10:00.000	4119.494905	1897.209496	-5056.463931	-1.036937	7.341468	1.920645
23	Jan	2015	13:11:40.000	3989.847314	2617.746650	-4832.679711	-1.553296	7.053977	2.550332
23	Jan	2015	13:13:20.000	3809.487775	3305.010965	-4547.470988	-2.050099	6.676706	3.147829
23	Jan	2015	13:15:00.000	3580.690700	3950.251479	-4204.441347	-2.521016	6.214382	3.705519
23	Jan	2015	13:16:40.000	3306.348097	4545.245018	-3807.932279	-2.960039	5.672823	4.216277
23	Jan	2015	13:18:20.000	2989.933715	5082.401599	-3362.969065	-3.361554	5.058862	4.673566
23	Jan	2015	13:20:00.000	2635.459660	5554.862076	-2875.197780	-3.720418	4.380268	5.071521
23	Jan	2015	13:21:40.000	2247.426003	5956.586734	-2350.814176	-4.032026	3.645644	5.405025
23	Jan	2015	13:23:20.000	1830.763990	6282.433648	-1796.485314	-4.292370	2.864323	5.669781
23	Jan	2015	13:25:00.000	1390.773541	6528.225715	-1219.264909	-4.498092	2.046249	5.862361
23	Jan	2015	13:26:40.000	933.055838	6690.805433	-626.503475	-4.646531	1.201847	5.980263
23	Jan	2015	13:28:20.000	463.441843	6768.076653	-25.754398	-4.735759	0.341897	6.021937
23	Jan	2015	13:30:00.000	-12.082321	6759.032671	575.322834	-4.764604	-0.522615	5.986813

Integración a largo plazo

Satellite-Satellite1: J2000 Position & Velocity

Time (UTCG)	x (km)	y (km)	z (km)	vx (km/sec)	vy (km/sec)	vz (km/sec)
23 Jan 2015 12:00:00.000	-808.293780	6549.978164	1565.730463	-4.676249	-1.956180	5.756175
24 Jan 2015 00:00:00.000	4094.471802	2064.184491	-5010.926985	-1.156199	7.283940	2.066858
24 Jan 2015 12:00:00.000	1247.024226	-6342.136310	-2103.766175	4.543298	2.723507	-5.524329
25 Jan 2015 00:00:00.000	-3955.812864	-2757.839022	4777.961512	1.646004	-6.995043	-2.662818
25 Jan 2015 12:00:00.000	-1657.471075	6041.772303	2598.339425	-4.380919	-3.464300	5.257437
26 Jan 2015 00:00:00.000	3789.498110	3369.242878	-4516.810340	-2.096753	6.635713	3.203512
26 Jan 2015 12:00:00.000	2062.024702	-5697.145206	-3081.786419	4.149574	4.155998	-4.907330
27 Jan 2015 00:00:00.000	-3562.973007	-3993.896860	4178.358877	2.543898	-6.188773	-3.732384
27 Jan 2015 12:00:00.000	-2433.020174	5265.181606	3515.525159	-3.890946	-4.818684	4.525098
28 Jan 2015 00:00:00.000	3316.799578	4525.168877	-3822.778383	-2.945127	5.693229	4.199118
28 Jan 2015 12:00:00.000	2786.031952	-4800.749271	-3923.813234	3.572746	5.405048	-4.073796
29 Jan 2015 00:00:00.000	-3012.384206	-5053.135591	3393.760651	3.328574	-5.109162	-4.636012
29 Jan 2015 12:00:00.000	-3100.487479	4254.695514	4276.541029	-3.228155	-5.958959	3.591781
30 Jan 2015 00:00:00.000	2697.196099	5480.660447	-2959.431858	-3.663600	4.497836	5.009365
30 Jan 2015 12:00:00.000	3387.090798	-3692.496723	-4592.681187	2.838173	6.415697	-3.060373

Propagación con J2

Satellite-Satellite2 : J2000 Classical Orbit Elements

Time (UTCG)	Semi-major Axis (km)	Eccentricity	Inclination (deg)	RAAN (deg)	Arg of Perigee (deg)
True Anomaly (deg)	Mean Anomaly (deg)				
23 Jan 2015 12:00:00.000 156.000	7370.000000 153.587	0.050000	47.000	86.000	37.000
23 Jan 2015 12:30:00.000 251.040	7370.000000 256.523	0.050000	47.000	85.914	37.083
23 Jan 2015 13:00:00.000 359.402	7370.000000 359.460	0.050000	47.000	85.828	37.167
23 Jan 2015 13:30:00.000 107.909	7370.000000 102.396	0.050000	47.000	85.743	37.250
23 Jan 2015 14:00:00.000 203.012	7370.000000 205.332	0.050000	47.000	85.657	37.334
23 Jan 2015 14:30:00.000 303.594	7370.000000 308.268	0.050000	47.000	85.571	37.417
23 Jan 2015 15:00:00.000 55.846	7370.000000 51.204	0.050000	47.000	85.485	37.500
23 Jan 2015 15:30:00.000 156.505	7370.000000 154.140	0.050000	47.000	85.399	37.584
23 Jan 2015 16:00:00.000 251.577	7370.000000 257.076	0.050000	47.000	85.314	37.667
23 Jan 2015 16:30:00.000 0.013	7370.000000 0.012	0.050000	47.000	85.228	37.751
23 Jan 2015 17:00:00.000 108.446	7370.000000 102.948	0.050000	47.000	85.142	37.834
23 Jan 2015 17:30:00.000 203.517	7370.000000 205.884	0.050000	47.000	85.056	37.917
23 Jan 2015 18:00:00.000 304.180	7370.000000 308.820	0.050000	47.000	84.970	38.001
23 Jan 2015 18:30:00.000 56.432	7370.000000 51.756	0.050000	47.000	84.884	38.084
23 Jan 2015 19:00:00.000 157.010	7370.000000 154.692	0.050000	47.000	84.799	38.168
23 Jan 2015 19:30:00.000 252.114	7370.000000 257.628	0.050000	47.000	84.713	38.251
23 Jan 2015 20:00:00.000	7370.000000	0.050000	47.000	84.627	38.334

0.625	0.565					
23 Jan 2015 20:30:00.000	108.983	7370.000000	0.050000	47.000	84.541	38.418
	103.501					
23 Jan 2015 21:00:00.000	204.022	7370.000000	0.050000	47.000	84.455	38.501
	206.437					
23 Jan 2015 21:30:00.000	304.766	7370.000000	0.050000	47.000	84.370	38.585
	309.373					
23 Jan 2015 22:00:00.000	57.017	7370.000000	0.050000	47.000	84.284	38.668
	52.309					
23 Jan 2015 22:30:00.000	157.514	7370.000000	0.050000	47.000	84.198	38.751
	155.245					
23 Jan 2015 23:00:00.000	252.652	7370.000000	0.050000	47.000	84.112	38.835
	258.181					
23 Jan 2015 23:30:00.000	1.236	7370.000000	0.050000	47.000	84.026	38.918
	1.117					
24 Jan 2015 00:00:00.000	109.520	7370.000000	0.050000	47.000	83.941	39.002
	104.053					
24 Jan 2015 00:30:00.000	204.527	7370.000000	0.050000	47.000	83.855	39.085
	206.989					
24 Jan 2015 01:00:00.000	305.353	7370.000000	0.050000	47.000	83.769	39.168
	309.925					
24 Jan 2015 01:30:00.000	57.602	7370.000000	0.050000	47.000	83.683	39.252
	52.861					
24 Jan 2015 02:00:00.000	158.019	7370.000000	0.050000	47.000	83.597	39.335
	155.797					
24 Jan 2015 02:30:00.000	253.191	7370.000000	0.050000	47.000	83.511	39.419
	258.733					
24 Jan 2015 03:00:00.000	1.848	7370.000000	0.050000	47.000	83.426	39.502
	1.670					
24 Jan 2015 03:30:00.000	110.056	7370.000000	0.050000	47.000	83.340	39.585
	104.606					
24 Jan 2015 04:00:00.000	205.033	7370.000000	0.050000	47.000	83.254	39.669
	207.542					
24 Jan 2015 04:30:00.000	305.940	7370.000000	0.050000	47.000	83.168	39.752
	310.478					
24 Jan 2015 05:00:00.000	58.187	7370.000000	0.050000	47.000	83.082	39.835
	53.414					
24 Jan 2015 05:30:00.000	158.523	7370.000000	0.050000	47.000	82.997	39.919
	156.350					
24 Jan 2015 06:00:00.000	253.730	7370.000000	0.050000	47.000	82.911	40.002
	259.286					
24 Jan 2015 06:30:00.000		7370.000000	0.050000	47.000	82.825	40.086

Datos STK

2.459	2.222					
24 Jan 2015 07:00:00.000	110.591	105.158	7370.000000	0.050000	47.000	82.739
24 Jan 2015 07:30:00.000	205.538	208.094	7370.000000	0.050000	47.000	82.653
24 Jan 2015 08:00:00.000	306.528	311.030	7370.000000	0.050000	47.000	82.568
24 Jan 2015 08:30:00.000	58.771	53.966	7370.000000	0.050000	47.000	82.482
24 Jan 2015 09:00:00.000	159.027	156.902	7370.000000	0.050000	47.000	82.396
24 Jan 2015 09:30:00.000	254.269	259.838	7370.000000	0.050000	47.000	82.310
24 Jan 2015 10:00:00.000	3.070	2.775	7370.000000	0.050000	47.000	82.224
24 Jan 2015 10:30:00.000	111.126	105.711	7370.000000	0.050000	47.000	82.138
24 Jan 2015 11:00:00.000	206.044	208.647	7370.000000	0.050000	47.000	82.053
24 Jan 2015 11:30:00.000	307.116	311.583	7370.000000	0.050000	47.000	81.967
24 Jan 2015 12:00:00.000	59.354	54.519	7370.000000	0.050000	47.000	81.881
						41.003

Datos simulador

Integración a corto plazo

Adams-Bashforth 4

Archivo de resultados

ISS AB4 1.0, clase sólido

Tiempo (s)	Posicion (Km)			Velocidad (Km/s)		
	X	Y	Z	Vx	Vy	Vz
0.000000	-808.293780	6549.978164	1565.730463	-4.676249	-1.956180	5.756175
100.000000	-1269.768847	6313.003911	2130.137809	-4.543382	-2.778609	5.519869
200.000000	-1715.038277	5995.453049	2667.358434	-4.352522	-3.565655	5.213099
300.000000	-2138.420571	5601.389052	3170.538529	-4.106106	-4.307236	4.839784
400.000000	-2534.511821	5135.838790	3633.255443	-3.807284	-4.993853	4.404700
500.000000	-2898.257493	4604.744578	4049.604543	-3.459886	-5.616715	3.913420
600.000000	-3225.017551	4014.887528	4414.275298	-3.068363	-6.167856	3.372239
700.000000	-3510.626161	3373.799982	4722.619573	-2.637737	-6.640234	2.788095
800.000000	-3751.445168	2689.668192	4970.711250	-2.173530	-7.027826	2.168475
900.000000	-3944.410640	1971.226562	5155.396371	-1.681690	-7.325704	1.521319
1000.000000	-4087.071884	1227.644898	5274.333174	-1.168520	-7.530097	0.854913
1100.000000	-4177.622423	468.410157	5326.021497	-0.640589	-7.638440	0.177785
1200.000000	-4214.922563	-296.795693	5309.821220	-0.104648	-7.649402	-0.501411
1300.000000	-4198.513269	-1058.220086	5225.959517	0.432456	-7.562902	-1.174003
1400.000000	-4128.621222	-1806.164898	5075.526894	0.963869	-7.380108	-1.831411
1500.000000	-4006.155019	-2531.110469	4860.462102	1.482822	-7.103418	-2.465274
1600.000000	-3832.692622	-3223.836997	4583.526193	1.982711	-6.736423	-3.067525
1700.000000	-3610.460252	-3875.541746	4248.266104	2.457188	-6.283864	-3.630525
1800.000000	-3342.303071	-4477.950584	3858.968297	2.900236	-5.751561	-4.147138

Datos simulador

						Datos simulador
1900.000000	-3031.648056	-5023.422446	3420.603123	3.306248	-5.146341	-4.610837
2000.000000	-2682.459588	-5505.045444	2938.760654	3.670094	-4.475946	-5.015770
2100.000000	-2299.188375	-5916.723450	2419.578859	3.987187	-3.748930	-5.356840
2200.000000	-1886.714374	-6253.252122	1869.665071	4.253537	-2.974555	-5.629767
2300.000000	-1450.284478	-6510.383476	1296.011768	4.465804	-2.162667	-5.831136
2400.000000	-995.445753	-6684.878280	705.907743	4.621330	-1.323573	-5.958441
2500.000000	-527.975096	-6774.545670	106.845791	4.718181	-0.467912	-6.010112
2600.000000	-53.806183	-6778.269569	-493.571935	4.755159	0.393481	-5.985534
2700.000000	421.045382	-6696.021653	-1087.729726	4.731823	1.249710	-5.885051
2800.000000	890.557793	-6528.860752	-1668.094110	4.648492	2.089950	-5.709961
2900.000000	1348.778724	-6278.918743	-2227.308466	4.506234	2.903587	-5.462499
3000.000000	1789.900049	-5949.373149	-2758.285317	4.306862	3.680344	-5.145808
3100.000000	2208.330696	-5544.406807	-3254.295196	4.052900	4.410413	-4.763898
3200.000000	2598.766770	-5069.155109	-3709.051015	3.747561	5.084570	-4.321597
3300.000000	2956.258087	-4529.641467	-4116.786930	3.394699	5.694296	-3.824490
3400.000000	3276.270332	-3932.701789	-4472.330750	2.998768	6.231877	-3.278854
3500.000000	3554.742092	-3285.898858	-4771.169016	2.564764	6.690506	-2.691574
3600.000000	3788.136054	-2597.427654	-5009.503964	2.098162	7.064359	-2.070062
3700.000000	3973.483768	-1876.012733	-5184.301658	1.604851	7.348679	-1.422168
3800.000000	4108.423384	-1130.798891	-5293.330705	1.091062	7.539828	-0.756077
3900.000000	4191.229915	-371.236429	-5335.191039	0.563290	7.635339	-0.080212
4000.000000	4220.837609	393.037616	-5309.332401	0.028210	7.633950	0.596872
4100.000000	4196.854138	1152.320818	-5216.062230	-0.507400	7.535618	1.266592
4200.000000	4119.566379	1896.967826	-5056.542844	-1.036747	7.341525	1.920452
4300.000000	3989.937683	2617.512415	-4832.777866	-1.553109	7.054069	2.550141
4400.000000	3809.596624	3304.787620	-4547.588022	-2.049917	6.676832	3.147643
4500.000000	3580.817301	3950.042432	-4204.576558	-2.520844	6.214542	3.705342
4600.000000	3306.491421	4545.053568	-3808.084630	-2.959878	5.673015	4.216111
4700.000000	2990.092441	5082.230873	-3363.137197	-3.361407	5.059084	4.673416
4800.000000	2635.632195	5554.714971	-2875.380033	-3.720289	4.380518	5.071389
4900.000000	2247.610504	5956.465868	-2351.008616	-4.031916	3.645918	5.404914
5000.000000	1830.958390	6282.341309	-1796.689759	-4.292282	2.864619	5.669692
5100.000000	1390.975581	6528.163825	-1219.476970	-4.498027	2.046561	5.862297
5200.000000	933.263100	6690.775517	-626.720586	-4.646492	1.202173	5.980225
5300.000000	463.651785	6768.079810	-25.973858	-4.735745	0.342232	6.021927
5400.000000	-11.872329	6759.069560	575.103820	-4.764617	-0.522276	5.986831

Runge-Kutta 4

Archivo de resultados

ISS RK4 1.0, clase sólido

Tiempo (s)	Posicion (Km)			Velocidad (Km/s)		
	X	Y	Z	Vx	Vy	Vz
0.000000	-808.293780	6549.978164	1565.730463	-4.676249	-1.956180	5.756175
100.000000	-1269.833405	6313.399937	2130.252330	-4.543391	-2.778561	5.519884
200.000000	-1715.200776	5996.253774	2667.630444	-4.352571	-3.565464	5.213178
300.000000	-2138.715125	5602.590311	3171.010858	-4.106245	-4.306819	4.839995
400.000000	-2534.976744	5137.438856	3633.976437	-3.807579	-4.993149	4.405132
500.000000	-2898.936576	4606.742285	4050.629529	-3.460418	-5.615688	3.914178
600.000000	-3225.961203	4017.279966	4415.667832	-3.069228	-6.166499	3.373446
700.000000	-3511.892372	3376.579756	4724.452433	-2.639041	-6.638574	2.789882
800.000000	-3753.100230	2692.820239	4973.067088	-2.175387	-7.025930	2.170983
900.000000	-3946.529578	1974.724560	5158.367958	-1.684219	-7.323682	1.524688
1000.000000	-4089.738515	1231.447296	5278.023146	-1.171841	-7.528107	0.859281
1100.000000	-4180.928969	472.455907	5330.541513	-0.644815	-7.636689	0.183278
1200.000000	-4218.968742	-292.591605	5315.290438	-0.109881	-7.648149	-0.494684
1300.000000	-4203.404797	-1053.971173	5232.502324	0.426130	-7.562458	-1.165960
1400.000000	-4134.467654	-1802.017600	5083.269809	0.956392	-7.380833	-1.822011
1500.000000	-4013.066896	-2527.248267	4869.529822	1.474167	-7.105716	-2.454511
1600.000000	-3840.777891	-3220.484000	4594.036767	1.972892	-6.740737	-3.055455
1700.000000	-3619.819971	-3872.965490	4260.325260	2.446267	-6.290666	-3.617265
1800.000000	-3353.026424	-4476.463789	3872.662994	2.888330	-5.761339	-4.132880
1900.000000	-3043.806752	-5023.383476	3435.994423	3.293535	-5.159585	-4.595844
2000.000000	-2696.101757	-5506.857216	2955.876082	3.656817	-4.493130	-5.000397
2100.000000	-2314.332089	-5920.830582	2438.404444	3.973659	-3.770496	-5.341507
2200.000000	-1903.340981	-6260.136123	1890.137309	4.240142	-3.000889	-5.614996
2300.000000	-1468.331958	-6520.555856	1318.009780	4.452992	-2.194077	-5.817524

Datos simulados

							Datos simulador
2400.000000	-1014.802328	-6698.871496	729.245942	4.609619	-1.360268	-5.946659	
2500.000000	-548.473356	-6792.901913	131.267374	4.708145	-0.509979	-6.000897	
2600.000000	-75.217995	-6801.527463	-468.400325	4.747425	0.346098	-5.979672	
2700.000000	399.012908	-6724.701028	-1062.221846	4.727054	1.197226	-5.883364	
2800.000000	868.265012	-6563.445728	-1642.746142	4.647374	2.032759	-5.713286	
2900.000000	1326.654591	-6319.839459	-2202.698940	4.509462	2.842273	-5.471664	
3000.000000	1768.441459	-5996.986532	-2735.072680	4.315114	3.615687	-5.161610	
3100.000000	2188.099837	-5598.976861	-3233.212829	4.066824	4.343394	-4.787074	
3200.000000	2580.386352	-5130.833259	-3690.899568	3.767748	5.016369	-4.352797	
3300.000000	2940.404362	-4598.447539	-4102.423915	3.421663	5.626281	-3.864252	
3400.000000	3263.663887	-4008.506231	-4462.657406	3.032922	6.165590	-3.327572	
3500.000000	3546.136437	-3368.406834	-4767.114558	2.606398	6.627638	-2.749476	
3600.000000	3784.304140	-2686.165622	-5012.007399	2.147425	7.006731	-2.137185	
3700.000000	3975.202590	-1970.318096	-5194.291448	1.661730	7.298199	-1.498337	
3800.000000	4116.456934	-1229.813274	-5311.702632	1.155368	7.498460	-0.840890	
3900.000000	4206.310787	-473.903040	-5362.784707	0.634644	7.605057	-0.173028	
4000.000000	4243.647624	287.972118	-5346.906866	0.106039	7.616690	0.496940	
4100.000000	4228.004410	1046.299683	-5264.271329	-0.423870	7.533231	1.160678	
4200.000000	4159.577281	1791.612588	-5115.910797	-0.948490	7.355728	1.809930	
4300.000000	4039.219197	2514.606241	-4903.675786	-1.461292	7.086392	2.436616	
4400.000000	3868.429568	3206.253692	-4630.211956	-1.955891	6.728570	3.032931	
4500.000000	3649.335962	3857.917554	-4298.927679	-2.426124	6.286710	3.591444	
4600.000000	3384.668051	4461.457339	-3913.952185	-2.866126	5.766302	4.105190	
4700.000000	3077.724110	5009.330862	-3480.084745	-3.270407	5.173820	4.567753	
4800.000000	2732.330413	5494.688480	-3002.735484	-3.633912	4.516637	4.973350	
4900.000000	2352.793999	5911.458941	-2487.858481	-3.952095	3.802939	5.316903	
5000.000000	1943.849341	6254.425759	-1941.877951	-4.220969	3.041624	5.594106	
5100.000000	1510.599566	6519.293097	-1371.608388	-4.437162	2.242195	5.801477	
5200.000000	1058.452914	6702.740281	-784.169633	-4.597957	1.414638	5.936405	
5300.000000	593.055213	6802.464178	-186.897904	-4.701331	0.569299	5.997188	
5400.000000	120.219219	6817.208853	412.746093	-4.745979	-0.283243	5.983051	

Predictor corrector

Archivo de resultados

ISS PC4 1.0, clase sólido

Tiempo (s)	Posicion (Km)			Velocidad (Km/s)		
	X	Y	Z	Vx	Vy	Vz
0.000000	-808.293780	6549.978164	1565.730463	-4.676249	-1.956180	5.756175
100.000000	-1269.815715	6313.235762	2130.216338	-4.543558	-2.778669	5.520087
200.000000	-1715.169518	5995.913090	2667.562652	-4.352891	-3.565748	5.213559
300.000000	-2138.672946	5602.061674	3170.913670	-4.106686	-4.307336	4.840510
400.000000	-2534.923409	5136.707990	3633.848373	-3.808102	-4.993937	4.405728
500.000000	-2898.868429	4605.793673	4050.464676	-3.460979	-5.616767	3.914799
600.000000	-3225.870923	4016.098627	4415.455682	-3.069781	-6.167869	3.374031
700.000000	-3511.769043	3375.153066	4724.178126	-2.639540	-6.640216	2.790378
800.000000	-3752.929749	2691.139657	4972.712067	-2.175792	-7.027807	2.171340
900.000000	-3946.295370	1972.787120	5157.911000	-1.684499	-7.325740	1.524871
1000.000000	-4089.422526	1229.256730	5277.441711	-1.171972	-7.530281	0.859267
1100.000000	-4180.512879	470.023298	5329.813332	-0.644788	-7.638908	0.183061
1200.000000	-4218.435315	-295.247731	5314.395234	-0.109702	-7.650341	-0.495093
1300.000000	-4202.739287	-1056.825363	5231.423544	0.426444	-7.564557	-1.166532
1400.000000	-4133.659167	-1805.038604	5081.996254	0.956810	-7.382786	-1.822702
1500.000000	-4012.109604	-2530.400769	4868.057042	1.474645	-7.107486	-2.455263
1600.000000	-3839.672001	-3223.730865	4592.368103	1.973380	-6.742311	-3.056204
1700.000000	-3618.572372	-3876.270443	4258.472436	2.446709	-6.292056	-3.617930
1800.000000	-3351.650917	-4479.794374	3870.646166	2.888668	-5.762586	-4.133410
1900.000000	-3042.323833	-5026.714145	3433.841645	3.293714	-5.160759	-4.596169
2000.000000	-2694.537896	-5510.172367	2953.622150	3.656794	-4.494329	-5.000461
2100.000000	-2312.718524	-5924.127344	2436.089132	3.973402	-3.771843	-5.341298
2200.000000	-1901.712037	-6263.426722	1887.803042	4.239637	-3.002524	-5.614492
2300.000000	-1466.722928	-6523.869367	1315.698819	4.452247	-2.196156	-5.816754

Datos simulados

							Datos simulador
2400.000000	-1013.247018	-6702.254835	726.997183	4.608667	-1.362951	-5.945677	
2500.000000	-547.001368	-6796.419936	129.112907	4.707046	-0.513421	-5.999791	
2600.000000	-73.851871	-6805.262087	-470.438757	4.746265	0.341755	-5.978563	
2700.000000	400.260540	-6728.749317	-1064.136306	4.725946	1.191866	-5.882405	
2800.000000	869.394169	-6567.916954	-1644.545707	4.646456	2.026300	-5.712659	
2900.000000	1327.680388	-6324.851171	-2204.412478	4.508893	2.834674	-5.471574	
3000.000000	1769.396145	-6002.659738	-2736.751033	4.315070	3.606961	-5.162277	
3100.000000	2189.034301	-5605.430459	-3234.930253	4.067492	4.333611	-4.788729	
3200.000000	2581.370974	-5138.177895	-3692.754307	3.769316	5.005660	-4.355669	
3300.000000	2941.529130	-4606.779108	-4104.537803	3.424318	5.614843	-3.868560	
3400.000000	3265.037711	-4017.899265	-4465.174456	3.036838	6.153684	-3.333514	
3500.000000	3547.885631	-3378.908033	-4770.198503	2.611728	6.615587	-2.757219	
3600.000000	3786.570071	-2697.787786	-5015.838213	2.154293	6.994912	-2.146857	
3700.000000	3978.138523	-1983.034733	-5199.060910	1.670226	7.287038	-1.510014	
3800.000000	4120.224143	-1243.554113	-5317.609028	1.165537	7.488420	-0.854592	
3900.000000	4211.074022	-488.550691	-5370.026817	0.646484	7.596625	-0.188711	
4000.000000	4249.570062	272.584189	-5355.677407	0.119494	7.610364	0.479387	
4100.000000	4235.242230	1030.387701	-5274.750045	-0.408911	7.529505	1.141438	
4200.000000	4168.274037	1775.441451	-5128.257419	-0.932196	7.355074	1.789254	
4300.000000	4049.500156	2498.486886	-4918.023089	-1.443886	7.089247	2.414823	
4400.000000	3880.396216	3190.538852	-4646.659149	-1.937650	6.735317	3.010404	
4500.000000	3663.060830	3842.995980	-4317.534348	-2.407374	6.297665	3.568622	
4600.000000	3400.190071	4447.746599	-3934.732999	-2.847236	5.781702	4.082558	
4700.000000	3095.044641	4997.268889	-3503.005137	-3.251780	5.193808	4.545834	
4800.000000	2751.410086	5484.724080	-3027.708438	-3.615978	4.541255	4.952692	
4900.000000	2373.550497	5904.041519	-2514.742577	-3.935300	3.832122	5.298066	
5000.000000	1966.156203	6249.994527	-1970.476747	-4.205764	3.075198	5.577645	
5100.000000	1534.286076	6518.266073	-1401.671184	-4.423993	2.279872	5.787932	
5200.000000	1083.305078	6705.503365	-815.393624	-4.587252	1.456022	5.926287	
5300.000000	618.817842	6809.360620	-218.931678	-4.693492	0.613892	5.990962	
5400.000000	146.599039	6828.529375	380.297794	-4.741370	-0.236034	5.981129	

Cambio de Δt

Archivo de resultados

ISS AB4 0.1, clase sólido

Tiempo (s)	Posición (Km)			Velocidad (Km/s)		
	X	Y	Z	Vx	Vy	Vz
0.000000	-830.792796	6732.298089	1609.312877	-4.612494	-1.929510	5.677697
100.000000	-1286.256582	6500.165452	2166.513526	-4.487847	-2.708608	5.455384
200.000000	-1726.602817	6191.635867	2698.250877	-4.310440	-3.455941	5.168931
300.000000	-2146.654692	5810.330449	3198.273204	-4.082358	-4.162692	4.821707
400.000000	-2541.474175	5360.728787	3660.702118	-3.806288	-4.820522	4.417802
500.000000	-2906.420686	4848.116027	4080.102434	-3.485487	-5.421674	3.961982
5000.000000	2715.440012	5701.696669	-2964.205594	-3.663961	4.332600	4.996130
5100.000000	2333.835208	6100.670514	-2448.185518	-3.960682	3.639029	5.314189
5200.000000	1924.850046	6428.070909	-1903.443609	-4.211026	2.902530	5.569997
5300.000000	1493.275565	6680.032848	-1336.363684	-4.412021	2.131736	5.760505
5400.000000	1044.171398	6853.576106	-753.596177	-4.561271	1.335693	5.883434

ISS AB4 1.5, clase sólido

Tiempo (s)	Posición (Km)			Velocidad (Km/s)		
	X	Y	Z	Vx	Vy	Vz
0.000000	-808.293780	6549.978164	1565.730463	-4.676249	-1.956180	5.756175
150.000000	-1494.792676	6164.090636	2402.588999	-4.455068	-3.177190	5.375070
300.000000	-2138.423745	5601.412155	3170.544461	-4.106110	-4.307221	4.839791
450.000000	-2720.732444	4878.105259	3847.580018	-3.639402	-5.313751	4.165720
600.000000	-3225.023388	4014.916623	4414.285096	-3.068378	-6.167831	3.372259
4800.000000	2635.830899	5554.518562	-2875.647787	-3.720123	4.380854	5.071206
4950.000000	2042.760996	6129.030051	-2077.451433	-4.168601	3.260880	5.545997

Datos simulador

5100.000000	1391.213464	6528.081664	-1219.784841	-4.497936	2.046980	5.862216
5250.000000	699.821705	6740.162279	-327.184331	-4.698564	0.773905	6.010641
5400.000000	-11.620536	6759.119175	574.789261	-4.764617	-0.521824	5.986869

ISS AB4 0.5, clase solido

Tiempo (s)	Posicion (Km)			Velocidad (Km/s)		
	X	Y	Z	Vx	Vy	Vz
0.000000	-808.293780	6549.978164	1565.730463	-4.676249	-1.956180	5.756175
50.000000	-1040.692062	6441.769934	1850.887725	-4.617187	-2.371169	5.647039
100.000000	-1269.767254	6312.991220	2130.134742	-4.543381	-2.778612	5.519868
150.000000	-1494.788311	6164.056307	2402.580630	-4.455065	-3.177202	5.375066
200.000000	-1715.036587	5995.439905	2667.355206	-4.352521	-3.565661	5.213097
250.000000	-1929.808675	5807.679808	2923.612798	-4.236076	-3.942745	5.034478
300.000000	-2138.418676	5601.375171	3170.534979	-4.106103	-4.307245	4.839780
5000.000000	1830.822025	6282.415074	-1796.511061	-4.292353	2.864381	5.669763
5050.000000	1613.399776	6415.542713	-1510.309634	-4.402228	2.459298	5.775255
5100.000000	1390.832909	6528.213146	-1219.292316	-4.498082	2.046311	5.862346
5150.000000	1163.830453	6620.063967	-924.386487	-4.579604	1.626735	5.930753
5200.000000	933.115822	6690.799213	-626.532226	-4.646529	1.201912	5.980251
5250.000000	699.424505	6740.190349	-326.679307	-4.698639	0.773198	6.010677
5300.000000	463.501703	6768.077041	-25.784144	-4.735764	0.341964	6.021929
5350.000000	226.099945	6774.367705	275.193275	-4.757781	-0.090412	6.013965
5400.000000	-12.023336	6759.039837	575.292472	-4.764616	-0.522547	5.986808

Archivo de resultados

ISS AB4 5.0, clase solido

Tiempo (s)	Posicion (Km)			Velocidad (Km/s)		
	X	Y	Z	Vx	Vy	Vz

0.000000	-808.293780	6549.978164	1565.730463	-4.676249	-1.956180	5.756175
50.000000	-1040.746017	6442.183700	1850.990297	-4.617194	-2.371128	5.647050
100.000000	-1269.821778	6313.408311	2130.238309	-4.543398	-2.778520	5.519897
150.000000	-1494.844002	6164.479233	2402.686158	-4.455095	-3.177060	5.375116
200.000000	-1715.094184	5995.871094	2667.463829	-4.352567	-3.565472	5.213171
250.000000	-1929.869055	5808.121596	2923.725823	-4.236141	-3.942510	5.034580
300.000000	-2138.482858	5601.829789	3170.653880	-4.106190	-4.306966	4.839914
5300.000000	468.453658	6768.166406	-32.043943	-4.735141	0.350790	6.021872
5350.000000	231.075244	6774.900176	268.940648	-4.757471	-0.081519	6.014310
5400.000000	-7.040423	6760.018004	569.067141	-4.764623	-0.513617	5.987556

ISS AB4 10.0, clase solido

Tiempo (s)	Posicion (Km)			Velocidad (Km/s)		
	X	Y	Z	Vx	Vy	Vz
0.000000	-808.293780	6549.978164	1565.730463	-4.676249	-1.956180	5.756175
50.000000	-1040.920116	6443.431038	1851.313995	-4.617207	-2.371045	5.647074
100.000000	-1269.997245	6314.663471	2130.564387	-4.543442	-2.778287	5.519971
150.000000	-1495.022612	6165.749828	2403.017493	-4.455179	-3.176677	5.375254
200.000000	-1715.278135	5997.164481	2667.803813	-4.352699	-3.564945	5.213382
250.000000	-1930.060966	5809.444847	2924.078354	-4.236329	-3.941844	5.034873
300.000000	-2138.685765	5603.189665	3171.023363	-4.106443	-4.306169	4.840300
5100.000000	1409.781776	6521.619750	-1243.820391	-4.490755	2.079560	5.855828
5150.000000	1183.117160	6615.151939	-949.202809	-4.573424	1.660722	5.925747
5200.000000	952.682141	6687.602595	-651.560282	-4.641530	1.236523	5.976792
5250.000000	719.210605	6738.737483	-351.841025	-4.694851	0.808318	6.008793
5300.000000	483.446370	6768.390458	-51.000184	-4.733213	0.377476	6.021643
5350.000000	246.140798	6776.464036	250.003238	-4.756488	-0.054627	6.015294
5400.000000	8.050381	6762.929749	550.209465	-4.764598	-0.486608	5.989763

Datos simulador

ISS AB4 2.0, clase solido

Tiempo (s)	Posicion (Km)			Velocidad (Km/s)		
	X	Y	Z	Vx	Vy	Vz

0.000000	-808.293780	6549.978164	1565.730463	-4.676249	-1.956180	5.756175
50.000000	-1040.700002	6441.832774	1850.902981	-4.617188	-2.371162	5.647041
100.000000	-1269.775288	6313.054614	2130.150162	-4.543384	-2.778597	5.519872
150.000000	-1494.796530	6164.120635	2402.596362	-4.455070	-3.177179	5.375074
200.000000	-1715.045104	5995.505535	2667.371423	-4.352528	-3.565631	5.213109
250.000000	-1929.817624	5807.747095	2923.629699	-4.236086	-3.942708	5.034494
300.000000	-2138.428212	5601.444451	3170.552790	-4.106116	-4.307201	4.839801
5100.000000	1391.546399	6527.966637	-1220.215737	-4.497808	2.047565	5.862103
5150.000000	1164.556563	6619.880884	-925.320625	-4.579373	1.628017	5.930567
5200.000000	933.852358	6690.680824	-627.474198	-4.646343	1.203218	5.980123
5250.000000	700.169211	6740.137707	-327.626170	-4.698499	0.774522	6.010609
5300.000000	464.252273	6768.090981	-26.732911	-4.735670	0.343302	6.021921
5350.000000	226.854028	6774.448842	274.245629	-4.757734	-0.089064	6.014018
5400.000000	-11.268124	6759.188559	574.348998	-4.764618	-0.521193	5.986923

Integración a largo plazo

Archivo de resultados

ISS AB4 1.0, clase sólido

Tiempo (s)	Posicion (Km)			Velocidad (Km/s)		
	X	Y	Z	Vx	Vy	Vz
0.000000	-808.293780	6549.978164	1565.730463	-4.676249	-1.956180	5.756175
32400.000000	3424.401645	4307.385928	-3977.005953	-2.783885	5.904480	4.012555
86400.000000	1244.019813	-6343.989622	-2100.117354	4.544323	2.718155	-5.526070
129600.000000	-3958.027274	-2748.611192	4781.528719	1.639326	-6.999646	-2.654747
172800.000000	-1648.798969	6048.686522	2587.936925	-4.385084	-3.448971	5.263979
216000.000000	3795.779229	3349.506961	-4526.395110	-2.082368	6.648393	3.186357
259200.000000	2045.579885	-5713.676120	-3062.343632	4.159869	4.127238	-4.922743
302400.000000	-3576.431524	-3961.194469	4198.101814	2.519845	-6.215477	-3.704155
345600.000000	-2407.301776	5296.970856	3485.610067	-3.911330	-4.774067	4.554593
388800.000000	3341.070355	4478.047314	-3857.400152	-2.910068	5.740520	4.158666
432000.000000	2750.528160	-4854.150876	-3883.304601	3.607507	5.344254	-4.122826
475200.000000	-3051.782797	-4992.093915	3448.681674	3.282652	-5.185061	-4.584182
518400.000000	-3055.434518	4336.941576	4226.337094	-3.282635	-5.882761	3.667045
561600.000000	2756.133470	5407.140756	-3040.115166	-3.607295	4.610052	4.947403
604800.000000	3333.996471	-3810.483443	-4535.261664	2.917002	6.327506	-3.167446

Datos simulador

Propagación con J2

Archivo de resultados

Objeto 1, clase solido

Tiempo (s)	Posicion (Km)			Velocidad (Km/s)			
Orbita prevista (Km, grados)	X	Y	Z	Vx	Vy	Vz	
Semieje mayor	Excentricidad	Inclinacion	Longitud nodo asc	Argumento perigeo	Anomalía verdadera		
0.000000	655.362443	-7570.164927	-1267.361828	4.781249	1.104002	-5.032187	
7370.000000	0.050000	47.000000	86.000000	37.000000	156.000000		
1800.000000	4996.003461	1969.377296	-5187.704554	-1.276734	6.886315	1.891751	
7362.648324	0.050573	46.971689	85.899882	37.097105	251.017736		
3600.000000	-2439.376172	5795.088192	3058.193507	-4.554719	-4.313984	4.534871	
7368.198944	0.051077	46.990460	85.845853	37.189382	359.548322		
5400.000000	-3339.907038	-5915.620499	3095.592819	3.595066	-4.686172	-4.219299	
7368.000970	0.050712	46.990739	85.706174	37.556139	107.898436		
7200.000000	4278.046142	-4121.630307	-4904.129898	2.727884	6.001594	-2.429174	
7364.091420	0.050737	46.976482	85.671242	36.939064	203.614681		
9000.000000	2077.208619	6627.407862	-1669.964965	-4.784099	2.509258	5.323428	

Datos simulador

	7370.703726	0.050952	46.999884	85.554585	36.443784	304.914022	
	10800.000000 7361.322592	-4888.316228 0.050480	-100.464265 46.967125	5211.180310 85.468412	-0.459458 36.964777	-7.561824 56.938016	-0.149347
	12600.000000 7369.913008	719.663176 0.050003	-7540.392214 46.999584	-1417.898369 85.398813	4.776892 37.557315	1.248900 157.013813	-4.998571
	14400.000000 7362.802927	4981.442235 0.050554	2119.725943 46.972300	-5132.579215 85.296462	-1.343727 37.665280	6.839403 252.086453	2.035563
152	16200.000000 7367.861474	-2506.541928 0.051036	5697.783174 46.989349	3184.132343 85.243697	-4.518373 37.789131	-4.457820 0.731477	4.430643
	18000.000000 7368.235644	-3294.475032 0.050690	-6013.750133 46.991648	2968.767897 85.104305	3.641210 38.171186	-4.559482 108.918094	-4.306490
	19800.000000 7363.896574	4317.757171 0.050758	-3986.040932 46.975720	-4976.329680 85.068272	2.679959 37.544240	6.076303 204.582122	-2.301134
	21600.000000 7370.902700	2011.014604 0.050986	6677.918087 47.000511	-1518.264724 84.953316	-4.821117 37.025328	2.348764 306.065770	5.371636

Datos simulador

	23400.000000 7361.384569	-4900.209650 0.050500	-264.586769 46.967296	5202.382690 84.864451	-0.385474 37.545315	-7.554864 58.089057	-0.313123
	25200.000000 7369.790899	784.191149 0.050009	-7506.999154 46.999125	-1567.383389 84.797594	4.772367 38.113872	1.392861 158.027706	-4.961066
	27000.000000 7362.970186	4966.508001 0.050536	2268.749048 46.972956	-5073.271439 84.693086	-1.411120 38.231086	6.788864 253.159406	2.178041
153	28800.000000 7367.519046	-2573.690440 0.050993	5597.122813 46.988223	3307.052770 84.641446	-4.481343 38.386125	-4.598766 1.917396	4.322318
	30600.000000 7368.462749	-3248.664334 0.050668	-6108.658906 46.992532	2839.425291 84.502508	3.687218 38.784435	-4.430887 109.937648	-4.389932
	32400.000000 7363.708265	4357.476124 0.050776	-3848.721380 46.974985	-5044.689461 84.465243	2.631661 38.150306	6.148030 205.549454	-2.171165
	34200.000000 7371.086519	1944.313954 0.051021	6724.474671 47.001084	-1365.269630 84.352095	-4.857882 37.609979	2.186746 307.216232	5.415139

Datos simulador

	36000.000000 7361.464954	-4911.683777 0.050521	-428.376785 46.967530	5188.892301 84.260511	-0.311333 38.128723	-7.543603 59.235308	-0.476265
	37800.000000 7369.658670	848.979429 0.050017	-7470.029838 46.998624	-1715.703360 84.196338	4.767660 38.670630	1.535800 159.040751	-4.919707
	39600.000000 7363.149684	4951.175703 0.050519	2416.345429 46.973655	-5009.824489 84.089759	-1.478942 38.794587	6.734722 254.236543	2.319065
154	41400.000000 7367.173062	-2640.837189 0.050951	5493.190015 46.987085	3426.840322 84.039098	-4.443600 38.980313	-4.736712 3.106079	4.210004
	43200.000000 7368.681619	-3202.443768 0.050644	-6200.294218 46.993390	2707.674420 83.900779	3.733108 39.395762	-4.300484 110.957241	-4.469569
	45000.000000 7363.527064	4397.209247 0.050794	-3709.762195 46.974280	-5109.154122 83.862159	2.582952 38.757137	6.216736 206.516828	-2.039361
	46800.000000 7371.254364	1877.072707 0.051054	6767.043728 47.001600	-1211.115737 83.750920	-4.894381 38.197745	2.023328 308.365382	5.453880

Datos simulador

	48600.000000 7361.563276	-4922.728163 0.050543	-591.716139 46.967828	5170.727000 83.656599	-0.237007 38.715043	-7.528075 60.376708	-0.638626
	50400.000000 7369.516731	914.059122 0.050027	-7429.518042 46.998082	-1862.743539 83.595043	4.762754 39.227733	1.677633 160.052834	-4.874531
	52200.000000 7363.340960	4935.418438 0.050503	2562.415055 46.974393	-4942.285145 83.486484	-1.547220 39.355863	6.677004 255.317800	2.458513
155	54000.000000 7366.824937	-2707.995991 0.050907	5386.070833 46.985939	3543.383598 83.436654	-4.405119 39.571665	-4.871552 4.297508	4.093814
	55800.000000 7368.891621	-3155.781234 0.050620	-6288.606616 46.994218	2573.626359 83.299117	3.778891 40.005056	-4.168371 111.977004	-4.545353
	57600.000000 7363.353525	4436.960746 0.050809	-3569.255178 46.973607	-5169.671426 83.259021	2.533795 39.364604	6.282378 207.484401	-1.905818
	59400.000000 7371.405471	1809.256869 0.051087	6805.595236 47.002058	-1055.940556 83.149786	-4.930602 38.788612	1.858633 309.513211	5.487804

Datos simulador

61200.000000 7361.679006	-4933.330284 0.050566	-754.487666 46.968187	5147.908801 83.052720	-0.162468 39.304295	-7.508317 61.513215	-0.800062
63000.000000 7369.365514	979.460449 0.050039	-7385.500401 46.997502	-2008.390286 82.993706	4.757632 39.785323	1.818278 161.063845	-4.825575
64800.000000 7363.543508	4919.207507 0.050488	2706.859165 46.975170	-4870.703684 82.883265	-1.615978 39.915005	6.615740 256.403098	2.596266
66600.000000 7366.476097	-2775.178921 0.050864	5275.854352 46.984792	3656.574377 82.834112	-4.365871 40.160173	-5.003183 5.491641	3.973863
68400.000000 7369.092158	-3108.643791 0.050594	-6373.549836 46.995014	2437.393740 82.697519	3.824580 40.612215	-4.034645 112.997057	-4.617235
70200.000000 7363.188182	4476.732752 0.050823	-3427.293294 46.972969	-5226.192064 82.655831	2.484150 39.972574	6.344921 208.452335	-1.770633
72000.000000 7371.539136	1740.832526 0.051119	6840.103075 47.002455	-899.882924 82.548687	-4.966529 39.382547	1.692785 310.659732	5.516862
73800.000000	-4943.475569	-916.575348	5120.463838	-0.087687	-7.484369	-0.960430

Datos simulados

157

7361.811550	0.050589	46.968606	82.448877	39.896482	62.644807		
75600.000000 7369.205477	1045.212668 0.050053	-7338.016353 46.996885	-2152.531137 82.392323	4.752273 40.343539	1.957654 162.073677		-4.772883
77400.000000 7363.756781	4902.512464 0.050474	2849.580367 46.975982	-4795.133859 82.280103	-1.685243 40.472121	6.550966 257.492342		2.732202
79200.000000 7366.127964	-2842.396257 0.050821	5162.632561 46.983645	3766.307727 82.231474	-4.325826 40.745849	-5.131507 6.688415		3.850272
81000.000000 7369.282669	-3060.997750 0.050568	-6455.080835 46.995776	2299.090659 82.095982	3.870183 41.217150	-3.899408 114.017511		-4.685172
82800.000000 7363.031554	4516.525292 0.050835	-3283.970574 46.972367	-5278.669690 82.052593	2.433979 40.580910	6.404328 209.420793		-1.633903
84600.000000 7371.654715	1671.765947 0.051149	6870.545074 47.002790	-743.082877 81.947619	-5.002144 39.979498	1.525911 311.804974		5.541012
86400.000000 7361.960258	-4953.147434 0.050612	-1077.864440 46.969082	5088.422317 81.845077	-0.012639 40.491584	-7.456279 63.771485		-1.119589

Datos simulador

Propagación con atmósfera

Archivo de resultados

Objeto 1, clase sólido

Tiempo (s)	Orbita prevista (Km, grados)	Semieje mayor	Excentricidad	Inclinacion	Longitud nodo asc	Argumento perigeo	Anomalía verdadera
0.000000	6789.999534	0.001122	51.746000	86.254000	37.759000	339.336000	
43200.000000	6789.999487	0.001122	51.746000	86.254000	37.713294	252.302606	
86400.000000	6789.999435	0.001122	51.746000	86.254000	37.713291	165.455652	
129600.000000	6789.999383	0.001122	51.746000	86.254000	37.713301	78.547560	
172800.000000	6789.999331	0.001122	51.746000	86.254000	37.713308	351.400628	
216000.000000	6789.999279	0.001122	51.746000	86.254000	37.713300	264.290208	
259200.000000	6789.999227	0.001122	51.746000	86.254000	37.713295	177.422283	
302400.000000	6789.999175	0.001122	51.746000	86.254000	37.713304	90.543423	
345600.000000	6789.999123	0.001122	51.746000	86.254000	37.713313	3.420916	
432000.000000	6789.999019	0.001122	51.746000	86.254000	37.713299	189.389173	

Apéndice B

Ejemplos de código

En este anexo se presentan algunos extractos del código del programa, para facilitar las consultas y las referencias rápidas.

Se clasifica el código según el archivo en el que esté, de manera que, si se quiere usar una función, se sea capaz de encontrarla con facilidad.

No está todo el código de todas las partes del programa, ni todo el código de las partes presentes. No se muestran aquellas funciones que no se consideran lo suficientemente interesantes, bien por ser partes incompletas o descartadas, bien por ser funciones de prueba que se han usado durante la construcción del programa, o bien por ser muy simples y poco interesantes. Por ello, pueden aparecer algunas llamadas sin su correspondiente función, pero en los archivos originales si que está definida. Para ver la lista completa de las funciones la mejor opción es, como no, los archivos originales.

Las palabras en negrita son palabras clave del lenguaje de Python, que controlan el código. Las partes precedidas por una almohadilla `#` son comentarios que el compilador ignora, puestos para explicar con más o menos claridad el código.

orbita.py

```
class orbita():      #Clase orbita , autocontenido

    def __init__(self):      #Inicializacion de parametros orbitales y constantes
        self.parametros = np.array([6378.,0.,0.,0.,0.,0.])
        self.constantes = np.array([398600.4418,1.,1.])      #Constantes necesarias # [GMR, ,]
        self.propiedades = propiedadesorbita()
        self.pos = np.array([[[]]])

    def setorbita(self,a,e,i,ascension,argumentoper,trueanom):  #Cambio de parametros orbitales
        self.parametros = np.array([a,e,i,ascension,argumentoper,trueanom])

    def par2xv(self):      #Transformacion de parametros a posicion/velocidad
        p = self.parametros[0]*(1-self.parametros[1]**2)
        theta = self.parametros[5]*np.pi/180.
        r = p/(1.+self.parametros[1]*np.cos(theta))

        x= r*np.array([1.,0.,0.])
        z1 = np.array([0.,0.,1.])
        x1 = np.array([1.,0.,0.])
        giro1 = cua.ang2cua(z1,self.parametros[3])
        x1 = cua.giro(x1,z1,self.parametros[3])
        giro2 = cua.ang2cua(x1,self.parametros[2])
        z1 = cua.giro(z1,x1,self.parametros[2])
        giro3 = cua.ang2cua(z1,self.parametros[4])
        giro4 = cua.ang2cua(z1,self.parametros[5])
```

```

giro = cua.pro(giro2, giro1)
giro = cua.pro(giro3, giro)
giro = cua.pro(giro4, giro)

h = np.sqrt(p*self.constantes[0])
v = np.array([self.constantes[0]*np.sin(theta)*self.parametros[1]/h, self.constantes[0]*(1.+self.parametros[1]*np.cos(theta))/h, 0.])

x = cua.conv(x, giro)
v = cua.conv(v, giro)
return x,v

def periodo(self):
    T = T = 2.0*np.pi*np.sqrt(self.parametros[0]**3.0 / self.constantes[0])
    return T

def plano(self):
    z1 = np.array([0.,0.,1.])
    x1 = np.array([1.,0.,0.])
    x1 = cua.giro(x1,z1,self.parametros[3])
    z1 = cua.giro(z1,x1,self.parametros[2])
    return z1

def xv2par(self, pos, vel): #Trasformacion de posicion/velocidad a parametros
    h = cross(pos, vel) #Momento cinetico

    if modulo(h)<1e-6:
        return "Caida libre"

    e = -(pos/modulo(pos)) - (cross(h, vel)/self.constantes[0]) #Excentricidad

    Ener = np.dot(vel, vel)/2.0
    Ener = Ener - (self.constantes[0]/modulo(pos)) #Energia de la orbita

    if Ener >=0.:
        return "Orbita_no_eliptica"

```

```

a = -self.constantes[0]/(2.0*Ener)    #Semieje mayor
T = 2.0*np.pi*np.sqrt(a**3.0/ self.constantes[0])  #Periodo

u3 = h/modulo(h)

if modulo(e) <=1e-6:      #Orbita circular, excentricidad nula
    if modulo(cross(u3,[0 ,0 ,1])) <=1e-6: #Orbita ecuatorial
        u1 = np.array([1. ,0. ,0.])
        anover = np.dot(pos,u1)/modulo(pos)
        if abs(anover)>=1.:
            anover = math.copysign(1.0, anover)
        trueanom = np.arccos(anover)
        self.parametros = np.array([a,0. ,0. ,0. ,0. ,trueanom*180./np.pi])

    return

else:    #Circular general

    hproy = np.array([h[0],h[1],0.])

    n1 = cross([0,0,1],u3)      #Definicion del triedro perifocal
    n1 = n1/modulo(n1)          #u3 direccion h, u1 direccion e(Circular->Nodo ascendente)
                                #u2 perpendicular a ambos
    u1 = n1
    u2 = cross(u3,u1)          #n1 direccion nodo ascendente
    u2 = u2/modulo(u2)          #u2 perpendicular, contenido en el plano ecuatorial

    argumentoper = 0.    #Por orbita circular

```

163

```
ascension = np.arccos(np.dot(n1,[1.,0.,0.])) #Longitud recta del nodo ascendente
sector = np.arcsin(np.dot(n1,[0.,1.,0.]))
if sector <0.:
    ascension = -ascension
if ascension <0.:
    ascension = 2.*np.pi + ascension

i = np.arccos(np.dot([0,0,1],u3)) #Inclinacion

anover = np.dot(pos,u1)/modulo(pos)
if abs(anover)>=1.:
    anover = math.copysign(1.0, anover)
trueanom = np.arccos(anover) #Anomalia verdadera
sector = np.arcsin(np.dot(pos,u2)/modulo(pos))

if sector <0.:
    trueanom = -trueanom
if trueanom<0.:
    trueanom = 2.*np.pi + trueanom

self.parametros = np.array([a,modulo(e),i*180./np.pi,ascension*180./np.pi,argumentoper*180./np.pi,
                           trueanom*180./np.pi])

return
```

orbita.py

```

else:      #Elíptica

    u1 = e/modulo(e)
    u3 = h/modulo(h)
    u2 = cross(u3,u1)
    n1 = cross([0.,0.,1.],u3)
    n1 = n1/modulo(n1)
    i = np.arccos(np.dot([0.,0.,1.],u3))

if i<1e-6:  #Elíptica ecuatorial
    ascension = 0.

    argumentoper = np.arccos(np.dot(u1,n1))
    sector = np.arcsin(np.dot(cross(n1,u1),u3))

    if sector <0.:
        argumentoper = -argumentoper

    if argumentoper <0.:
        argumentoper = 2.*np.pi + argumentoper

    anover = np.dot(pos,u1)/modulo(pos)
    if abs(anover)>=1.:
        anover = math.copysign(1.0, anover)
    trueanom = np.arccos(anover)  #Anomalía verdadera
    sector = np.arcsin(np.dot(pos,u2)/modulo(pos))

    if sector <0.:
        trueanom = -trueanom

```

```
    if trueanom<0.:
        trueanom = 2.*np.pi + trueanom

    self.parametros = np.array([a,modulo(e),i*180./np.pi,ascension*180./np.pi,argumentoper*180./np.pi,
                               trueanom*180./np.pi])

    return

else: #Eliptica general

    ascension = np.arccos(np.dot(n1,[1.,0.,0.])) #Longitud recta del nodo ascendente
    sector = np.arcsin(np.dot(n1,[0.,1.,0.]))
    if sector <0.:
        ascension = -ascension
    if ascension<0.:
        ascension = 2.*np.pi + ascension

    argumentoper = np.arccos(np.dot(u1,n1))
    sector = np.dot(cross(n1,u1),u3)
    if sector <0.:
        argumentoper = -argumentoper
    if argumentoper <0.:
        argumentoper = 2.*np.pi + argumentoper
    anover = np.dot(pos,u1)/modulo(pos)
```

```

if abs(anover)>=1.:
    anover = math.copysign(1.0, anover)
trueanom = np.arccos(anover) #Anomalía verdadera

sector = np.arcsin(np.dot(pos,u2)/modulo(pos))

if sector <0.:
    trueanom = -trueanom

if trueanom<0.:
    trueanom = 2.*np.pi + trueanom

self.parametros = np.array([a,modulo(e),i*180./np.pi,ascension*180./np.pi,argumentoper*180./np.pi,
                           trueanom*180./np.pi])

return

def xenorbita(self,x):

    npuntos = 1000
    Atheta = 400./npuntos
    theta = 0.
    orbitatest = orbita()
    while theta <=360.:
        orbitatest.setorbita(self.parametros[0], self.parametros[1], self.parametros[2], self.parametros[3], self.
                             parametros[4],theta)
        xtest, = orbitatest.par2xv()
        r = (x[0]-xtest[0])**2+(x[1]-xtest[1])**2+(x[2]-xtest[2])**2

        if r<=1.:
            return True,theta
        theta = theta+Atheta

    return False,0.

def orbita(self):

```

```

import cuaternios as cua

npuntos = self.propiedades.resolucion
Atheta = 400./npuntos
p = self.parametros[0]*(1-self.parametros[1]**2)
theta = Atheta

r = p/(1+ self.parametros[1])
x = r * np.array([1.,0.,0.])

z1 = np.array([0.,0.,1.])
x1 = np.array([1.,0.,0.])
giro1 = cua.ang2cua(z1, self.parametros[3])
x1 = cua.giro(x1,z1, self.parametros[3])
giro2 = cua.ang2cua(x1, self.parametros[2])
z1 = cua.giro(z1,x1, self.parametros[2])
giro3 = cua.ang2cua(z1, self.parametros[4])

giro = cua.pro(giro3, giro2)
giro = cua.pro(giro, giro1)

x = cua.conv(x, giro)
self.pos = np.array([x])

while theta <=360.:

    r = p/(1+ self.parametros[1]*np.cos(np.pi*theta/180))
    x = r * np.array([1.,0.,0.])
    z1 = np.array([0.,0.,1.])
    x = cua.giro(x,z1,theta)
    x = cua.conv(x, giro)
    x = np.array([x])

    self.pos = np.append(self.pos,x,0)

```

```
theta = theta+ Atheta

class base():
    def __init__(self):
        self.pos = np.array([[6378.,0,0]])
        self.lat = 0.
        self.lon = 0.
        self.propiedades = propiedadesbase()

class sensor():
    def __init__(self):
        self.objeto = ""
        self.propiedades = propiedadessensor()
        self.pos = np.array([[6978.,0,0]])
        self.t = np.array([[0.]])
        self.vector = np.array([-1.,0.,0.])
        self.semiangulo = 15.
        self.resolucion = 5.
        self.dmax = 10000.
        self.actitud = "Nadir"
    def setnombre(self,nombre):
        self.propiedades.nombre = nombre
    def calcvector(self,x):
        if self.actitud == "Nadir":
            lat,lon = xtogeox(x,0.)
            nadir = geotox(lat,lon)
            vector = np.array([nadir[0]-x[0],nadir[1]-x[1],nadir[2]-x[2]])
            vector = vector/np.sqrt(vector[0]**2+vector[1]**2+vector[2]**2)
            return vector
        elif self.actitud == "Cenit":
            lat,lon = xtogeox(x,0.)
```

```

nadir = geotox(lat,lon)
vector = np.array([nadir[0]-x[0],nadir[1]-x[1],nadir[2]-x[2]])
vector = -vector/np.sqrt(vector[0]**2+vector[1]**2+vector[2]**2)
return vector
elif self.actitud == "Direccion_fija":
    return self.vector

def nadir(self):
    lat,lon = xtogeoo(self.pos[:,0],0.)
    nadir = geotox(lat,lon)
    vector = np.array([nadir[0]-self.pos[0,0],nadir[1]-self.pos[0,1],nadir[2]-self.pos[0,2]])
    vector = vector/np.sqrt(vector[0]**2+vector[1]**2+vector[2]**2)
    self.vector = vector
def cenit(self):
    lat,lon = xtogeoo(self.pos[:,0],0.)
    nadir = geotox(lat,lon)
    vector = np.array([nadir[0]-self.pos[0,0],nadir[1]-self.pos[0,1],nadir[2]-self.pos[0,2]])
    vector = -vector/np.sqrt(vector[0]**2+vector[1]**2+vector[2]**2)
    self.vector = vector
def cono(self):
    lista = np.array([[]])
    vector2 = np.array([-self.pos[0,1],self.pos[0,0],0])
    vector2 = vector2/np.sqrt(vector2[0]**2+vector2[1]**2+vector2[2]**2)
    vector = cua.giro(self.vector,vector2, self.semiangulo)
    punto = self.rayo(vector)
    punto = np.array([punto])
    lista = punto
    for i in range(0,int(360./self.resolucion)):
        vector = cua.giro(vector,self.vector,self.resolucion)
        punto = self.rayo(vector)
        punto = np.array([punto])
        lista= np.append(lista ,punto,0)
    return lista

```

```

def rayo(self ,vector):
    lat = xtogeoo(self .pos[0 ,:],0)
    dist = np.sqrt(self .pos[0,0]**2+ self .pos[0,1]**2+ self .pos[0,2]**2) - radioterra(lat[0])
    punto = self .pos[0]
    while dist< self .dmax:
        punto = self .puntorayo(vector ,dist)
        if interseccióntierra(punto):
            return punto
        dist = dist+1.

    return punto

def puntorayo(self , vector , par):
    par = np.array([par])
    return self .pos[0 ,:] + par*vector

def actualizar(self , objetos):
    if self .objeto == "":
        pass
    else:
        self .pos = np.array([objetos[self .objeto].pos[-1,:]])
        self .t = np.array([objetos[self .objeto].t[-1]])

    if self .actitud == "Nadir":
        self .nadir()
    elif self .actitud == "Cenit":
        self .cenit()

def interseccióntierra(x):
    radio = np.sqrt(x[0]**2+x[1]**2+x[2]**2)
    pos = xtogeoo(x,0.)
    radiot= radioterra(pos[0])

```

```

if radio<=radiot:
    return True
else:
    return False
def interseccionorbitas(orbital , orbita2):
    u1 = orbital.plano()
    u2 = orbita2.plano()
    vector = cross(u1,u2)
    if np.sqrt(vector[0]**2+vector[1]**2+vector [2]**2)< 1e-8: #orbitas coplanarias
        listaang = []
        listaang2 = []

        p1 = orbital.parametros[0]*(1.-orbital.parametros[1]**2)
        p2 = orbita2.parametros[0]*(1.-orbita2.parametros[1]**2)
        apol = orbital.parametros[0]*(1.+orbital.parametros[1])
        apo2 = orbita2.parametros[0]*(1.+orbita2.parametros[1])
        peri1 = orbital.parametros[0]*(1.-orbital.parametros[1])
        peri2 = orbita2.parametros[0]*(1.-orbita2.parametros[1])
        r = max(peri1 ,peri2)
        if apol<peri2 or apo2 <peri1:
            return [] ,[]

        theta = 0.
        Atheta = 360./10000
        theta1 = theta-orbital.parametros[4]
        theta2 = theta-orbita2.parametros[4]
        signo = p1/(1.+orbital.parametros[1]*np.cos(np.pi*theta1/180.))-p2/(1.+orbita2.parametros[1]*np.cos(np.pi*theta2/180.))

        if orbital.parametros[1]<1e-8 and orbita2.parametros[1]<1e-8:
            return [] ,[]

        elif orbita2.parametros[1]<1e-8:
            theta1 = np.arccos(((p1/r)-1.)/orbital.parametros[1])*180./np.pi
            listaang.append(theta1)
            theta2 = theta1+orbital.parametros[4]

```

```

        listaang2.append(theta2)

    elif orbita1.parametros[1]<1e-8:
        theta2 = np.arccos(((p2/r)-1.)/orbita2.parametros[1])*180./np.pi
        thetasum2 = theta2 + orbita2.parametros[4]
        theta1 = thetasum2 - orbital1.parametros[4]
        listaang.append(theta1)
        listaang2.append(theta2)

    while theta<=360.:

        theta1 = theta-orbital1.parametros[4]
        theta2 = theta-orbita2.parametros[4]
        r1 = p1/(1.+orbital1.parametros[1]*np.cos(np.pi*theta1/180.))
        r2 = p2/(1.+orbita2.parametros[1]*np.cos(np.pi*theta2/180.))
        if signo*(r1-r2)<=0.:
            listaang.append(theta1)
            listaang2.append(theta2)
        signo = r1-r2
        theta = theta+Atheta

    return listaang , listaang2
x1 = np.array([1.,0.,0.])
z1 = np.array([0.,0.,1.])
x1 = cua.giro(x1,z1,orbita1.parametros[3])

ang = np.dot(vector,x1)/(modulo(vector)*modulo(x1))
ang = np.arccos(ang)
sector = np.arcsin(np.dot(vector,x1))
ang = ang*180./np.pi
if sector <0.:
    ang = -ang
ang = ang-orbital1.parametros[4]
while ang<0.:
    ang = 360. + ang
orbitatest = orbita()

```

```
orbitatest.setorbita(orbita1.parametros[0], orbita1.parametros[1], orbita1.parametros[2], orbita1.parametros[3],  
                     orbita1.parametros[4], ang)  
posicion1,_ = orbitatest.par2xv()  
en1, ang1 = orbita2.xenorbita(posicion1)  
angcomp = ang+180.  
if ang>360.:  
    angcomp = angcomp-360.  
orbitatest.setorbita(orbita1.parametros[0], orbita1.parametros[1], orbita1.parametros[2], orbita1.parametros[3],  
                     orbita1.parametros[4], angcomp)  
posicion2,_ = orbitatest.par2xv()  
en2, ang2 = orbita2.xenorbita(posicion2)  
if en1 and en2:  
    return [ang,angcomp],[ang1,ang2]  
elif en1:  
    return [ang],[ang1]  
elif en2:  
    return [angcomp],[ang2]  
else:  
    return [],[]  
class maniobra():  
    def __init__(self):  
        self.propiedades = propiedadesmaniobra()  
        self.pos = np.array([0.,0.,0.])  
        self.Av = np.array([0.,0.,0.])  
        self.orbitainicial = ""  
        self.orbitafinal = ""  
        self.posicioninicial = np.array([0.,0.,0.])  
        self.posicionfinal = np.array([0.,0.,0.])  
        self.anoini = 0.  
        self.anofin = 0.  
        self.velocidadad inicial = np.array([0.,0.,0.])  
        self.velocidadad final = np.array([0.,0.,0.])  
  
        self.impulsoesp = ""  
        self.mi = 1.  
        self.consumo = 0.
```

integracion.py

```
class solido(object):

    def __init__(self):      #Inicializacion de estado inicial
        self.pos = np.array([[6378.,0,0]])
        self.vel = np.array([[0,0,0]])
        self.t = np.array([[0.]])
        self.propiedades = propiedades()
        self.int = "AB4"
        self.At = 1.
        self.tmax = 1e4

    def setnombre(self, texto):
        self.propiedades.nombre = texto

    def setpos(self, x0):      #Cambio de posicion inicial      Necesario que x0 sea un array tipo []
        x0 = np.array([x0])
        self.pos = np.append(self.pos, x0, 0)

    def reset(self):
        self.pos, self.vel, self.t=np.array([[6378.,0,0]]),np.array([[0,0,0]]), np.array([[0.]]) 

    def setvel(self,v0):      #Cambio de velocidad inicial
        v0 = np.array([v0])
        self.vel = np.append(self.vel, v0, 0)

    def sett(self,t):
        t0 = np.array([t])
        self.t = np.append(self.t, t0, 0)

    def setinicio(self,x0,v0): #Cambio de posicion y velocidad iniciales
        self.pos=np.array([x0])
        self.vel=np.array([v0])
```

174

integracion.py

```
def setintegrador(self, integrador): #Cambio de metodo de integracion
    self.int = integrador
#####
def integrar(self, progreso):
    if self.int == "RK4":

        At = self.At
        imax = int(np.ceil((self.tmax-self.t[-1,0])/self.At))
        if imax < 100:
            imax = 100
        tiempo = 0.
        i=0

        while not self.parada():

            if i%(imax/100) ==0:
                texto = "~~Calculando...~~" + str(i/(imax/100)) + "%"
                progreso.set(texto,)

            y1 = self.rungekutta4(At,tiempo)

            self.setpos(self.pos[-1,:]+y1[0,:])
            self.setvel(self.vel[-1,:]+y1[1,:])
            self.sett(self.t[-1]+At)
            i=i+1

            texto = "~~Completado"
```

```
progreso.completado()

elif self.int == "AB4":

    At = self.At
    imax = int(np.ceil((self.tmax-self.t[-1,0])/self.At))
    if imax < 100:
        imax = 100
    tiempo = 0.
    i=0

    for i in range(0,4):
        y1 = self.rungekutta4(At,tiempo)

        self.setpos(self.pos[-1,:]+y1[0,:])
        self.setvel(self.vel[-1,:]+y1[1,:])
        self.sett(self.t[-1]+At)
        i=i+1

    while not self.parada():

        if i%(imax/100) ==0:
            texto = "Calculando ... " + str(i/(imax/100)) + "%"
            progreso.set(texto,)

        y1= self.adamsbashforth4(At)

        self.setpos(self.pos[-1,:]+y1[0,:])
        self.setvel(self.vel[-1,:]+y1[1,:])
        self.sett(self.t[-1]+At)
        i=i+1
```

177

```

    texto = "Completado"
    progreso.completado()

elif self.int == "PC4":

    At = self.At
    imax = int(np.ceil((self.tmax-self.t[-1,0])/self.At))
    if imax < 100:
        imax = 100
    tiempo = 0.
    i=0

    for i in range(0,3):
        y1 = self.rungekutta4(At,tiempo)

        self.setpos(self.pos[-1,:]+y1[0,:])
        self.setvel(self.vel[-1,:]+y1[1,:])
        self.sett(self.t[-1]+At)
        i=i+1

    while not self.parada():

        if i%(imax/100) ==0:
            texto = "Calculando... " + str(i/(imax/100)) + "%"
            progreso.set(texto,)

        y1= self.predictorcorrector4(At)

        self.setpos(self.pos[-1,:]+y1[0,:])
        self.setvel(self.vel[-1,:]+y1[1,:])
        self.sett(self.t[-1]+At)
        i=i+1

```

```

    texto = "~~Completado"
    progreso.completado()

#####
def f(self, paso = -1, tiempo = 0., punto=0):      #De la ecuacion  $y' = f(t, y)$  con  $y = [[x], [x']]^T$ 

#Paso es el indice de la self.pos donde se calcula f
#Tiempo es el diferencial con self.t[paso]
#Punto es la diferencia entre el punto donde calcular f y self.pos[paso]
#####
#Matriz  $\begin{bmatrix} x' \\ x'' \end{bmatrix} = \begin{bmatrix} x' \\ x''' \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} * \begin{bmatrix} x \\ x' \end{bmatrix}$ 
#
if np.shape(punto) ==():
    punto = np.array([0,0,0])

```

```

A = 0.
B = 1.
C = 0.          #Fuerzas dependientes de la posicion
D = 0.          #Fuerzas dependientes de la velocidad

x = self.pos[paso,:]+ punto [:]
xp= self.vel[paso,:]

matriz = np.array([A*x+B*xp])
matriz1 = np.array([C*x+D*xp])

matriz = np.append(matriz, matriz1,0)

```

```

#####
# Fuerzas independientes

fuerzas1 = np.array ([[0 ,0 ,0]])
fuerzas = self.sumafuerzas(x,xp)

fuerzas = np.append(fuerzas1, fuerzas,0)

f=matriz+fuerzas
return f

def sumafuerzas(self,x,v):
    if len(x) and len(v) == 3:
        pass

    else:
        return "Error_en_integracion/sumafuerzas, x y v deben tener tamano 3"

    grav = gravedad(x)
    aero = self.resistencia(x,v)

    return grav + aero

def resistencia(self,x,v):
    global escenario
    coefbal = self.propiedades.balistico
    r = np.sqrt(x[0]**2+x[1]**2+x[2]**2)
    lat,lon = xtoge(x,0.)
    h = r - radioterra(lat,lon)
    rho = escenario.atmosfera.densidad(h)
    w = np.array([0.,0.,-7292115e-11]) #rad/s
    vatm = cua.productovectorial(w,x)
    vrel = np.array([v[0]+vatm[0],v[1]+vatm[1],v[2]+vatm[2]])

    res = -0.5*1000.*rho*vrel*np.sqrt( vrel[0]**2+vrel[1]**2+vrel[2]**2)/coefbal
    return res

```

```

def parada(self):
    global escenario
    if escenario.colisiones == "Si":
        if self.choquetierra():
            return True
    if self.paradatiempo():
        return True

def paradatiempo(self):
    if self.t[-1,0] >= self.tmax:
        return True

def choquetierra(self):
    x = self.pos[-1,:]
    radio = np.sqrt(x[0]**2+x[1]**2+x[2]**2)
    lat = np.arcsin(x[2]/radio)
    radiot= radioterra(lat,geodes = 0)
    if radio <= radiot:
        return True

def predictorcorrector4(self ,At):
    error = 1.
    i = 0
    y0 = self.adamsbashforth4(At)
    while i<30:
        y1 = self.adamsmoulton4(At,y0)
        error = np.sqrt((y1[0,0]-y0[0,0])**2+(y1[0,1]-y0[0,1])**2+(y1[0,2]-y0[0,2])**2+(y1[1,0]-y0[1,0])**2+(y1[1,1]-y0[1,1])**2+(y1[1,2]-y0[1,2])**2)/np.sqrt((y1[0,0])**2+(y1[0,1])**2+(y1[0,2])**2+(y1[1,0])**2+(y1[1,1])**2+(y1[1,2])**2)
        y0 = y1
        if error <1e-7:
            return y1
        i = i+1
    return y1

def adamsmoulton4(self ,At ,fpre):

```

```

f0 = self.f(paso = -1)
f1 = self.f(paso = -2)
f2 = self.f(paso = -3)
y1 = At*(9*fpre+19*f0-5*f1+f2)/24.
return y1

def rungekutta4(self ,At,tiempo):
    k1 = self.f(paso = -1, tiempo = tiempo)
    k2 = self.f(paso = -1,tiempo=tiempo+At/2,punto=At*k1[0 ,:]/2)
    k3 = self.f(paso = -1,tiempo=tiempo+At/2,punto=At*k2[0 ,:]/2)
    k4 = self.f(paso = -1,tiempo=tiempo+At,punto=At*k3[0 ,:])
    y1 = At*(k1+2*k2+2*k3+k4)/6.
    return y1

181 def adamsbashforth4(self ,At):
    f0 = self.f(paso=-1)
    f1 = self.f(paso=-2)
    f2 = self.f(paso=-3)
    f3 = self.f(paso=-4)

    y1 = At*(55*f0-59*f1+37*f2-9*f3 )/24.
    return y1

def gravedad(x, overridemodele = 0):  #Posicion en kilometros
    global escenario
    if overridemodele == 0:
        modelo = escenario.modelo.gravedad
    else:
        modelo = overridemodele
    h = np.sqrt(x[0]**2+x[1]**2+x[2]**2)

```

```

fuerza = 398600.4418/(h*h)
u = np.array([-x[0]*fuerza/h,-x[1]*fuerza/h,-x[2]*fuerza/h])
if modelo == "Esferico":
    return u
J2 = 1.082636e-3
R = 6378.1363
c = J2*398600.4418*R*R/2.
pert0 = -3.*c*x[0]*(1.-5.*(x[2]/h)**2)/(h**5)
pert1 = -3.*c*x[1]*(1.-5.*(x[2]/h)**2)/(h**5)
pert2 = -3.*c*x[2]*(3.-5.*(x[2]/h)**2)/(h**5)
upert = np.array([[u[0,0]+pert0,u[0,1]+pert1,u[0,2]+pert2]])
if modelo == "J2":
    return upert
if modelo == "SoloJ2":
    return np.array([[pert0,pert1,pert2]])

def xtogeo(x,t): #Trasnforma coordenadas absolutas en geograficas
    global escenario

    def eclat(lat,x,z,a,e):
        valor = x1*np.sin(lat)-z1*np.cos(lat)
        valor = valor-a*(e*e*np.sin(lat)*np.cos(lat)/np.sqrt(1-(e*np.sin(lat))**2))

        return valor
    def declat(lat,x,z,a,e):
        valor = x*np.cos(lat)-z*np.sin(lat)
        R=np.sqrt(1-(e*np.sin(lat))**2)
        valor = valor-a*e*e*(np.cos(2*lat)/R+e*e*np.sin(2*lat)*np.sin(2*lat)/(4*(R**1.5)))
        return valor

    if isinstance(t,float) or isinstance(t,int):
        pass
    else:

```

```

t = t[0]

uxy = np.array([x[0],x[1],0])
lat = cua.productoescalar(x,uxy)
lat = lat/cua.norma(x)
lat = lat/cua.norma(uxy)
if lat>1.:
    lat = 1.
lat = np.arccos(lat)

if x[2] < 0 :
    lat = -lat
#Primer paso de integracion , latitud geocentrica
lat1= lat*180/np.pi

v = np.array([1,0,0])
lon = cua.productoescalar(uxy,v)
lon = lon/cua.norma(uxy)
lon = np.arccos(lon)
lon = lon*180/np.pi
if x[1] < 0:
    lon = -lon

lon = lon - girotierra(t)
while lon >180.:
    lon = lon-360.
while lon<-180.:
    lon = lon+360.

if escenario.reptierra == "Esferica":
    lat =lat*180/np.pi
    return lat,lon

elif escenario.reptierra == "WGS84":
    #Calculo de la latitud geodesica

```

183

integracion.py

```
error = 1.
a=6378.137
f=298.257223563
f=1./f
e=np.sqrt(f*(2-f))
x1 = np.sqrt(x[0]**2+x[1]**2)
z1 = x[2]
i=0
lat = lat + e*e*np.sin(2.*lat)/2. + e**4*np.sin(2.*lat)*(1.+np.cos(2.*lat))/4.
eps = eclat(lat,x1,z1,a,e)
f1=declat(lat,x1,z1,a,e)
lat1 = lat-eps/f1
eps1= eclat(lat1,x1,z1,a,e)
while abs(error)>=1.e-4:
    if lat==0:
        error = abs(lat1-lat)
    else:
        error = abs(lat1-lat)/abs(lat)
    if error <= 1.e-4:
        lat = lat1
        break
    if eps*eps1<0.:
        lattemp = (lat+lat1)/2.
        while lattemp>90.:
            lattmep = lattemp-180.
        while lattemp<-90.:
            lattemp = lattemp+180.
        epstemp = eclat(lattemp,x1,z1,a,e)
        if epstemp*eps <0:
            lat1 = lattemp
            eps1 = epstemp
        else:
            lat = lattemp
            eps = epstemp
    else:
```

```

lat = lat1
eps = eps1
lat1 = lat-eps/f1
while lat1 >90.:
    lat1 = lat1 -180.
while lat1 <-90.:
    lat1 = lat1 +180.
eps1= eclat(lat1 ,x1,z1 ,a,e)

lat = lat*180/np.pi
while lat >90.:
    lat = lat -180.
while lat <-90.:
    lat = lat +180.
return np.array([lat ,lon])

def geotox(lat ,lon , t=0):#Transforma coordenadas geograficas en absolutas ,sobre la tierra
185
    u=np.array([1.,0.,0.])
    lon = lon + girotierra(t)
    radio , lat = radioterra(lat ,geodes=1,geocen=1)
    u1 = np.array([0.,0.,1.])
    u2 = cuia.giro(u,u1,lon -90.)
    u = cuia.giro(u,u1,lon)
    u = cuia.giro(u,u2,lat)
    u = radio*np.array(u)

    return u

def trayectoriageo(x,t): #Transforma una trayectoria en su traza

    i = 1
    imax = len(x)
    traza = np.array([xtogeo(x[0 ,:],t[0 ,0])])
    while i <imax:
        posgeo = np.array([xtogeo(x[i ,:],t[i ,0])])

```

```

posgeo[0,1] = posgeo[0,1]
while posgeo[0,1] < -180:
    posgeo[0,1] = posgeo[0,1] + 360
    traza = np.append(traza, posgeo, 0)
    i = i+1
i = 1
i0 = 0
j=1
trazas = []
while i<imax:
    if traza[i,1] * traza[i-1,1] < -100:
        trazas.append(traza[i0:i,:])
        i0 = i
        j = j+1
    i = i+1
trazas.append(traza[i0:i,:])
return trazas, j

def girotierra(t):
    global escenario
    tiempoi = escenario.tiempo
    D = juliana(tiempoi) - 2451545.
    D0 = np.floor(D)
    D0 = D0+0.5
    if D0 - D>=0.:
        D0 = D0-1.
    GMST = 18.697374558 + 24.06570982441908*D # Formula simplificada
    while GMST>24.:
        GMST = GMST-24.
    while GMST<0.:
        GMST = GMST+24.

    GMST = GMST*360./24.
    w = 7292115e-11 #rad/s
    ang = w*t
    ang = ang*180/np.pi + GMST

```

```

while ang>360.:
    ang = ang-360.
return ang

def dibujartraza(x,t,color,mapa):
    b , ntrazas = trayectoriageo(x,t)
    i = 0
    while ( i < ntrazas):
        a = b[ i ]
        mapa.plot(a[:,1],a[:,0],color = color) #Si cargas el mapa desde imagen
        i = i+1

def radioterra(lat , geodes=1, geocen=0): #geodes indica si es latitud geodesica. Geocen indica si tiene que devolver
    ademas la goocentrica
    global escenario

    if escenario.reptierra == "Esferica":
        if geocen==1:
            return 6378., lat
        else:
            return 6378.

    lat = lat*np.pi/180.
    a=6378.137
    f=298.257223563
    f=1./f
    e=np.sqrt(f*(2-f))

    if geodes == 0:
        r = a*np.sqrt(1.-e**2)/np.sqrt(1-e*e*np.cos(lat)*np.cos(lat))
        if geocen==1:
            return r , lat*180/np.pi

```

```
    else:
        return r

x0 = a*np.cos(lat)/np.sqrt(1.-(e*np.sin(lat))**2)
z0 = a*np.sin(lat)*(1-e**2)/np.sqrt(1.-(e*np.sin(lat))**2)

radio = np.sqrt(x0**2+z0**2)
if geocen ==1:
    latcen = 180*np.arccos(x0/radio)/np.pi
    if z0<0:
        latcen = -latcen

    return radio ,latcen
else:
    return radio

def dibujarorbita(x,t,color,ax):
    mpl.axis("off")

    ax.mouse_init()
    ax.set_axis_off()
    ax._axis3don = False

    max_range = np.array([abs(x[:,0].max())+abs(-x[:,0].min()), abs(x[:,1].max())+abs(-x[:,1].min()), abs(x[:,2].max())
                         ()+abs(-x[:,2].min())]).max()/2.0
    max_range = max(max_range,8000.)

    ax.set_xlim(- max_range, max_range)
    ax.set_ylim(- max_range, max_range)
    ax.set_zlim(- max_range, max_range)

    ax.plot(xs=x[:,0],ys=x[:,1],zs=x[:,2],color = color)
```

```

i=1
imax = len(x)
j=0
apo = x[0,:]
peri = x[0,:]
asc = np.array([[0,0,0]])
des = np.array([[0,0,0]])
while i<imax:

    if cua.norma(x[i,:])> cua.norma(apo):
        apo = x[i,:]
    if cua.norma(x[i,:])< cua.norma(peri):
        peri = x[i,:]

    if x[i,2]*x[i-1,2] < 0. and x[i,2]<0:
        des[0,:] = x[i,:]
        j = j+1

    if x[i,2]*x[i-1,2] < 0. and x[i,2]>0:
        asc[0,:] = x[i,:]
        j=j+1
        i = i+1
asc = np.append(asc,np.array([[0.,0.,0.]]),0)
des = np.append(des,np.array([[0.,0.,0.]]),0)

ax.scatter(apo[0],apo[1],apo[2],color = color,marker="^")
ax.scatter(peri[0],peri[1],peri[2],color = color,marker="v")

ax.plot(asc[:,0],asc[:,1],asc[:,2],color = color, linestyle = "--")
ax.plot(des[:,0],des[:,1],des[:,2],color = color, linestyle = "-.")
mpl.axis("off")

```

cuaternios.py

```
def ang2cua(v, theta, rad=0):

    if rad==0:
        theta = theta * np.pi/180
    elif rad==1:
        pass
    else:
        return "Error en cuaternios/ang2cua, rad debe ser 0 o 1"

    if len(v) == 3:
        pass
    else:
        return "Error en cuaternios/ang2cua, v debe ser una lista de 3 elementos"
    theta=theta/2

    modv = norma(v)
    q=[0,0,0,0]
    q[0] = np.cos(theta)
    q[1] = np.sin(theta)* v[0]/modv
    q[2] = np.sin(theta)* v[1]/modv
    q[3] = np.sin(theta)* v[2]/modv

    modq = norma(q)

    if modq <=1e-8:
        pass
    else:
        for i in range(0,4):
            q[i] = q[i]/modq

    return q
```

```
def pro(p,q):
    #Producto de dos cuaterniones , producto de Hamilton

    if len(p) and len(q) == 4:
        pass
    else:
        return "Error en cuaternios/pro , deben ser dos cuaternios de 4 elementos"
    u=[0,0,0,0]
    u[0] = p[0]*q[0] - p[1]*q[1] - p[2]*q[2] - p[3]*q[3]
    u[1] = p[0]*q[1] + p[1]*q[0] + p[2]*q[3] - p[3]*q[2]
    u[2] = p[0]*q[2] + p[2]*q[0] + p[3]*q[1] - p[1]*q[3]
    u[3] = p[0]*q[3] + p[3]*q[0] + p[1]*q[2] - p[2]*q[1]

    return u
def conj(q):
    if len(q)==4:
        pass
    else:
        return "Error en cuaternios/conj , debe ser una lista de 4 elementos"

191
r = [0,0,0,0]
r[0], r[1], r[2], r[3] = q[0], -q[1], -q[2], -q[3]
return r

def norma(q):
    if len(q) == 4:
        norma = q[0]**2+q[1]**2+q[2]**2+q[3]**2
    elif len(q) ==3:
        norma = np.sqrt(q[0]**2+q[1]**2+q[2]**2)
    else:
        return "Error en cuaternios/norma , debe ser una lista de 4 elementos"

    return norma
```

```
def conv(v,q):
    p = [0,v[0],v[1],v[2]]

    p1 = pro(p,conj(q))
    p1 = pro(q,p1)

    if len(p1) == 4:
        p1 = [p1[1],p1[2],p1[3]]

    return p1

def giro(u,v,theta): #Giro de u, theta grados alrrededor de v
    q = ang2cua(v,theta)
    u1 = conv(u,q)
    return u1

def girar(u,q):
    v = conv(u,q)
    return [v[1],v[2],v[3]]
```

mision.py

```
class prescenario():
    def __init__(self):
        self.reptierra = "Esferica"
        self.modeloatmosfera = "No"
        self.modelogravedad = "Esferico"
        self.colisiones = "Si"
        self.atmosfera = atm.atmosfera()
        self.actualizar()
        self.tiempo = [2015,6,21,12,00,00]
        self.ultimospuntos = 100000
        self.puntos = 1
    def actualizar(self):
        self.atmosfera.setmodelo(self.modeloatmosfera)
        self.atmosfera.actualizar()

def visibilidad(pos1, pos2,h=0.):
    import numpy as np
    if np.sqrt(pos2[0]**2+pos2[1]**2+pos2[2]**2) < np.sqrt(pos1[0]**2+pos1[1]**2+pos1[2]**2):
        postemp = pos2
        pos2 = pos1
        pos1 = postemp

    vector = np.array([pos2[0]-pos1[0],pos2[1]-pos1[1],pos2[2]-pos1[2]])
    dist = np.sqrt(vector[0]**2+vector[1]**2+vector[2]**2)
    rmax = np.sqrt(pos1[0]**2+pos1[1]**2+pos1[2]**2)
    vector = vector/dist
    dist = min(dist,1.5*np.sqrt(rmax**2+7000.**2))
    a = 0.
    Aa = dist/1000

    vis = True
    rant = rmax
```

```

194

while a<dist:
    x = pos1 + a*vector
    radio = np.sqrt(x[0]**2+x[1]**2+x[2]**2)
    latgeocen = np.arcsin(x[2]/radio)*180./np.pi
    if radio>=7000. and radio>rant:
        break
    elif radio >=7000.:
        pass
    else:
        radiot = integracion.radioterra(latgeocen , geodes=0) + h
        if radio-radiot<=Aa:
            vis = False
            break
        rant = radio
    a = a+Aa

return vis

def juliana(t):
    import math
    #t lista de 6 numeros   año/mes/día/hora/min/seg
    if t[1] == 1 or t[1] == 2:
        ano = t[0]-1
        mes = t[1] + 12
    B = (2-int(ano/100.)+ int((int(ano/100.))/4.))

    JD = int(365.25*(ano+4716)) + int(30.6001*(mes + 1)) + t[2] +(t[3])/24. + t[4]/1440. + t[5]/86400. + B - 1524.5
    if JD <2299160:
        JD = int(365.25*(ano+4716)) + int(30.6001*(mes + 1)) + t[2] +(t[3])/24. + t[4]/1440. + t[5]/86400. - 1524.5
    return JD

def posicionsol(JD):  #Algoritmo de www.psa.es/sdg/sunpos.htm
    dJD = JD - 2451545.
    dOmega = 2.1429 - 0.0010394594*dJD
    dMeanLongitude = 4.8950630 + 0.017202791698*dJD

```

```

dMeanAnomaly = 6.2400600 + 0.0172019699*dJD
dEclipticLongitude = dMeanLongitude + 0.03341607*np.sin(dMeanAnomaly) + 0.00034894*np.sin(2.*dMeanAnomaly) -
0.0001134 - 0.0000203*np.sin(dOmega)
dEclipticObliquity = 0.4090928 - 6.2140e-9*dJD + 0.0000396*np.cos(dOmega)

R = (1.00014 - 0.01671*np.cos(dMeanAnomaly) - 0.00014*np.cos(2.*dMeanAnomaly))* 149597870700.

dY=np.cos(dEclipticObliquity)*np.sin(dEclipticLongitude)
dX=np.cos(dEclipticLongitude)
ascensionrecta= np.arctan(dY/dX)
while ascensionrecta <0.:
    ascensionrecta=ascensionrecta+2*np.pi
declinacion= np.arcsin(np.sin(dEclipticObliquity)*np.sin(dEclipticLongitude))

ascensionrecta=ascensionrecta*180./np.pi
declinacion=declinacion*180./np.pi
u=np.array([R,0.,0.])
u1=np.array([0.,-1.,0.])
u3=np.array([0.,0.,1.])

u=cua.giro(u,u1,declinacion)
u=cua.giro(u,u3,ascensionrecta)

return u

```