Mecánica Celeste
Simulación del Sistema Solar

Alejandro García Montoro
Ana Isabel Ruiz Arroyo
Manuel Ruiz Cárdenas
Antonio Solís Izquierdo

17 de diciembre de 2015

1. Introducción

Este documento describe la simulación del Sistema Solar implementada por los autores. En la primera sección se explica el proceso de instalación para usuarios de GNU/Linux; en la segunda, se da una breve descripción de los controles disponibles; en la tercera, se describe con detalle el trabajo matemático y su implementación.

Una versión actualizada de esta documentación, así como del código del programa, puede encontrarse en el repositorio de GitHub kepler_laws[1].

2. Descarga e instalación

2.1. Dependencias

Para la correcta ejecución del programa, el sistema sobre el que se instale necesita tener las siguientes dependencias:

- OpenGL (con freeGLUT)
- wxPython

La mayoría de las distribuciones de Linux tienen estos paquetes en sus repositorios oficiales.

En **Ubuntu**, por ejemplo, la siguiente orden es suficiente para instalar todo lo necesario:

```
1    sudo apt-get install python-opengl freeglut3 python-wxgtk2.8
```

En **Arch Linux** es muy parecido:

[1] `https://github.com/agarciamontoro/kepler_laws`

```
pacaur -S python-opengl freeglut wxpython
```

2.2. Descarga del programa

Si aún no los tienes, usa el siguiente enlace para descargar todos los ficheros del programa —incluida esta documentación— en tu ordenador:

kepler_laws-master.zip[2]

Una vez descargado, descomprímelo y abre una terminal en la carpeta donde se encuentren todos los archivos.

3. Uso

Para comenzar el programa, basta con ejecutar la siguiente orden desde la terminal:

```
python2 main.py
```

3.1. Controles

Se puede rotar la escena haciendo click con el ratón en cualquier lugar de la imagen y moviéndolo. Además, se puede controlar el zoom con la rueda.

Para controlar la velocidad de la animación -que por defecto se comporta de manera que por cada segundo de la vida real transcurra un día en la simulación- se usan las siguientes teclas:

1. **X**: Acelera un paso la animación; es decir, añade un día de la simulación por cada segundo de la vida real.
2. **Z**: Decelera un paso la animación. Se puede usar repetidamente esta tecla para revertir el tiempo.

Para terminar el programa, pulsar la tecla **Q** o cerrar la ventana de la simulación.

La interfaz gráfica que se muestra junto a la ventana de la animación permite:

1. Seleccionar los planetas cuya órbita se desea visualizar.
2. Trasladar la fecha de la animación a un día concreto y mostrar la información relevante de cada planeta marcado.
3. Introducir un ángulo en radianes, seleccionar un planeta, y calcular la fecha en la que su anomalía excéntrica coincide con el ángulo introducido.

[2] `https://github.com/agarciamontoro/kepler_laws/archive/master.zip`

4. Descripción del trabajo

4.1. Unión de trabajos individuales

Este trabajo se ha construido sobre las implementaciones que los integrantes del grupo habían desarrollado individualmente.

Así, al principio ya disponíamos de:

1. Cálculo de las posiciones $x(t)$ de todos los planetas dado un tiempo t, basado en el cálculo de la anomalía excéntrica con el algoritmo de Newton-Raphson.

2. Cálculo de la energía y del momento angular. Estas funciones se han calculado directamente a partir de las definiciones; es decir:

$$E(t) = \frac{1}{2}\|\dot{x}\|^2 + \frac{\mu}{\|x\|}$$
$$c = x \wedge \dot{x}$$

   Así, al resultar constantes ambos valores, se ha observado que la implementación es correcta.

4.2. Mejoras matemáticas

A partir de este trabajo individual, los integrantes del grupo hemos implementado las siguientes mejoras en la parte matemática del programa:

1. Se ha aplicado la rotación real de los planos de las órbitas con respecto al plano de la eclíptica. Esta rotación se ha calculado obteniendo la matriz de giro a partir del ángulo de inclinación, del ángulo con respecto a la línea de nodos y del ángulo con respecto al eje de excentricidad.

2. Se ha añadido el cálculo de la fecha en la que un planeta tiene una anomalía excéntrica dada.

4.2.1. Rotación de los planos orbitales

Para la composición y determinación analítica del movimiento rígido que determinará los planos de las órbitas de los planetas se compondrán 4 endomorfismos de $\mathbb{R}^3$. Dichos endomorfismos dependen obviamente de los datos de cada planeta. El primero es meramente orientativo: llevará los puntos de $\mathbb{R}^3$ del plano XY a los del plano YZ.

$$f : \mathbb{R}^3 \longmapsto \mathbb{R}^3$$
$$f(x) = Afx \ \ \forall x \in \mathbb{R}^3$$

donde

$$Af = \begin{pmatrix} 0 & 0 & -1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

El segundo endomorfismo gira el plano resultante de la imagen del plano XY por f de ω grados respecto del eje X. Donde ω es el ángulo del eje x_1 al eje de excentricidad.

$$G_\omega : \mathbb{R}^3 \longmapsto \mathbb{R}^3$$
$$G_\omega(x) = AG_\omega x \ \ \forall x \in \mathbb{R}^3$$

donde

$$AG_\omega = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\omega) & -\sin(\omega) \\ 0 & \sin(\omega) & \cos(\omega) \end{pmatrix}$$

Ahora se procederá a girar el plano $G_\omega(f(XY))$ en torno al eje Y de tal forma que el vector normal positivo a XY y el resultante de la imagen de $G_\omega(f(XY))$ por la siguiente aplicación formen un ángulo i.

$$G_i : \mathbb{R}^3 \longmapsto \mathbb{R}^3$$
$$G_i(x) = AG_i x \ \ \forall x \in \mathbb{R}^3$$

donde

$$AG_i = \begin{pmatrix} \sin(i) & 0 & -\cos(i) \\ 0 & 1 & 0 \\ \cos(i) & 0 & \sin(i) \end{pmatrix}$$

Ahora sólo queda girar el plano $G_i(G_\omega(f(XY)))$ un ángulo Ω respecto del eje Z para que la nueva línea de nodos forme un ángulo Ω con el eje Y.

$$G_\Omega : \mathbb{R}^3 \longmapsto \mathbb{R}^3$$
$$G_\Omega(x) = AG_\Omega x \ \ \forall x \in \mathbb{R}^3$$

donde

$$AG_\Omega = \begin{pmatrix} \cos(\Omega) & -\sin(\Omega) & 0 \\ \sin(\Omega) & \cos(\Omega) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

La aplicación $M = G_\Omega \circ G_i \circ G_\omega \circ f$ es el movimiento rígido buscado, y su matriz de representación como endomorfismo es el producto siguiente: $AM = AG_\Omega AG_i AG_\omega Af$. Haciendo las operaciones nos queda que AM es la siguiente matriz:

$$\begin{pmatrix} cos(\Omega)\cos(i)\cos(\omega) + \sin(\Omega)\sin(\omega) & -\cos(i)\sin(\omega)\cos(\Omega) - \sin(\Omega)\cos(\omega) & \cos(\Omega)\sin(i) \\ -\cos(i)\cos(\omega)\sin(\Omega) - \cos(\Omega)\sin(\omega) & \sin(\Omega)\cos(i)\sin(\omega) + \cos(\Omega)\cos(\omega) & \sin(\Omega)\sin(i) \\ \sin(i)\cos(\omega) & \sin(i)\sin(\omega) & \cos(i) \end{pmatrix}$$

Para implementar este movimiento rígido, basta aplicar la matriz obtenida a las coordenadas que hasta ahora teníamos.

En la figura 1 se puede observar el resultado de la implementación: se ve claramente cómo las órbitas no son ya coplanarias como lo eran antes, ni tienen el eje de excentricidad apoyado en el eje de coordenadas X.

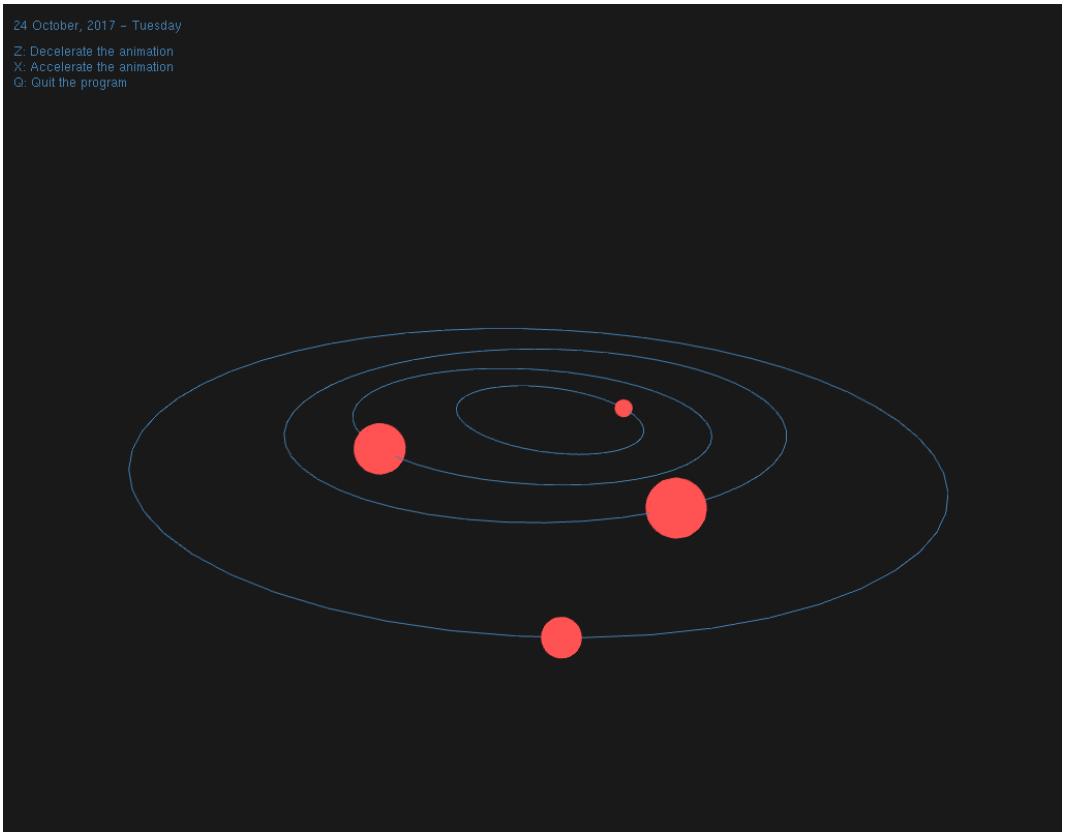

Figura 1: Rotación de los planos orbitales con respecto al plano de la eclíptica.

El código que implementa el movimiento rígido calculado es el siguiente:

```
1  def getGUICoords(self, pos):
2      """Coordinates translation and plane rotation.
3
4      Translates 2D coordinates in a XY plane to 3D coordinates in a
5      XYZ OpenGL space -i.e., X horizontal axis; Y vertical axis; Z
           'depth'
6      axis, with Z decreasing from the monitor towards the user-.
7      Furthermore, it rotates the orbit plane with the matrix
           rotation
8      of the tilt angle, the node line angle and the eccentricity
           angle.
9
```

```
        Args:
            pos: 2D coordinates in a [x,y] form

        Returns:
            The pos coordinates translated to the 3D OpenGL world,
                after have
            applied the rotation of the orbit planes with respect to
                the
            ecliptic plane.
        """

        i = self.tilt_angle
        big_o = self.node_angle
        small_o = self.ecc_angle

        cos = math.cos
        sin = math.sin

        # Rotation matrix
        m_11 = -cos(big_o)*cos(small_o)*cos(i) +
            sin(big_o)*sin(small_o)
        m_12 = -cos(i)*sin(small_o)*cos(big_o) -
            sin(big_o)*cos(small_o)
        m_13 = cos(big_o)*sin(i)

        m_21 = -cos(i)*cos(small_o)*sin(big_o) -
            cos(big_o)*sin(small_o)
        m_22 = -sin(big_o)*cos(i)*sin(small_o) +
            cos(big_o)*cos(small_o)
        m_23 = sin(big_o)*sin(i)

        m_31 = sin(i)*cos(small_o)
        m_32 = sin(i)*sin(small_o)
        m_33 = cos(i)

        matrix = [
            [m_11, m_12, m_13],
            [m_21, m_22, m_23],
            [m_31, m_32, m_33]
        ]

        GUI_coords = []

        pos.append(0.0)

        # Matrix product (apply rotation)
        for row, coord in zip(matrix, pos):
            GUI_coords.append(scalarProduct(row, pos))

```

```
53      # XYZ coordinates are translated into OpenGL as XZ-Y
            coordinates
54      return [GUI_coords[0], GUI_coords[2], -GUI_coords[1]]
```

4.2.2. Cálculo de la fecha dada la anomalía excéntrica

Para calcular el día en el que un planeta dado tiene una anomalía excéntrica u, basta usar la siguiente función, deducida en clase:

$$t(u) = t_0 + \frac{a^{3/2}}{\sqrt{\mu}}(u - \varepsilon sen(u)) \tag{1}$$

Usado la igualdad

$$\sqrt{\mu} = 2\pi \frac{a^{3/2}}{p}$$

con a el semi-eje mayor y p, el periodo del planeta, podemos convertir la ecuación 1 en la siguiente, que ya no contiene la constante μ:

$$t(u) = t_0 + \frac{p}{2\pi}(u - \varepsilon sen(u))$$

El código que implementa esta función es el siguiente:

```
1   def getDate(self, u):
2       """Retrieves the date from the eccentric anomaly
3
4       Obtains the date in which the eccentric anomaly of the planet
            is the
5       given one.
6
7       Args:
8           u: Eccentric anomaly, in radians.
9
10      Returns:
11          The (first after self.t0) date -codified as a datetime
                object- in
12          which the planet had the given u.
13      """
14      p = self.period
15      e = self.eccentricity
16
17      delta = p * (u - e*math.sin(u)) / (2*math.pi)
18
19      return self.t0 + timedelta(days=delta)
```

4.3. Mejoras gráficas

Además del trabajo matemático, se ha mejorado notablemente la interacción gráfica con el programa. Se ha implementado una interfaz que permite al usuario acceder a la información relevante de todos los planetas en fechas arbitrarias mientras se simula el movimiento gráficamente. La figura 2 muestra la interfaz diseñada.

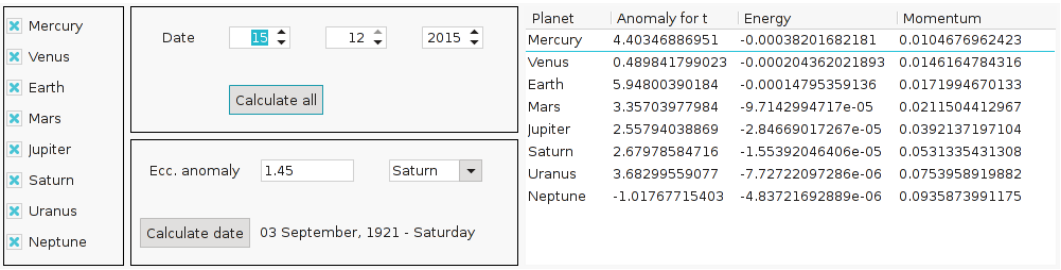

Figura 2: Captura de pantalla de la interfaz gráfica implementada.