FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Análisis Numérico I
[75.12/95.04]
Curso 3
Trabajo Práctico 2
Problema de los Tres Cuerpos Restringido

Grupo 5
Primer cuatrimestre de 2019

Integrantes del grupo		
Santa María Tomás	92797	tomasisantamaria@gmail.com
Hemmingsen Lucas	76187	lhemmingsen@fi.uba.ar
Huenul Matías	102135	matias.huenul.07@gmail.com

Índice

1. Introducción

El objetivo del presente trabajo práctico es utilizar los distintos metodos numéricos de problema de valor inicial para ecuaciones diferenciales ordinarias, para resolver el "Problema de los Tres Cuerpos Restringido o de Euler" y realizar una comparación de los resultados con cada método.

Los métodos que utilizaremos son el método de Euler, Runge-Kutta de orden 2 y 4, Nyström y Newmark.

2. Conceptos teóricos

2.1. Método de Euler

Sea $y'(t) = f(t, y)$ una ecuación diferencial ordinaria de primer orden con condición inicial $y(t_0) = y_0$. Se quiere obtener la solución en $N + 1$ puntos en un intervalo dado, uniformemente espaciado con paso h. Se puede aproximar y(t) por su Polinomio de Taylor de primer orden:

$$y(t_k + h) \cong y(t_k) + hy'(t_k) \tag{1}$$

Pero como $y'(t_k) = f(t_k, y(t_k))$ y además $t_{k+1} = t_k + h$ para todo $0 \leq k \leq N$, esta expresión se puede reescribir como:

$$y(t_{k+1}) \cong t(t_k) + hf(t_k, y(t_k)) \tag{2}$$

Este método puede usarse para resolver sistemas de ecuaciones diferenciales, aplicando lo anterior a cada componente del sistema.
El error cometido con este método se debe en parte al error de truncamiento local (dado por el Teorema de Taylor) en cada paso k y al error cometido por utilizar como condición inicial en dicho paso y_{k-1}, obtenido en el anterior. Se puede demostrar que si el problema está bien planteado, el error total es $e = O(h)$.

2.2. Método de Runge-Kutta

Los métodos de Runge-Kutta son una familia de métodos para resolver numéricamente ecuaciones diferenciales ordinarias de orden n, de la forma $y'(t) = f(t, y)$, con condición inicial $y(t_0) = y_0$. En particular para este Trabajo Práctico se desea hallar soluciones utilizando los métodos de orden 2 y orden 4, cuyas expresiones se mencionan a continuación.

2.2.1. Euler Modificado

Dado el Problema de Valor Inicial, se puede expresar la solución de la ecuación diferencial ordinaria de orden 2 de la siguiente forma:

$$y_{n+1} = y_n + \frac{h}{2}(k_1 + k_2) \tag{3}$$

donde k_1, k_2 son:

$$\begin{aligned} k_1 &= f(t_i, y_i) \\ k_2 &= f(t_i + h, y_i + hk_1) \end{aligned}$$

2.2.2. Runge-Kutta de orden 4

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4) \tag{4}$$

donde k_1, k_2, k_3, k_4 son:

$$
\begin{aligned}
k_1 &= f(t_i, y_i) \\
k_2 &= f(t_i + \frac{h}{2}, y_i + h\frac{k_1}{2}) \\
k_3 &= f(t_i + \frac{h}{2}, y_i + h\frac{k_2}{2}) \\
k_4 &= f(t_i + h, y_i + hk_3)
\end{aligned}
$$

El error global cometido por estos métodos es $e = O(h^n)$.

2.3. Método de Nyström

El método de Nyström es un método de paso múltiple, que aplica de la siguiente manera, dada una EDO de primer orden:

$$
\begin{aligned}
y'(t) &= f(t, y) \\
y(t_0) &= y_0
\end{aligned}
$$

Las ecuaciones adoptan la siguiente forma:

$$y_{i+1} = y_{i-1} + 2hy'(t_i), i = 1, 2, ..., n-1$$

Este valor es refinado localmente:

$$
\begin{aligned}
y_{n+1}^{k+1} &= 2y_n - y_{n-1} + h^2(y'(t) - y_n) \\
y_{n+1}^{k+1} &= 2y_n - y_{n-1} + h^2(\frac{y_{n+1}^k - y_{n-1}}{2h} - y_n) \\
|y_{n+1}^{k+1} - y_{n+1}^k| &< \epsilon
\end{aligned}
$$

Donde ϵ debe adoptar un orden mayor al error de truncamiento local propio al método ($O(h^2)$).

Se observa que se debe obtener un valor de arranque, y_1, para poder comenzar a resolver el problema. En el presente trabajo práctico se utilizará el método de Euler Modificado para aproximar el valor necesario.

2.4. Método de Newmark

Este método se obtiene a partir de la serie de Taylor y es usado para resolver EDOs de segundo orden. Sea $y'' = f(t, y, y')$, entonces las ecuaciones son:

$$
\begin{cases}
y(t_{k+1}) = y(t_k) + hy'(t_k) + \frac{h^2}{2}[\alpha f(t_k; y(t_k); y'(t_k)) + (1-\alpha)f(t_{k+1}, y(t_{k+1}); y'(t_{k+1}))] \\
y'(t_{k+1}) = y'(t_k) + h[\beta f(t_k; y(t_k); y'(t_k) + (1-\beta)f(t_{k+1}, ; y(t_{k+1}); y'(t_{k+1}))]
\end{cases} \tag{5}
$$

2.5. Sistema de ecuaciones diferenciales de primer orden

[?] Sea un sistema de ecuaciones diferenciales de primer orden con la forma:

$$
\begin{cases}
\frac{du_1}{dt} = f_1(t, u_1, u_2, ..., u_m) \\
\frac{du_2}{dt} = f_2(t, u_1, u_2, ..., u_m) \\
\vdots \\
\frac{du_m}{dt} = f_m(t, u_1, u_2, ..., u_m)
\end{cases} \tag{6}
$$

para $a \leq t \leq b$, con condiciones iniciales:

$$u_1(a) = \alpha_1, u_2(a) = \alpha_2, \ldots, u_m(a) = \alpha_m \tag{7}$$

El objetivo es encontrar m funciones $u_1(t), u_2(t), \ldots, u_m(t)$ que satisfacen cada una de las ecuaciones diferenciales junto con las condiciones iniciales. Los métodos para resolver sistemas de ecuaciones diferenciales de primer orden son simplemente generalizaciones de los métodos para una sola ecuación diferencial presentados anteriormente.

2.6. Ecuaciones diferenciales de orden mayor a 1

[?] Una ecuación diferencial de grado m

$$y^{(m)}(t) = f(t, y, y', \ldots, y^{(m-1)}) \tag{8}$$

con $a \leq t \leq b$ y condiciones iniciales $y(a) = \alpha_1, y'(a) = \alpha_2, \ldots, y^{(m-1)(a)=\alpha_m}$ puede convertirse a un sistema de ecuaciones de la forma de (??) y (??).

Sea $u_1(t) = y(t), u_2(t) = y'(t), \ldots, u_m(t) = y^{(m-1)}(t)$. Esto produce el sistema de ecuaciones diferenciales de primer orden

$$\frac{du_1}{dt} = \frac{dy}{dt} = u_2, \frac{du_2}{dt} = \frac{dy'}{dt} = u_3, \ldots, \frac{du_{m-1}}{dt} = \frac{dy^{(m-2)}}{dt} = u_m \tag{9}$$

y

$$\frac{du_m}{dt} = \frac{dy^{(m-1)}}{dt} = y^{(m)} = f(t, y, y', \ldots, y^{(m-1)}) = f(t, u_1, u_2, \ldots, u_m) \tag{10}$$

con condiciones iniciales

$$u_1(a) = y(a) = \alpha_1, u_2(a) = y'(a) = \alpha_2, \ldots, u_m(a) = y^{(m-1)}(a) = \alpha_m \tag{11}$$

3. Desarrollo

3.1. Parte A

Tenemos el siguiente sistema de ecuaciones diferenciales de segundo orden que representan el movimiento de un satélite viajando entre la tierra y la luna e influenciado gravitatoriamente solo por estos dos cuerpos:

$$\begin{cases} x_1'' = 2x_2' + x_1 - \eta\frac{x_1+\mu}{d_1^3} - \mu\frac{x_1-\eta}{d_2^3} \\ x_2'' = -2x_1' + x_2 - \eta\frac{x_2}{d_1^3} - \mu\frac{x_2}{d_2^3} \end{cases} \tag{12}$$

Siendo $d_1 = \sqrt{(x_1+\mu)^2 + x_2^2}$ y $d_2 = \sqrt{(x_1-\eta)^2 + x_2^2}$.

Sea:

$$\begin{cases} v_1(t) = x_1'(t) \\ v_2(t) = x_2'(t) \end{cases} \tag{13}$$

Entonces podemos transformar el sistema de ecuaciones anterior en un sistema de cuatro ecuaciones diferenciales de primer orden:

$$\begin{cases} x_1'(t) = v_1(t) \\ v_1'(t) = 2v_2(t) + x_1(t) - \eta\frac{x_1(t)+\mu}{d_1(t)^3} - \mu\frac{x_1(t)-\eta}{d_2(t)^3} \\ x_2'(t) = v_2(t) \\ v_2'(t) = -2v_1(t) + x_2(t) - \eta\frac{x_2(t)}{d_1(t)^3} - \mu\frac{x_2(t)}{d_2(t)^3} \end{cases} \tag{14}$$

Con valores iniciales en $t = t_0$:

$$\begin{cases} x_1(t_0) = x_{1_0} \\ v_1(t_0) = v_{1_0} \\ x_2(t_0) = x_{2_0} \\ v_2(t_0) = v_{2_0} \end{cases} \tag{15}$$

3.2. Parte B

Ahora resolvamos el problema numéricamente con la función *lsode* de *Octave*. Primero creamos la función *yprima*, que representa a la función *f(t,y)*. A continuación se detalla el código de la misma:

```
function [f]=yprima(y, t)
    mu = 1 / 81.3;
    eta = 1 - mu;

    d1 = sqrt((y(1) + mu)^2 + y(3)^2);
    d2 = sqrt((y(1) - eta)^2 + y(3)^2);

    f(1)= y(2);
    f(2)= 2*y(4) + y(1) - eta*((y(1) + mu)/d1^3) -  mu*((y(1)-eta)/d2^3);
    f(3)= y(4);
    f(4)= -2*y(2) + y(3) - eta*(y(3)/d1^3) -  mu*(y(3)/d2^3);

end
```

Luego, desde la consola de *Octave* ejecutamos la función *lsode* con la función *yprima* como entrada, con una posición inicial $(x_1, x_2) = (1{,}2, 0)$ y velocidad inicial $(v_1, v_2) = (0, -0{,}8)$ en el intervalo $[t_0, t_1) = [0, 2]$ con un $h = 0{,}01$. Para ello, ejecutamos:

```
        [y]=lsode('yprima',[1.2  0  0  -0.8],0:0.01:2)

```

con lo que en la última iteración (en t=2) obtenemos una posición final $(x_1, x_2) = (-0{,}51306, 0{,}07881)$ y una velocidad final $(v_1, v_2) = (-1{,}18383, -0{,}48564)$.

A continuación se muestra un gráfico de la trayectoria del satélite, que se mueve desde el extremo derecho del gráfico hacia la izquierda.

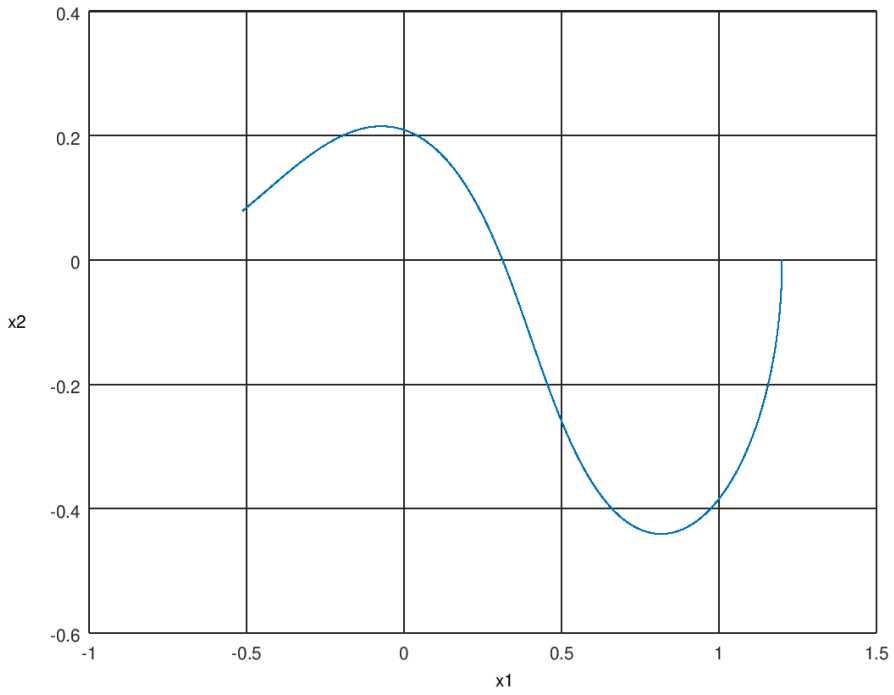

Figura 1: Trayectoria del satélite obtenida con *lsode*.

3.3. Parte C

Veremos ahora como se modifica la trayectoria aumentando el paso h. Para ello implementamos siguiente función.

```
function lsode_con_distintos_h(yprima, a, b, h, y0)
  % Grafica las trayectorias obtenidas con lsode para distintos valores de h.
  % Recibe:
  % -> yprima = f(y, t)
  % -> a, b extremos del intervalo
  % -> h vector con distintos valores de paso
  % -> y0 vector de condiciones iniciales

  for h_i = h
    Y = lsode(yprima, y0, a:h_i:b);
    x1 = Y(:, 1);
    x2 = Y(:, 3);
    plot(x1, x2, sprintf(";h = %d;", h_i));
    hold on
  end
end
```

En el siguiente gráfico se puede observar lo obtenido para cuatro valores de h en particular, utilizando como condiciones iniciales las mismas que al comienzo de la parte b.

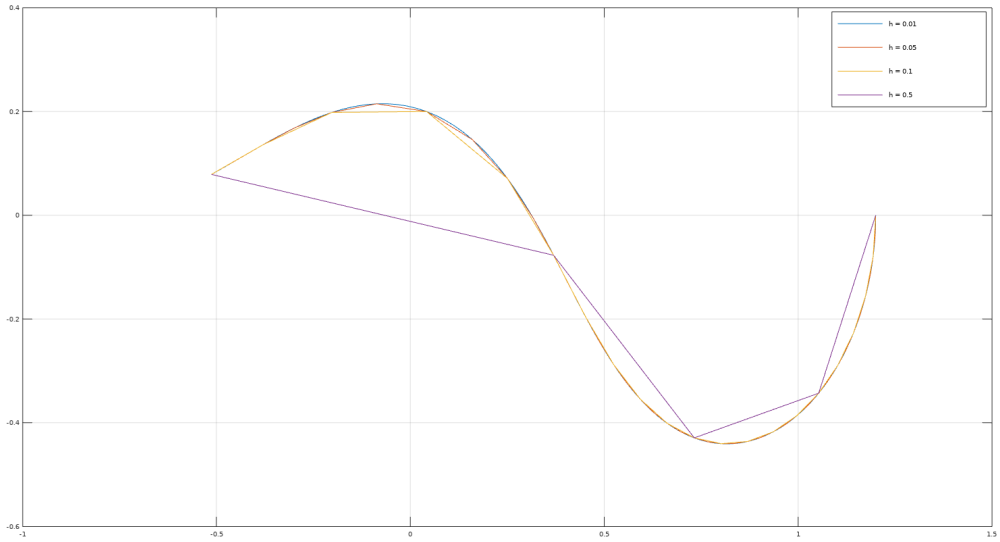

Figura 2: Trayectoria obtenida con distintos valores de h.

3.4. Parte D

Resolveremos ahora el sistema en el intervalo $[0;2]$ mediante el método de Euler. Las ecuaciones son:

$$\begin{cases} x_1(t_{k+1}) = x_1(t_k) + hv_1(t_k) \\ v_1(t_{k+1}) = x_2(t_k) + h(2v_2(t_k) + x_1(t_k) - \eta\frac{x_1(t_k)-\mu}{d_1(t_k)^3}) \\ x_2(t_{k+1}) = x_2(t_k) + hv_2(t_k) \\ v_2(t_{k+1}) = v_2(t_k) + h(-2v_1(t_k) + x_2(t_k) - \eta\frac{x_2(t_k)}{d_1(t_k)^3}) - \mu\frac{x_2(t_k)}{d_2(t_k)^3}) \end{cases} \tag{16}$$

El código de la función implementada en *Octave* para la resolución con este método se muestra a continuación.

```
function [Y] = euler(f, a, b, h, y0)
  % Recibe:
  % -> f = yprima
  % -> a, b extremos del intervalo
  % -> h paso
  % -> y0 vector de condiciones iniciales
  % Devuelve:
  % -> Y matriz donde cada fila es el correspondiente vector yk = y(tk)

  t = a:h:b;
  k = (b - a) / h;

  Y = zeros(k, length(y0));
  Y(1, :) = y0;
  for i = 2 : k + 1

    y_n = Y(i-1, :);
    yprima_n = feval(f, Y(i-1, :));

    Y(i, :) = y_n + h * yprima_n;

  end
end
```

Primero utilizamos como condición inicial el vector $y_0 = (1,2;0;0;-0,8)$ y como tolerancia el valor $h = 0{,}01$.

Los resultados para cada t del intervalo se encuentran en el archivo *resultados-parte-d.txt*.
En particular, el resultado obtenido para $t = 2$ es:

$$y(2) = (-0{,}54958; -1{,}49872; 0{,}21286; -0{,}03654) \tag{17}$$

La trayectoria obtenida en este caso es la siguiente.

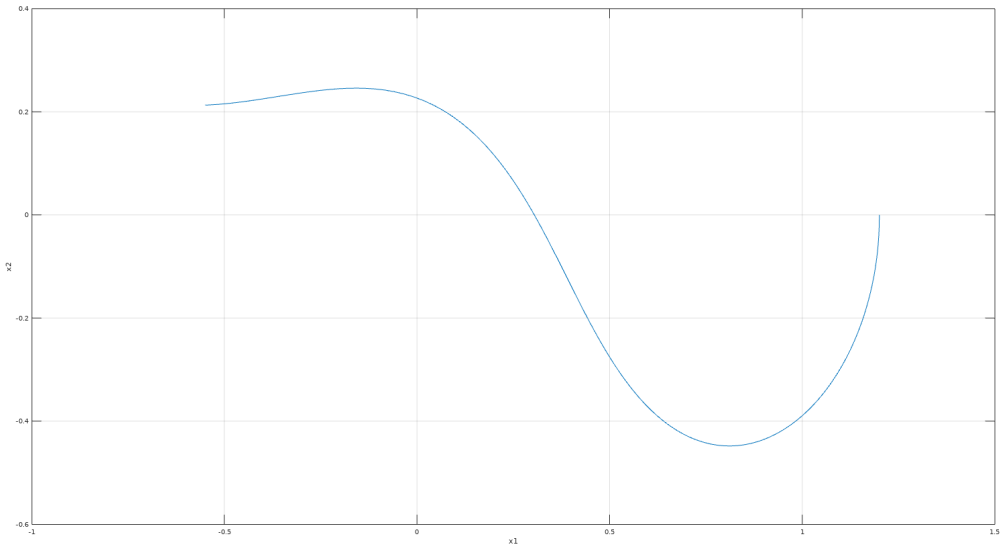

Figura 3: Trayectoria del satélite obtenida con el método de Euler.

Calcularemos ahora el error cometido por este método. Asumiendo como valor correcto el obtenido por la función *lsode* en la parte B, el error es:

$$e = (x_{1_{lsode}}; x_{2_{lsode}}) - (x_{1_{euler}}; x_{2_{euler}}) = (0,036514; -0,134042) \tag{18}$$

Modificando el paso h, vemos que el error es proporcional a éste, de acuerdo a la teoría. Por ejemplo, para $h = 0{,}001$ el error es $e = (0{,}0063200; -0{,}0160951)$ y para $h = 0{,}1$ es $e = (0,0098268; 0{,}61340)$. En el siguiente gráfico se puede ver que la dependencia entre h y la norma del error absoluto es aproximadamente lineal.

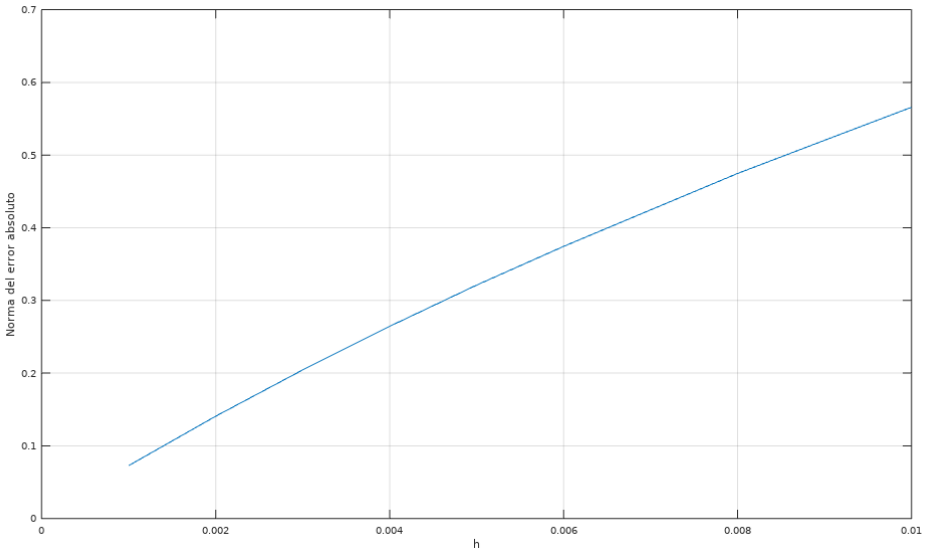

Figura 4: Error cometido con el método de Euler en función de h.

3.5. Parte E

Resolveremos el sistema en el intervalo $[0; 2]$ mediante el método de Euler Modificado (RK2) y Runge-Kutta de orden 4, utilizando las mismas ecuaciones planteadas en **??**. Los método de Runge-Kutta consiguen órdenes de convergencia similares a los de Taylor, pero no necesitan obtener las derivadas de la función $f(t, y)$. El código de la función implementada en *Octave* para la resolución con Euler Modificado se muestra a continuación.

```
function [Y] = modified_euler(f, a, b, h, y0)
  % Recibe:
  % -> f = yprima
  % -> a, b extremos del intervalo
  % -> h paso
  % -> y0 vector de condiciones iniciales
  % Devuelve:
  % -> Y matriz donde cada fila es el correspondiente vector yk = y(tk)

  t = a:h:b;
  k = int64((b - a) / h);

  Y = zeros(k, length(y0));
  Y(1, :) = y0;

  for i = 2 : k + 1

    y_n = Y(i-1, :);
    yprima_n = feval(f, Y(i - 1, :));

    q1 = yprima_n;
    q2 = feval(f, y_n + h * yprima_n);

    Y(i, :) = y_n + (h / 2) * (q1 + q2);

  end
end
```

Como mismas condiciones iniciales planteadas en la parte B del presente Trabajo Práctico. Se analizaron también los resultados tomando intervalos $h = 0{,}1$ y $h = 0{,}001$. Los resultados de cada corrida se encuentran en el repositorio en los siguientes archivos: *modified_euler-0.01.txt*,

modified_ euler-0.1.txt y *modified_ euler-0.001.txt*.
El resultado obtenido para $t = 2$, con $h = 0{,}01$ es:

$$y(2) = (-0{,}513149; -1{,}189898; 0{,}084632; -0{,}468974) \tag{19}$$

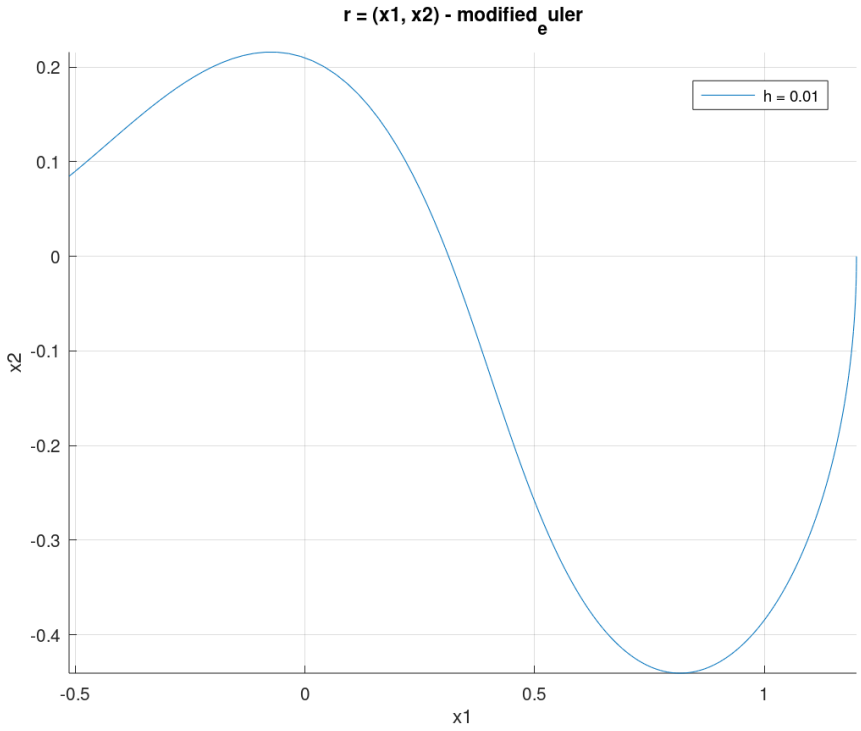

Figura 5: Trayectoria del satélite obtenida con el método de Euler Modificado.

Se comprueba que, como dice la teoría, modificando el intervalo h se tiene más precisión en los resultados, ya que para $h = 0{,}001$,

$$y(2) = (-0{,}513058; -1{,}183882; 0{,}078874; -0{,}485479)$$
$$x(2) = (-0{,}513058; 0{,}078874)$$
$$v(2) = (-1{,}183882; -0{,}485479)$$

cuyos resultados se aproximan al vector determinado por la función *lsode* de Octave.

Para la resolución mediante el método de Runge-Kutta de orden 4 se implementó la siguiente funcion:

```
function [Y] = rk4(f, a, b, h, y0)
  % Recibe:
  % -> f = yprima
  % -> a, b extremos del intervalo
  % -> h paso
  % -> y0 vector de condiciones iniciales
  % Devuelve:
  % -> Y matriz donde cada fila es el correspondiente vector yk = y(tk)

  t = a:h:b;
  k = (b - a) / h;

```

```
  Y = zeros(k, length(y0));
  Y(1, :) = y0;

  for i = 2 : k + 1

    y_n = Y(i-1, :);
    yprima_n = feval(f, Y(i - 1, :));

    q1 = yprima_n;
    q2 = feval(f, y_n + 0.5 * h * q1);
    q3 = feval(f, y_n + 0.5 * h * q2);
    q4 = feval(f, y_n + h * q3);

    Y(i, :) = y_n + (1 / 6) * h * (q1 + 2 * q2 + 2 * q3 + q4);

  end
end
```

Con las mismas condiciones iniciales, y analizando en los mismos intervalos de h que en la resolución para el 2do orden, se llegaron a los siguientes resultados:

$$y(2) = (-0{,}496993; -1{,}107211; 0{,}073956; -0{,}541870), h = 0{,}1$$
$$y(2) = (-0{,}513059; -1{,}183832; 0{,}078815; -0{,}485642), h = 0{,}01$$
$$y(2) = (-0{,}513060; -1{,}183835; 0{,}078817; -0{,}485633), h = 0{,}001$$

Los resultados de cada corrida se encuentran en el repositorio en los siguientes archivos: *rk4-0.01.txt*, *rk4-0.1.txt* y *rk4-0.001.txt*. La trayectoria obtenida es la siguiente:

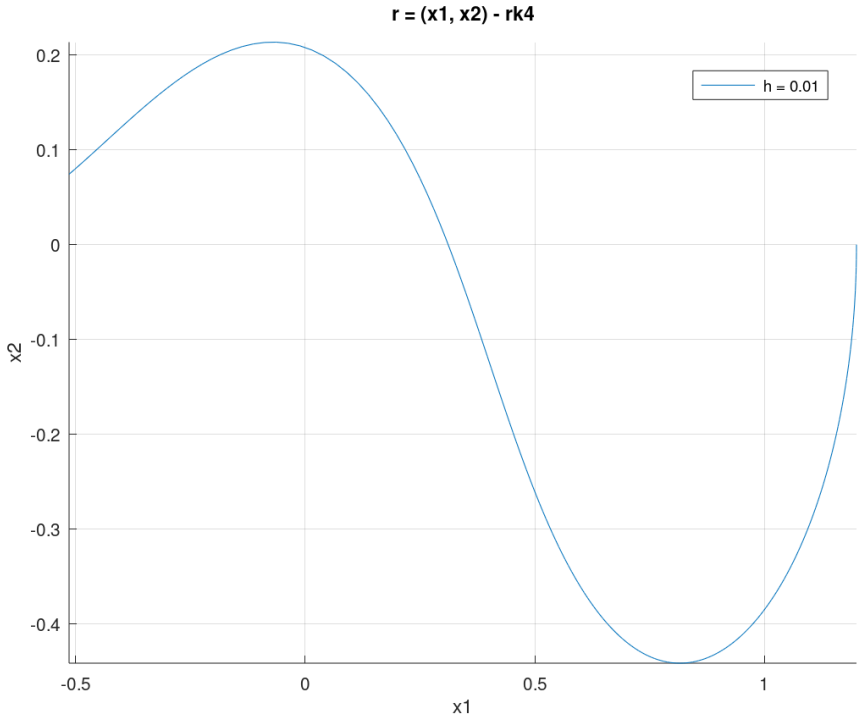

Figura 6: Trayectoria del satélite obtenida con el método de Runge-Kutta orden 4.

Para calcular el error absoluto, se implementó la siguiente función que recibe por parámetro el método sobre el cual calcularlo:

```
function error_segun_metodo(metodo, f, a, b, h, y0)
  % Recibe:
  % -> f = yprima
  % -> a, b extremos del intervalo
  % -> h paso
  % -> y0 vector de condiciones iniciales
  % -> metodo mediante el cual resolver el sistema
  % Devuelve:
  % -> error para t = b (error de la posicion final)

  yfinal_lsode = lsode(f, y0, a:h:b);
  y_metodo = feval(metodo, f, a, b, h, y0);

  rownum = rows(yfinal_lsode);
  errores = zeros(rownum, length(y0));

  for i = 1 : rownum

    errores(i, :) = yfinal_lsode(i, :) -  y_metodo(i, :);

  end

  error_t2 = errores(rownum, :);
  fprintf("(e_x1, e_x2) = (%.8f, %.8f)\n", error_t2(1), error_t2(3));

end
```

Al llamarlo mediante las siguientes expresiones:

```
    >> error_segun_metodo('modified_euler', 'yprima', 0, 2, 0.01, [1.2 0 0 -0.8])
    (e_x1, e_x2) = (0.00008750, -0.00581758)

    >> error_segun_metodo('rk4', 'yprima', 0, 2, 0.01, [1.2 0 0 -0.8])
    (e_x1, e_x2) = (-0.00000185, -0.00000042)

```

Se puede observar que, al igual que en el método de Euler, al hacer más pequeño el paso h el error tiende a ser más pequeño. Además, se observa que el método de Runge-Kutta de orden 4 converge más rápido que los otros métodos.

```
    >> error_segun_metodo('modified_euler', 'yprima', 0, 2, 0.001, [1.2 0 0 -0.8])
    (e_x1, e_x2) = (-0.00000351, -0.00005982)

    >> error_segun_metodo('rk4', 'yprima', 0, 2, 0.001, [1.2 0 0 -0.8])
    (e_x1, e_x2) = (-0.00000137, -0.00000293)

```

3.6. Parte F

Utilizando las mismas condiciones iniciales ya planteadas, procederemos a resolver el sistema mediante el método de Nystrom, utilizando como método de arranque para obtener y_1 a Euler Modificado. El código que implementa la función está preparado para correr utilizando cualquier método de los ya presentados pasando como parámetro el archivo .m que lo contiene:

```
    [Y] = nystrom('yprima', 0, 2, 0.01, [1.2 0 0 -0.8], 'modified_euler')

```

```
function [Y] = nystrom(f, a, b, h, y0, metodo_arranque)
  % Recibe:
  % -> f = yprima
  % -> a, b extremos del intervalo
  % -> h paso
  % -> y0 vector de condiciones iniciales
  % Devuelve:
  % -> Y matriz donde cada fila es el correspondiente vector yk = y(tk)

```

```
  t = a:h:b;
  k = (b - a) / h; # cantidad de pasos

  # error de truncamiento local de nystrom es O(h^2), entonces tolerancia: O(h^3)
  tolerancia = h ^ 3;
  max_iter = 1000;
  iteraciones = 1;

  Y = zeros(k, length(y0));
  Y(1, :) = y0;

  # obtengo y1 mediante el metodo de arranque
  ARRANQUE = feval(metodo_arranque, f, a, b, h, y0);
  Y(2, :) = ARRANQUE(2, :);

  for i = 3 : k + 1

    y_anterior = Y(i -1, :);
    y_anterior_anterior = Y(i - 2, :);

    y_local_anterior = Y(i -1, :);
    yprima_n = feval(f, Y(i - 1, :));
    y0_local = Y(i - 2, :) + 2 * h * yprima_n;

    while (y0_local - y_anterior > tolerancia || iteraciones < max_iter)

      y_local_anterior = y0_local;
      y0_local = 2 * y_anterior - y_anterior_anterior + h^2 * ((y_local_anterior -
    y_anterior_anterior) / (2 * h) - y_anterior);

      iteraciones += 1;

    end

    Y(i, :) = y0_local;

  end


end
```

Los resultados de cada corrida se encuentran en el repositorio en los siguientes archivos: *nystrom-0.01.txt*, *nystrom-0.1.txt* y *nystrom-0.001.txt*. Con las mismas condiciones iniciales, y analizando en los mismos intervalos de h que en las resoluciones anteriores, se llegaron a los siguientes resultados:

$$y(2) = (-0{,}748174; -1{,}080440; 0{,}484854; 0{,}989748), h = 0{,}1$$
$$y(2) = (-0{,}509491; -1{,}218613; 0{,}079715; -0{,}465028), h = 0{,}01$$
$$y(2) = (-0{,}513022; -1{,}184207; 0{,}078824; -0{,}485425), h = 0{,}001$$

Mientras que la trayectoria del satélite se puede observar en el siguiente gráfico:

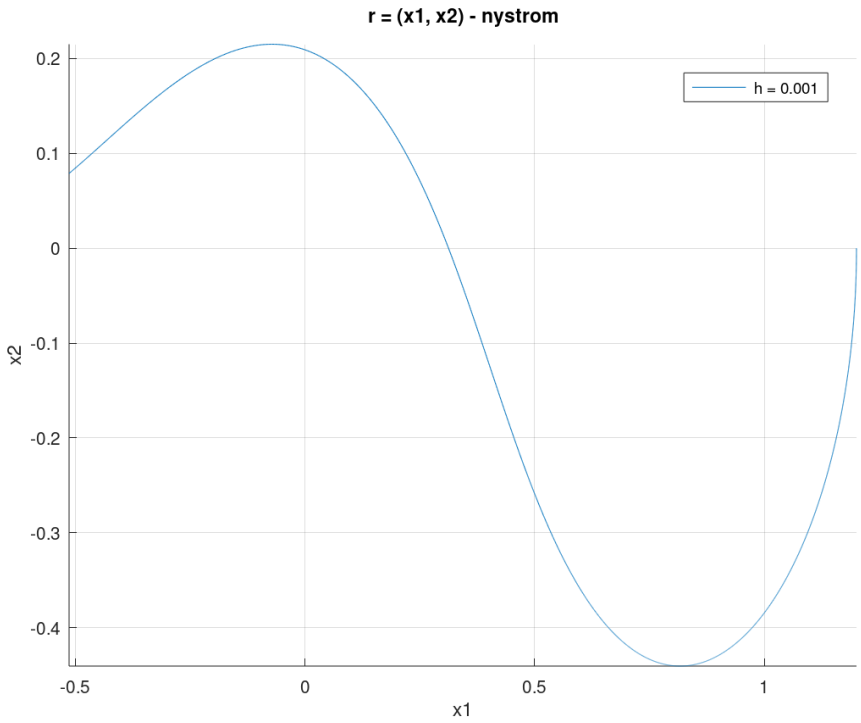

Figura 7: Trayectoria del satélite obtenida con el método de Nystrom.

Para calcular el error absoluto, se implementó la siguiente función que recibe por parámetro el método sobre el cual calcularlo:

```
function error_segun_metodo_arranque(metodo, f, a, b, h, y0, arranque)
  yfinal_lsode = lsode(f, y0, a:h:b);
  y_metodo = feval(metodo, f, a, b, h, y0, arranque);

  rownum = rows(yfinal_lsode);
  errores = zeros(rownum, length(y0));

  for i = 1 : rownum

    errores(i, :) = yfinal_lsode(i, :) -  y_metodo(i, :);

  end

  error_t2 = errores(rownum, :);
  fprintf("(e_x1, e_x2) = (%.8f, %.8f)\n", error_t2(1), error_t2(3));

end
```

Al llamarlo mediante las siguientes expresiones:

```
    >> error_segun_metodo_arranque('nystrom', 'yprima', 0, 2, 0.1, [1.2 0 0 -0.8],
    'modified_euler')
    (e_x1, e_x2) = (0.23511281, -0.40603998)
    >> error_segun_metodo_arranque('nystrom', 'yprima', 0, 2, 0.01, [1.2 0 0 -0.8],
     'modified_euler')
    (e_x1, e_x2) = (-0.00356984, -0.00090063)
    >> error_segun_metodo_arranque('nystrom', 'yprima', 0, 2, 0.001, [1.2 0 0
    -0.8], 'modified_euler')
    (e_x1, e_x2) = (-0.00003906, -0.00001002)

```

se observa que nuevamente al modificar el intervalo de h, se realizan más iteraciones y en consecuencia el error absoluto es más pequeño.

Referencias

[1] Richard L. Burden, J. Douglas Faires
Numerical Analysis (9th edition).
Brooks/Cole - Cengage Learning
Sección 5.9 Higher Order Equations and Systems of Differential Equations
Páginas 328-334

[2] Richard L. Burden, J. Douglas Faires
Numerical Analysis (9th edition).
Brooks/Cole - Cengage Learning
Sección 5.9 Higher Order Equations and Systems of Differential Equations
Páginas 334-336

[3] Richard L. Burden, J. Douglas Faires
Numerical Analysis (9th edition).
Brooks/Cole - Cengage Learning
Sección 5.2 Euler's Method
Páginas 266-268

[4] UBC Math - University of British Columbia
Error Analysis of the Euler Method
http://www.math.ubc.ca/~israel/m215/euler2/euler2.html