

Universidad de Buenos Aires  
Facultad de Ciencias Exactas y Naturales  
Departamento de Computación

## Métodos Numéricos

### Trabajo Práctico N°2

Chocan los Planetas...

Nombre	LU	Mail
Pablo Herrero	332/07	pablodherrero@gmail.com
Thomas Fischer	489/08	tfischer@dc.uba.ar
Kevin Allekotte	490/08	kevinalle@gmail.com

### Abstract

El siguiente trabajo se propone la comparación entre distintos métodos de simulación de sistemas gravitatorios de varios cuerpos usando resolución de sistemas lineales. El objetivo final es calcular las coordenadas para tirar un proyectil desde la órbita del planeta Neptuno para que el mismo impacte en la Tierra.

**Simulación    Gravitación    Sistemas Lineales    Proyectil**

# Índice

<b>Introducción teorica</b>	<b>2</b>
<b>Desarrollo</b>	<b>3</b>
<b>Resultados</b>	<b>6</b>
<b>Discusión</b>	<b>8</b>
Algoritmo de búsqueda del mínimo <code>mindist</code> . . . . .	8
Gauss . . . . .	9
$\Delta t$ . . . . .	9
<b>Conclusiones</b>	<b>9</b>
<b>Apéndices</b>	<b>10</b>
Apéndice A: Enunciado . . . . .	10
Apéndice B: Códigos Fuente . . . . .	15
<b>Referencias</b>	<b>20</b>

# Introducción teórica

En la mecánica clásica o Newtoniana, el problema de los  $n$  cuerpos es el problema de predecir el movimiento de cuerpos celestes en mutua interacción gravitatoria para cualquier instante en el futuro o deducir su movimiento en cualquier instante del pasado.

La acción gravitatoria entre los cuerpos depende de las posiciones relativas entre ellos, por lo que las ecuaciones de interacción gravitatoria quedan definidas en términos de ecuaciones diferenciales.

El problema que surge al plantear el problema con  $3 \leq n$  es que las ecuaciones del sistema quedan definidas en función de demasiadas variables como para aplicar los métodos de integración conocidos hoy en día para resolver las ecuaciones.

Luego tenemos que encontrar otra forma de calcular las variables. En nuestro caso elegimos el método de simulación, el cual introduce errores analíticos por discretizar el tiempo y de redondeo por trabajar con la aritmética finita de la computadora, pero que tiene la ventaja que el sistema a resolver es lineal, para lo cual conocemos un método.

Para minimizar estos errores implementamos distintos métodos de cálculo numérico de aproximación que validamos con sistemas familiares como el sistema solar, del cual sabemos por ejemplo que las órbitas son elípticas y bastante estables. Si dibujamos las órbitas simuladas podemos ver si se comportan de manera similar a lo esperado. Además, tenemos datos que se supone son bastante precisos descargados de la web oficial de la NASA para comparar.

# Desarrollo

En primer lugar el desarrollo consistió en implementar módulos para las operaciones con y entre Matrices y Vectores que fuesen necesarios para resolver sistemas lineales. Esto implicaba también implementar el algoritmo de eliminación Gaussiana con pivoteo parcial para resolver el sistema lineal del segundo método de resolución. Los módulos fueron implementados en el lenguaje C++ para garantizar una buena performance.

Escribimos el módulo `Matriz.h` que implementa una matriz con las operaciones:

```
T Transpuesta
resolver Solución al sistema  $Mx = b$ 
triangular Triangula la matriz con el método de eliminación gausseana con pivoteo simple
Suma, Resta, Multiplicación, Multiplicación por escalar
```

Elegimos implementar la triangulación con el método gausseano de pivoteo simple porque es una de las formas más simples para resolver un sistema de ecuaciones lineal.

Además escribimos la clase `Vector3` que implementa un vector en  $\mathbb{R}^3$  con las operaciones básicas como `Suma`, `Resta` `Producto Escalar`, `Producto Vectorial`, `Normalizar`, etc.

Luego procedimos a implementar un entorno de simulación (C++) y un pequeño script para plotear y poder visualizar los resultados (Python).

Con todo el backend funcionando, procedimos a implementar gradualmente los distintos algoritmos de solución de sistemas lineales, intercalando la producción de cada uno, con fases de validación que consistían en simular los sistemas propuestos en el enunciado, y cuyos resultados se pueden observar en la sección **resultados**.

Para validar los métodos utilizados en la simulación corrimos pruebas que consistían en casos simples como por ejemplo Sol-Tierra-Luna, o el sistema solar y observamos si las trayectorias de los cuerpos se comportaban como esperadas. Por ejemplo comprobamos si la órbita de la tierra alrededor del sol era aproximadamente circular y si recorría una vuelta entera en 365 días. Además vimos si la trayectoria de la luna parecía acompañar a la tierra. Observamos que con un  $dt$  de una hora los planetas se comportaban suficientemente parecido a lo que esperábamos.

Validados los métodos de simulación, tuvimos que idear un algoritmo para calcular el vector de velocidad inicial de nuestro proyectil dirigido a la tierra, de tal manera que le pegase. La idea fue hacer una especie de búsqueda local, ya que suponíamos que de no colisionar con otro planeta en la trayectoria hasta la tierra, si pasabamos con una trayectoria más cerca de la tierra que con otra, podíamos reducir la búsqueda de la solución a un espacio localizado cerca de la primera. Con un espacio o una trayectoria nos referimos a una velocidad inicial, la que en nuestro código está determinada por dos ángulos de

disparo. Uno alrededor el eje  $\hat{z}$ , sobre el plano  $\hat{x}\hat{y}$  ( $\phi$ ) y otro alrededor del eje  $\hat{y}$ , sobre el plano  $\hat{x}\hat{z}$  ( $\theta$ ).

Dadas las condiciones iniciales de un sistema (posiciones y velocidades de todos los cuerpos), podemos simular este sistema con alguno de los métodos y calcular las posiciones para algun tiempo en el futuro. Nuestro objetivo es encontrar la velocidad inicial de nuestro proyectil para que esté a menos de  $10^{-4}$  AU de la tierra en algun instante futuro. Definimos entonces una función (`mindist`) que dadas las condiciones iniciales, un cuerpo `proyectil` y un cuerpo `target`, nos devuelva la distancia mínima a la que se encontrarán estos cuerpos en algun futuro acotado. Nos restringimos a los casos en los que la distancia es monótonamente decreciente, y consideramos sólo los tiempos para los cuales los cuerpos se acercan. (Esto es una desicion de implementación, ya que no estaríamos encontrando las soluciones en las que el proyectil se aleja y luego se vuelve a acercar).

Como posición de lanzamiento elegimos un punto aproximadamente sobre la órbita de neptuno, esto es, a unos 30AU del sol, sobre el plano  $xy$ . Por ejemplo, los lanzamientos de prueba tenían como posición inicial el punto  $(30, 0, 0)$ .

Tenemos entonces que minimizar esta función (o encontrar un valor para el cual sea menor que  $10^{-4}$ ). Para esto elegimos una velocidad inicial a mano (aproximadamente hacia el cuerpo `target`) y corremos un algoritmo que va ajustando la velocidad para minimizar la función `mindist`. [Más detalles en **Discusión**]

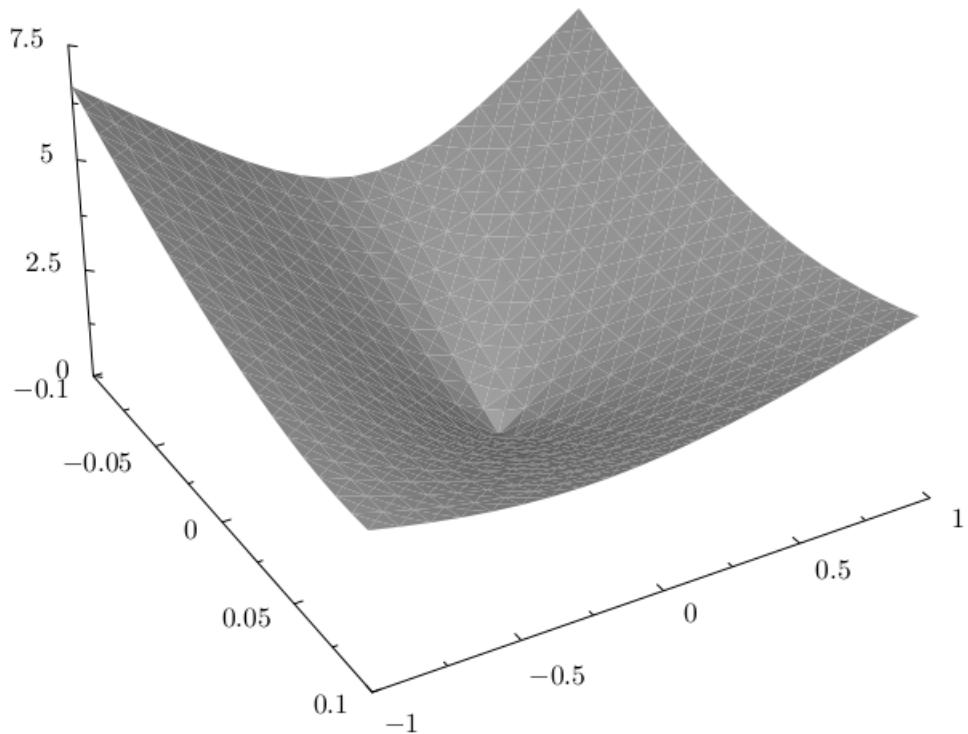


Figura 1: Esta es la forma de la función `mindist` para ciertas condiciones iniciales, donde el dominio es  $\phi \times \theta$  (la velocidad inicial del proyectil) y la imagen es la distancia mínima.

## **Resultados**

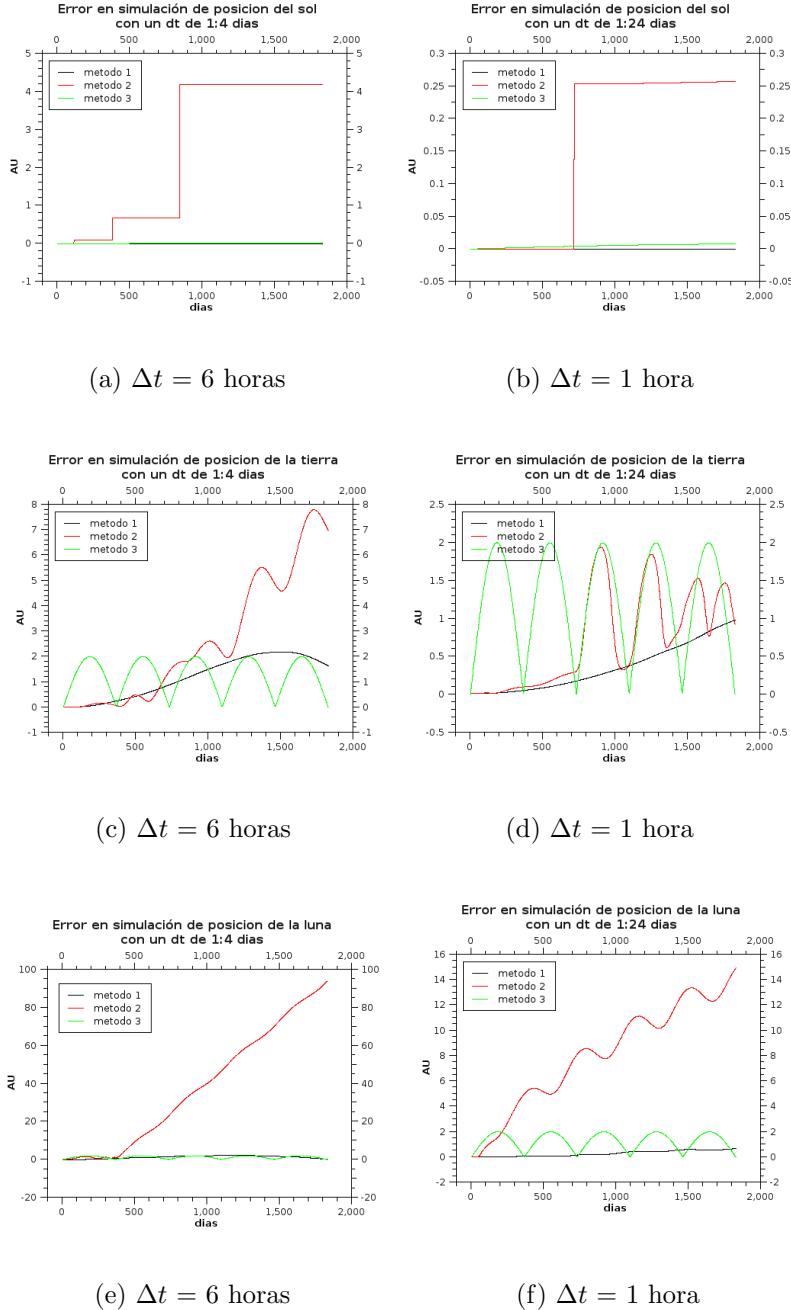


Figura 2: Errores de las posiciones de los planetas con respecto a los datos de la NASA en función del tiempo.

# Discusión

## Algoritmo de búsqueda del mínimo mindist

Para cada trayectoria inicial, definimos una grilla de  $3 \times 3$  puntos, donde el punto central era ella misma, y los otros puntos son las rotaciones de la velocidad sobre los ejes anteriormente mencionados por dos ángulos, respectivamente, de un módulo inicial que se reduce a la mitad en cada iteración de la búsqueda, y luego disparámos proyectiles en todas las 9 direcciones. Así, aunque en algunos casos asintóticamente, podemos llegar a cualquier punto del espacio contenido en la grilla inicial (la definida en la primera iteración) si allí se encuentra una solución. A la grilla inicial le ponemos un ancho inicial grosero, de manera de incluir casi necesariamente a alguna solución.

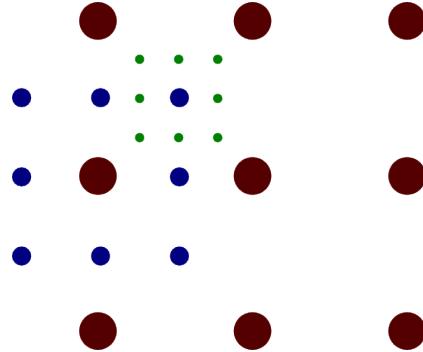


Figura 3: En la imagen vemos una representación gráfica de los sucesivos espacios de búsqueda local. el orden de las iteraciones es rojo → azul → verde.

Para elegir la trayectoria inicial del algoritmo, calculamos primero aproximadamente (simulando y visualizando) cuánto tiempo tardaba el proyectil en llegar hasta la órbita terrestre desde la órbita de Neptuno. Luego calculamos aproximadamente la posición de la Tierra en esa cantidad de tiempo después de la fecha inicial de lanzamiento, tomando los datos de la posición de ese día de la información que provee la NASA.

También de esos datos, elegimos un punto arbitrario en la órbita de Neptuno para el lanzamiento del proyectil. Como dirección de la velocidad inicial del proyectil, elegimos el vector resultante de restar la posición inicial del proyectil a la posición aproximada del impacto sobre la órbita terrestre. El módulo de la velocidad inicial, es el propuesto por el enunciado del problema.

Luego de probar para un par de posiciones y velocidades iniciales distintas, encontramos soluciones donde pudimos destruir la tierra. Las mismas están detalladas en la sección siguiente.

## Gauss

Elegimos resolver los sistemas lineales usando el método de Gauss porque es fácil de implementar y es un poco más eficiente que los métodos de factorización. No se justifica usar LU porque la ventaja principal es que hay que factorizar la matriz una sola vez para resolver muchos sistemas, pero nosotros resolvemos siempre sistemas con diferentes matrices.

$\Delta t$

Un problema que observamos durante las simulaciones, cuando tratábamos de pegarle con el proyectil a la tierra, es que el  $\Delta t$  no era lo suficientemente chico para lograr la presicion de impacto requerida. De un instante de tiempo al otro, el proyectil estaba adelante y luego atrás de la tierra, siendo la distancia entre los cuerpos nunca suficientemente chica. Por esto en la función `mindist` alteramos el  $\Delta t$  dinámicamente a medida que el proyectil se acercaba al `target`, para tener más presicion cuando se encontraban cerca.

## Conclusiones

Llegamos a la conclusión que, dados los métodos implementados, el mejor en la práctica para el ejercicio de destruir la tierra, resultó ser el método 1, ya que, los otros solo resultaban muy buenos (mejores incluso en cuanto a los resultados finales que el método 1) si el  $\Delta t$  era significativamente muy bajo, lo que para nuestro algoritmo de búsqueda local era muy malo en cuanto a tiempo de ejecución.

No obstante, más alla de la sensibilidad del método 2 a los  $\Delta t$  altos, el método de triangulación resultó ser bastante mas rápido de lo que esperabamos, en la práctica.

Un detalle que notamos fue que la precisión relativa entre el método 1 y 2 de la simulación, a medida que el  $\Delta t$  se iba haciendo mas chico, era muy similar. Lo que concluimos fue que a pesar de que los 2 algoritmos son bastante distintos, y el segundo pareciera ser mucho más sofisticado, las aproximaciones que hacemos en ambos casos son siempre lineales, lo que parece justificar nuestras observaciones sobre la similitud de los métodos antes mencionada, y nos dice que a partir de un cierto  $\Delta t$  suficientemente chico (ya que el método 2 es mas sensible con  $\Delta t$  grande) las aproximaciones del segundo método no van a ser mejores que las del primero.

El método 3 parece arreglar este problema, aproximando varias derivadas en un mismo  $\Delta t$ , hasta que la diferencia entre dos derivadas sucesivas se haga mas chica que un umbral. Sin embargo esto es muy parecido, y prácticamente igual de eficiente que achicar todavía mas el  $\Delta t$  para los métodos anteriores.

# Apéndices

## Apéndice A: Enunciado

### Laboratorio de Métodos Numéricos - Primer cuatrimestre 2010 Trabajo Práctico Número 2: Chocan los Planetas...

---

## Introducción

Nos encontramos nuevamente en el C.O.L.L. – CENTRO DE OPERACIONES LOGÍSTICAS LETALES. Atrás ha quedado la dramática XLII Guerra Intergaláctica, a partir de la cual, gracias al valor de héroes anónimos del pasado, el planeta Z-80 pudo gozar de un largo período de tranquilidad. Lamentablemente, nuestros equipos de monitoreo del espacio exterior revelan una nueva fuente de problemas que amenaza con interrumpir la paz.

Desde un lejano planeta de un sistema solar de la vía láctea nos llegan emisiones similares a nuestros prehistóricos sistemas de televisión que de llegar a ser decifrados por las generaciones jóvenes de nuestra población destruirían su capacidad creativa e intelectual. El Honorable Consejo Supremo del Planeta ha decidido cortar de plano con esta amenaza. La orden es terminante: “acabar con la fuente de dichas emisiones”.

Para ello, se nos ha ordenado proporcionar las coordenadas y condiciones de lanzamiento de un proyectil que deberá impactar en el mencionado *Planeta Tierra*, según lo denominan sus habitantes, a fin de enviarles un claro mensaje antes de continuar con hostilidades mayores. Esto se realizará desde una nave ubicada en la órbita del planeta denominado *Neptuno*, lugar más próximo al que podemos acceder sin ser detectados. La supervivencia de nuestro planeta está en sus manos.

## La segunda Ley de Newton y el movimiento de cuerpos en el espacio

El movimiento de un cuerpo viene dado por la conocida ley de Newton (a quién se le ocurrió cuando una manzana impactó sobre su incipiente calva):

$$m \mathbf{a} = \mathbf{F},$$

donde  $m$  es la masa del cuerpo,  $\mathbf{a}$  es la aceleración del cuerpo y  $\mathbf{F}$  es la resultante de la sumatoria de las fuerzas actuantes. La aceleración se relaciona con la velocidad del cuerpo y su posición por medio de las siguientes ecuaciones:  $\mathbf{a} = \frac{d\mathbf{v}}{dt} = \frac{d^2\mathbf{x}}{dt^2}$

En el movimiento de cuerpos en el espacio (estrellas, planetas, lunas, asteroides, satélites, proyectiles, naves, etc.) las principales fuerzas presentes son las de atracción gravitatoria. La fuerza gravitatoria ejercida por un cuerpo  $j$  de masa  $m_j$  que se encuentra en una posición  $\mathbf{x}_j = [x_j \ y_j \ z_j]^T$  sobre otro cuerpo  $i$  de masa  $m_i$  y posición  $\mathbf{x}_i = [x_i \ y_i \ z_i]^T$  viene dada según la siguiente fórmula:

$$\mathbf{F}_{ij} = -G m_i m_j \frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|^3},$$

donde  $G = 6,67428 \times 10^{-11} \text{ m}^3 \text{ kg}^{-1} \text{ s}^{-2}$  es la constante de gravedad universal.

Entonces, dado un conjunto de  $N$  cuerpos interactuando entre sí en el espacio, las ecuaciones que describirán sus movimientos serán

$$m_i \mathbf{a}_i = \sum_{\substack{j=1 \\ j \neq i}}^N \mathbf{F}_{ij}, \quad i = 1..N.$$

Este sistema de ecuaciones diferenciales de segundo orden se puede convertir en un sistema de primer orden escribiéndolo en términos de las velocidades y posiciones de los cuerpos, llegando a

$$\begin{cases} \frac{d\mathbf{v}_i}{dt} = \frac{1}{m_i} \sum_{\substack{j=1 \\ j \neq i}}^N \mathbf{F}_{ij}, & i = 1..N. \\ \frac{d\mathbf{x}_i}{dt} = \mathbf{v}_i, \end{cases} \quad (1)$$

## Métodos para aproximar la solución

El sistema de la ecuación (1) puede pensarse como un sistema de la forma

$$\frac{dy}{dt} = \mathbf{f}(\mathbf{y}), \quad (2)$$

donde  $\mathbf{y}^T = [\mathbf{v}_1^T \ \mathbf{v}_2^T \ \dots \ \mathbf{v}_N^T \ \mathbf{x}_1^T \ \mathbf{x}_2^T \ \dots \ \mathbf{x}_N^T]$ .

Este tipo de sistemas puede resolverse discretizando  $t$  en pasos de tiempo  $t_n = t_0 + n\Delta t$  y buscando calcular  $\mathbf{y}^n = \mathbf{y}(t_n)$  approximando la derivada temporal entre dos pasos de tiempo  $n$  y  $n+1$  en forma discreta como

$$\frac{dy}{dt} \approx \frac{\mathbf{y}^{n+1} - \mathbf{y}^n}{\Delta t}.$$

Reemplazando en (2) obtenemos una relación entre pasos sucesivos:

$$\mathbf{y}^{n+1} = \mathbf{y}^n + \Delta t \mathbf{f}(\mathbf{y}). \quad (3)$$

En este trabajo práctico vamos a estudiar tres técnicas para calcular el paso siguiente según la ecuación (3):

- 1) Utilizar  $\mathbf{f}(\mathbf{y}) = \mathbf{f}(\mathbf{y}^n)$ .
- 2) Utilizar  $\mathbf{f}(\mathbf{y}) = \mathbf{f}(\mathbf{y}^{n+1})$ , aproximándola por medio de una expansión de Taylor como:  $\mathbf{f}(\mathbf{y}^{n+1}) \approx \mathbf{f}(\mathbf{y}^n) + \mathbf{Df}(\mathbf{y}^n)(\mathbf{y}^{n+1} - \mathbf{y}^n)$ , donde  $\mathbf{Df}$  es la matriz que resulta de derivar cada componente del vector  $\mathbf{f}$  respecto de las variables en el vector  $\mathbf{y}$  (ver el apéndice para más detalles). De esta forma resulta la ecuación lineal

$$[\mathbf{I} - \Delta t \mathbf{Df}(\mathbf{y}^n)] \mathbf{y}^{n+1} = \mathbf{y}^n + \Delta t [\mathbf{f}(\mathbf{y}^n) - \mathbf{Df}(\mathbf{y}^n)\mathbf{y}^n].$$

- 3) Realizar una corrección iterativa en cada paso de tiempo del método del ítem 2:

$$\mathbf{w}^0 = \mathbf{y}^n$$

repetir

resolver

$$[\mathbf{I} - \Delta t \mathbf{Df}(\mathbf{w}^k)] \mathbf{w}^{k+1} = \mathbf{y}^n + \Delta t [\mathbf{f}(\mathbf{w}^k) - \mathbf{Df}(\mathbf{w}^k)\mathbf{w}^k].$$

hasta que la diferencia entre  $\mathbf{w}^k$  y  $\mathbf{w}^{k+1}$  sea pequeña

$$\mathbf{y}^{n+1} = \mathbf{w}^{k+1}.$$

## Enunciado

El objetivo de este trabajo práctico es la implementación de los modelos descriptos en la sección anterior para simular la interacción de un conjuntos de  $N$  cuerpos en el espacio. Mediante el modelo que resulte más adecuado se realizará la simulación de la trayectoria del proyectil que debe impactar en la tierra, del cuál deberán indicar las condiciones iniciales para que esto ocurra.

Experimentos obligatorios:

1. Validación de los modelos mediante la simulación de la interacción Sol, Tierra, Luna (Período de tiempo: 30 días, 1 año y 5 años.  $\Delta t = 1$  día, y analizar fracciones y múltiplos).
2. Validación de los modelos mediante la simulación del Sistema Solar (sol, planetas) (Período y  $\Delta t$  idem anterior).
3. Cálculo de las condiciones iniciales para impactar en la tierra y simulación de la trayectoria de los siguientes proyectiles:

- a) *Torpedo de protones*, de masa despreciable y velocidad inicial 256 km/s.
- b) *Bomba Oscura*, de masa  $50 \times 10^{24}$  kg y velocidad inicial 60 km/s.

Consideramos impacto que el proyectil esté a menos de  $10^{-4}$  AU de la posición de la Tierra. Fecha de lanzamiento: 12 de junio de 2010, 11:00hs, UTC-3.

Para realizar estos experimentos utilicen distintas técnicas para resolver sistemas de ecuaciones lineales. Analizar las ventajas y desventajas de cada una de las técnicas propuestas por ustedes para la aproximación de la solución.

## Datos astronómicos

Para obtener los datos astronómicos utilizaremos datos de una agencia espacial de los terrestres. Usando la interfase web (<http://ssd.jpl.nasa.gov/horizons.cgi>),

- cambiar *Ephemeris Type* a **VECTORS (Vector Table)**,
- cambiar *Coordinate Origin* a **Solar System Barycenter** (ingresando @0).
- elegir *Target Body* y *Time Span* a gusto.
- en *Display Output* se puede elegir **download** para guardar a archivo.

De esta forma se obtienen datos de las coordenadas  $\mathbf{x} = [x \ y \ z]^T$ , las velocidades  $\mathbf{v} = [v_x \ v_y \ v_z]^T$  y las masas de los planetas, y mucho más.

---

Fecha de entrega: 21 de mayo de 2010

## Apéndice – Una pequeña ayuda para calcular Df

Dado  $\mathbf{y}^T = [\mathbf{v}_1^T \ \mathbf{v}_2^T \ \dots \ \mathbf{v}_N^T \ \mathbf{x}_1^T \ \mathbf{x}_2^T \ \dots \ \mathbf{x}_N^T]$  la matriz  $\mathbf{Df}$  está formada por 4 submatrices de características particulares

$$\mathbf{Df} = \frac{\partial \mathbf{f}}{\partial \mathbf{y}} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}.$$

Las submatrices  $\mathbf{A}_{11}$  y  $\mathbf{A}_{22}$  son matrices de ceros de  $\mathbb{R}^{3N \times 3N}$  y la submatriz  $\mathbf{A}_{21}$  es la matriz identidad de  $\mathbb{R}^{3N \times 3N}$ . Entonces, la submatriz más compleja de calcular será  $\mathbf{A}_{12}$  que viene dada por

$$\mathbf{A}_{12} = \begin{bmatrix} \frac{1}{m_1} \sum_{j \neq 1} \frac{\partial \mathbf{F}_{1j}}{\partial \mathbf{x}_1} & \frac{1}{m_1} \frac{\partial \mathbf{F}_{12}}{\partial \mathbf{x}_2} & \frac{1}{m_1} \frac{\partial \mathbf{F}_{13}}{\partial \mathbf{x}_3} & \cdots & \frac{1}{m_1} \frac{\partial \mathbf{F}_{1N}}{\partial \mathbf{x}_N} \\ \frac{1}{m_2} \frac{\partial \mathbf{F}_{21}}{\partial \mathbf{x}_1} & \frac{1}{m_2} \sum_{j \neq 2} \frac{\partial \mathbf{F}_{2j}}{\partial \mathbf{x}_2} & \frac{1}{m_2} \frac{\partial \mathbf{F}_{23}}{\partial \mathbf{x}_3} & \cdots & \frac{1}{m_2} \frac{\partial \mathbf{F}_{2N}}{\partial \mathbf{x}_N} \\ \frac{1}{m_3} \frac{\partial \mathbf{F}_{31}}{\partial \mathbf{x}_1} & \frac{1}{m_3} \frac{\partial \mathbf{F}_{32}}{\partial \mathbf{x}_2} & \frac{1}{m_3} \sum_{j \neq 3} \frac{\partial \mathbf{F}_{3j}}{\partial \mathbf{x}_3} & \cdots & \frac{1}{m_3} \frac{\partial \mathbf{F}_{3N}}{\partial \mathbf{x}_N} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{m_N} \frac{\partial \mathbf{F}_{N1}}{\partial \mathbf{x}_1} & \frac{1}{m_N} \frac{\partial \mathbf{F}_{N2}}{\partial \mathbf{x}_2} & \frac{1}{m_N} \frac{\partial \mathbf{F}_{N3}}{\partial \mathbf{x}_3} & \cdots & \frac{1}{m_N} \sum_{j \neq N} \frac{\partial \mathbf{F}_{Nj}}{\partial \mathbf{x}_N} \end{bmatrix}.$$

Las derivadas de las fuerzas  $\mathbf{F}_{ij}$  son las matrices

$$\frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{x}_i} = -G m_i m_j \left( \frac{1}{d^3} \mathbf{I} - \frac{3}{d^5} \mathbf{S} \right) \quad \text{y} \quad \frac{\partial \mathbf{F}_{ij}}{\partial \mathbf{x}_j} = G m_i m_j \left( \frac{1}{d^3} \mathbf{I} - \frac{3}{d^5} \mathbf{S} \right) \quad (4)$$

donde  $\mathbf{I}$  es la matriz identidad de  $\mathbb{R}^{3 \times 3}$  y  $\mathbf{S} = \mathbf{d}\mathbf{d}^T \in \mathbb{R}^{3 \times 3}$  con  $\mathbf{d} = \mathbf{x}_i - \mathbf{x}_j$  y  $d = |\mathbf{d}|$ .

Por ejemplo, en el caso de un problema de 3 cuerpos interactuando, la submatriz  $\mathbf{A}_{12}$  tendrá dimensión  $9 \times 9$  y estará compuesta por 9 matrices de  $3 \times 3$ , de la siguiente forma

$$\mathbf{A}_{12} = \begin{bmatrix} \frac{1}{m_1} \left( \frac{\partial \mathbf{F}_{12}}{\partial \mathbf{x}_1} + \frac{\partial \mathbf{F}_{13}}{\partial \mathbf{x}_1} \right) & \frac{1}{m_1} \frac{\partial \mathbf{F}_{12}}{\partial \mathbf{x}_2} & \frac{1}{m_1} \frac{\partial \mathbf{F}_{13}}{\partial \mathbf{x}_3} \\ \frac{1}{m_2} \frac{\partial \mathbf{F}_{21}}{\partial \mathbf{x}_1} & \frac{1}{m_2} \left( \frac{\partial \mathbf{F}_{21}}{\partial \mathbf{x}_2} + \frac{\partial \mathbf{F}_{23}}{\partial \mathbf{x}_2} \right) & \frac{1}{m_2} \frac{\partial \mathbf{F}_{23}}{\partial \mathbf{x}_3} \\ \frac{1}{m_3} \frac{\partial \mathbf{F}_{31}}{\partial \mathbf{x}_1} & \frac{1}{m_3} \frac{\partial \mathbf{F}_{32}}{\partial \mathbf{x}_2} & \frac{1}{m_3} \left( \frac{\partial \mathbf{F}_{31}}{\partial \mathbf{x}_3} + \frac{\partial \mathbf{F}_{32}}{\partial \mathbf{x}_3} \right) \end{bmatrix}.$$

A continuación se muestran algunas de las matrices de derivadas de fuerzas para ese caso:

$$\frac{\partial \mathbf{F}_{12}}{\partial \mathbf{x}_1} = -G m_1 m_2 \begin{bmatrix} \frac{1}{d^3} - \frac{3}{d^5} (x_1 - x_2)^2 & -\frac{3}{d^5} (x_1 - x_2)(y_1 - y_2) & -\frac{3}{d^5} (x_1 - x_2)(z_1 - z_2) \\ -\frac{3}{d^5} (x_1 - x_2)(y_1 - y_2) & \frac{1}{d^3} - \frac{3}{d^5} (y_1 - y_2)^2 & -\frac{3}{d^5} (y_1 - y_2)(z_1 - z_2) \\ -\frac{3}{d^5} (x_1 - x_2)(z_1 - z_2) & -\frac{3}{d^5} (y_1 - y_2)(z_1 - z_2) & \frac{1}{d^3} - \frac{3}{d^5} (z_1 - z_2)^2 \end{bmatrix}$$

con  $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$ .

$$\frac{\partial \mathbf{F}_{13}}{\partial \mathbf{x}_3} = G m_1 m_3 \begin{bmatrix} \frac{1}{d^3} - \frac{3}{d^5} (x_1 - x_3)^2 & -\frac{3}{d^5} (x_1 - x_3)(y_1 - y_3) & -\frac{3}{d^5} (x_1 - x_3)(z_1 - z_3) \\ -\frac{3}{d^5} (x_1 - x_3)(y_1 - y_3) & \frac{1}{d^3} - \frac{3}{d^5} (y_1 - y_3)^2 & -\frac{3}{d^5} (y_1 - y_3)(z_1 - z_3) \\ -\frac{3}{d^5} (x_1 - x_3)(z_1 - z_3) & -\frac{3}{d^5} (y_1 - y_3)(z_1 - z_3) & \frac{1}{d^3} - \frac{3}{d^5} (z_1 - z_3)^2 \end{bmatrix}$$

con  $d = \sqrt{(x_1 - x_3)^2 + (y_1 - y_3)^2 + (z_1 - z_3)^2}$ .

## Apéndice B: Códigos Fuente

### Algunos defines

```
#define G 0.000148818071108351462554986428198016585385
#define F(i,j,x) (-G*Cuerpos[i].m*Cuerpos[j].m*(y[3*N+3*i+x]-y~~
~~[3*N+3*j+x])/(dist*dist*dist))
#define AC(i,j,x) (-G*Cuerpos[j].m*(y[3*N+3*i+x]-y[3*N+3*j+x])/(~~
~~dist*dist*dist))
#define yX(i) y[3*N+3*(i)]
#define yY(i) y[3*N+3*(i)+1]
#define yZ(i) y[3*N+3*(i)+2]
#define XYZ(i) V3(yX(i),yY(i),yZ(i))
#define VEL(i) V3(y[3*(i)],y[3*(i)+1],y[3*(i)+2])
#define sq(x) ((x)*(x))
```

### Extracto de main, que minimiza mindist

```
int intento=0;
while(min.first>1e-4 && intento<26){
    intento++;
    for(int ii=-1;ii<=1;ii++) for(int jj=-1;jj<=1;jj++){
        Cuerpos[misil_index].v = pdir.rotate(ii*span,jj*span); // ~
~~Calculo direccion
        y=makeY(); // Rehago el vector y
        pair<long double,int> new_min = mindist(y,misil_index,~~
~~target_index);
        if( new_min.first<min.first || ( new_min.first==min.first ~~
~~&& new_min.second<min.second ) ){
            min = new_min;
            mindir = Cuerpos[misil_index].v;
        }
    }

    pdir=mindir;
    span*=.6;
    clog << "mindist:" << min.first << "span:" << span << "~~
~~dir:" << pdir << "it:" << min.second << endl;
}
```

### Función mindist

```
pair<long double,int> mindist(const Vn& y_in, int obj, int ~~
~~target){
```

```

/* Calcula la minima distancia a la que le pasa el proyectil ~
~~a la tierra */
// Devuelve la distancia minima que alcanzan los objetos obj ~
~~y target mientras se esten acercando y mientras no ~
~~supere el tiempo de simulacion
double dtbak=dt;
Vn y(y_in);
long double d=(XYZ(obj)-XYZ(target)).norm();
long double dans;
int i=0;
do{
    dans=d;
    y=next(y);
    d=(XYZ(obj)-XYZ(target)).norm();
    if(VEL(obj).norm()*dt>d*.005){ dt*=.5; /*clog << "dt: " << ~
~~ dt << endl; */ }
}while(d< dans && ++i<resolution);
dt=dtbak;
return pair<long double,int>(dans,i);
}

```

### Funciones que calculan la matriz Df

```

Matriz dFdx(int i, int j, const Vn& y){
    long double d=sqrt( sq(yX(i)-yX(j)) + sq(yY(i)-yY(j)) + sq(yZ~~
~~(i)-yZ(j)) );
    long double d3=1/(d*d*d);
    long double d5=3/(d*d*d*d*d);
    long double dx=yX(i)-yX(j);
    long double dy=yY(i)-yY(j);
    long double dz=yZ(i)-yZ(j);
    Matriz dFdx(3,3,0);
    dFdx(0,0)=d3-d5*dx*dx; dFdx(0,1) = -d5*dx*dy; dFdx(0,2) = -d5~~
~~*dx*dz;
    dFdx(1,0) = -d5*dx*dy; dFdx(1,1)=d3-d5*dy*dy; dFdx(1,2) = -d5~~
~~*dy*dz;
    dFdx(2,0) = -d5*dx*dz; dFdx(2,1) = -d5*dy*dz; dFdx(2,2)=d3-d5~~
~~*dz*dz;
    return -G*Cuerpos[i].m*Cuerpos[j].m*dFdx;
}


```

```

Matriz Df(const Vn& y){
    Matriz res(6*N,6*N,0);
    // lleno A21
    forn(i,3*N) forn(j,3*N) if(i==j) res(3*N+i,j) = 1;

```

```

// lleno A12
forn(i,N) forn(j,N) {
    Matriz S(3,3,0);
    if(i==j){
        forn(k,N) if(k!=i) S += dFdx(i,k,y);
        S *= (1./Cuerpos[i].m);
    }else{
        S = (1./Cuerpos[i].m)*dFdx(i,j,y);
    }
    forn(ii,3) forn(jj,3) res(3*i+ii,3*N+3*j+jj) = (long ~
    ~>double)S(ii,jj);
}
return res;
}

```

Funcion  $f(y)$  ( $\frac{\delta y}{\delta t}$ )

```

Vn f(const Vn& y){
    Vn res(6*N,0);
    forn(i,N){
        long double sumx=0,sumy=0,sumz=0;
        forn(j,N) if(j!=i){
            long double dist=(V3(y[3*N+3*i],y[3*N+3*i+1],y[3*N+3*i~~
                ~~+2])-V3(y[3*N+3*j],y[3*N+3*j+1],y[3*N+3*j+2])).~~
                ~~norm();
            sumx+=AC(i,j,0);
            sumy+=AC(i,j,1);
            sumz+=AC(i,j,2);
        }
        res[3*i] =sumx;
        res[3*i+1]=sumy;
        res[3*i+2]=sumz;
    }
    forn(i,3*N) res[i+3*N]=y[i];
    return res;
}

```

Método 2 (Taylor)

```

Matriz Taylor(const Vn& y){
    Matriz Dfy = Df(y);
    Matriz A( Matriz::ID(6*N) - dt*Dfy );
    return A.resolver( y + dt*( f(y) - Dfy*y ) );
}

```

### Método 3 (Iterativo)

```
Vn MetodoIterativo(const Vn& y, const long double dx){
    Vn w0(y); Vn w1(y);
    long double d0 = INF; long double d1 = INF;
    int i=0;
    do{
        w0 = w1;
        Matriz Dfw = Df(w0);
        Matriz A( Matriz::ID(6*N) - dt*Dfw );
        Matriz B( y + dt*( f(w0) - Dfw*w0 ) );
        w1 = A.resolver( B );

        d0 = d1;
        d1 = dist(w1-w0);
        i++;
    }while( d1<d0 && d1<dx );
    return w0;
}
```

### Resolucion de sistemas lineales (Gauss)

```
Matriz Matriz::resolver(const Vector& b){
    double U2[n*m]; forn(i,n*m) U2[i] = M[i];
    int P2[n]; forn(i,n) P2[i] = i;

    //double res[n]; forn(i,n) res[i] = b.elem(i,0);
    Vector res(b);

    forn(k,n-1) triangular(res,U2,P2,k);

    fornr(i,n){
        forsn(j,i+1,n) res[i] -= U2[i*m + j]*res[j];
        res[i] /= U2[i*m + i];
    }
    return res;
}

void Matriz::triangular(Vector& res, double* U2, int* P2, const ~
~int k){
    int f = k;
    double p = U2[f*m + k];

    // elijo la fila pivote donde el k-esimo elem es maximo
    forsn(i,k,n) if( abs(U2[i*m + k]) > abs(p) ){ f = i; p = U2[f ~
~*m + k]; }
```

```

// swapeo las filas f y k en U y res
forn(j,n){
    double temp_U2 = U2[f*m + j];
    U2[f*m + j] = U2[k*m + j];
    U2[k*m + j] = temp_U2;
    double temp_r = res[f];
    res[f] = res[k];
    res[k] = temp_r;
}

// actualizo el vector de perumtacion
int temp = P2[f];
P2[f] = P2[k];
P2[k] = temp;

// triangulo las filas k+1 -> n
forsn(i,k+1,n){
    // elijo el m_ij
    double a = U2[i*m+k]/p;
    // la k-esima columna es 0
    U2[i*m+k] = 0;
    // calculo el valor de las columnas k+1 -> n
    forsn(j,k+1,n) U2[i*m+j] -= a*U2[k*m+j];
    res[i] -= a*res[k];
}
}

```

## Referencias

1. <http://ssd.jpl.nasa.gov/horizons.cgi>  
Datos de la NASA sobre las posiciones y velocidades de los planetas
2. <http://matplotlib.sourceforge.net/contents.html>  
Referencia de matplotlib, para los gráficos