FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Análisis numérico I - 75.12/95.04

Trabajo práctico N° 2

Sistemas de ecuaciones diferenciales ordinarias

Alumnos:

José Higuera — Padrón N° 100251
jhiguera@fi.uba.ar

Diego Luna — Padrón N° 75451
diegorluna@gmail.com

Juan Segundo Marquez — Padrón N° 100556
segundoprez777@gmail.com

Juan Manuel Maschio — Padrón N° 100801
juanmmaschio@gmail.com

Docentes:

Mag. Ing. Miryam Sassano
Ing. Ignacio Bello
Ing. Matías Payva
Lic. Andrés Porta
Ing. Ezequiel García
Ing. Ignacio Santiago Cerruti

29 de Diciembre de 2019

Índice

Índice de figuras

1. Enunciado

1.1. Resumen enunciado

Este TP consiste en realizar una implementación del algoritmo Runge-Kutta de orden 4, para cuyo testeo se propuso resolver el sistema de ecuaciones diferenciales que se obtiene a partir de la conocida ecuación diferencial de segundo orden que se obtiene al plantear la oscilación de un péndulo:

$$\ddot{\theta} + \frac{b}{m} \cdot \dot{\theta} + \frac{g}{l} \cdot sin\,(\theta) = 0 \tag{1.1.1}$$

Planteando:

$$\theta = x_1$$
$$\dot{\theta} = x_2$$

Y reemplazando en la ecuación (1.1.1), se obtiene el sistema:

$$L_0 = \begin{cases} \dot{x_1} = x_1 \\ \dot{x_2} = -\frac{b}{m} \cdot x_2 - \frac{g}{l} \cdot sin\,(x_1) \end{cases} \tag{1.1.2}$$

Además de lo anterior, se pide que se implemente el método de integración de Romberg, y se aplique al cálculo del área encerrada bajo la curva del módulo del desplazamiento del péndulo.

1.2. Resolución

Para la resolución de la parte de programación del trabajo práctico e implementar los algoritmos pedidos, decidimos usar **MATLAB**, mayormente por conocerlo previamente y la sencillez con la que se pueden escribir scripts que implementen los algoritmos. A pesar de que la resolución se realizó en **MATLAB**, se prestó atención a la compatibilidad con **Octave**, ya que la compatibilidad en los paquetes básicos es alta y con un poco de cuidado y algo de programación condicional se puede lograr que los scripts funcionen en ambos entornos. Todos los resultados numéricos se guardaron por código desde **MATLAB** en formato **“CSV”** y las imágenes se guardaron también por código en formato **“PNG”** y luego se incorporaron desde LaTeX.

2. Implementación de los algoritmos

El script resuelve en principio los casos pedidos en el enunciado, y luego toma interactivamente con diálogos nuevos parámetros al usuario, los cuales valida y utiliza para resolver el nuevo sistema, en cada caso se grafica las funciones de posición (angulo) y su derivada (velocidad angular) y se calcula la integral del módulo de la posición (angulo) en el intervalo.

A continuación se listan los archivos de **MATLAB** y su función:

"tp2.m" (apéndice [A.2.1]): Script principal que se debe ejecutar para realizar todos los cálculos y generar los archivos de resultados.

"pendulum.m" (apéndice [A.2.2]): Función que invoca nuestra implementación de Runge-Kutta de orden 4 para el sistema del péndulo.

"rk4.m" (apéndice [A.2.3]): Función con nuestra implementación del algoritmo Runge-Kutta de orden 4 en forma genérica matricial.

"plot_solution.m" (apéndice [A.2.4]): Función que grafica el ángulo y la velocidad obtenidas.

"trapezcomp.m" (apéndice [A.2.6]): Función que implementa el método de integración de trapecios compuesto (usado en Romberg).

"romberg.m" (apéndice [A.2.5]): Función que implementa el método de integración de Romberg.

En el apéndice correspondiente (apéndice [A.2]) se incluye el código completo de cada archivo.

"salida.txt" (apéndice [B.2.1]): La salida del script principal, se captura automáticamente en **MATLAB** al ejecutar el script principal.

Algo importante a aclarar, es que decidimos implementar el método de integración de Romberg en su forma "standard", en esta forma se requiere la evaluación de la función a integrar en puntos arbitrarios, para poder luego integrar la función que representa el módulo de la posición del péndulo, se interpoló la función entre los puntos obtenidos con Runge-Kutta, usando un spline. Esta no era la única forma de realizar la integración, pero era la que mas se adaptaba a usar el algoritmo de Romberg sin modificaciones, además era muy simple dado que **MATLAB** y **Octave** proveen la función **interp1** que puede realizar distintos tipos de interpolación sobre un array de puntos, entre ellas, spline. El método de Romberg se implementó,así como está descripto en la bibliografía, usando interpolación de Richardson y el método de trapecios compuesto, para lo cual se implementó el mismo, también en su forma “standard”.

El error del método de integración de Romberg ($R_{n,n}$), asumiendo que la función es suficientemente diferenciable, está en $O\left(h^{(2\cdot n+2)}\right)$, donde h es el paso, que en el caso de esté método, vale en cada iteración $h = \frac{1}{2^n} \cdot (a - b)$, siendo a y b los límites de integración y n el número de iteración.

El hecho de que usemos interpolación en la función que integramos, sin embargo, hace que este error no sea del todo correcto, una determinación mas detallada, haría necesario analizar como afecta el spline este resultado. Dado que usamos un nivel de Romberg de 6 y el intervalo tiene en todos los casos una longitud de $20s$, ignorando la cuestión de la interpolación, tendríamos un error del orden de 0,3.

2.1. Sobre los archivos de MATLAB y Octave

Hay algunas cosas a comentar sobre las diferencias entre **MATLAB** y **Octave**, como se comentó anteriormente, se logró la compatibilidad de ejecución entre los entornos, sin embargo las salidas no son completamente equivalentes, debido a limitaciones en **Octave**, las salidas gráficas no son completamente equivalentes, en particular **MATLAB** permite la generación de **DataTips**, cosa que **Octave** aún no soporta, otra cuestión quizás mas importante es la eficiencia en ejecución, en algunos casos los tiempo de ejecución en **Octave** se hacen demasiado largos si se usan arrays muy extensos, lo cual se mitigó usando compilación condicional. Otra cosa a mencionar que no hace a la funcionalidad directamente, pero si a la presentación, es que debido a limitaciones en ambos entornos respecto a la codificación de los archivos y soporte incompleto o inadecuado de **UNICODE** en la línea de comando de **Windows**, se producen problemas en las salidas con símbolos que no sean parte de Latin-1 (ISO 8859-1), en particular las palabras con tilde, esto se ve aún mas complicado porque **Windows** usa una variante (CP1252) que no es completamente compatible con Latin-1 y el hecho de que los entornos de **MATLAB** y **Octave** no se comportan consistentemente en **Windows** y sistemas tipo Unix como **Linux**, en Unix es prácticamente universal la codificación de **UNICODE**, **UTF-8**. **MATLAB** sigue la codificación del sistema operativo, mientras que **Octave** intenta usar **UTF-8** siempre, pero en **Windows** no es completo el soporte. El tema es complicado y no hace al trabajo práctico el lidiar con el mismo, dado que trabajamos mayormente con **MATLAB**, tanto en **Windows** como en **Linux**, y la mayoría usa **Windows**, se optó por dejar los archivos en **CP1252**, siendo esta la codificación usual. Se incluyen simplemente por comodidad dos scripts de **Python**, **"utf8.py"** y **"cp1252.py"**, que convierten la codificación de todos los archivos **".m"** a las respectivas codificaciones, de esa manera, según el sistema en que se ejecuten los scripts, se puede lograr una salida con codificación correcta.

3. Aproximación de la solución del sistema

Para aproximar el valor de las soluciones del sistema del péndulo se escribió una función específica (apéndice [A.2.2]) que declara la función del sistema, los parámetros, las condiciones iniciales, llama a nuestra implementación del algoritmo de Runge-Kutta de orden 4, y devuelve la aproximación de la solución, luego el script principal se encarga de llamar a las funciones que grafican lo pedido. La implementación del método de Runge-Kutta de orden 4 (apéndice [A.2.3]), se realizó en forma matricial, lo cual permitió un código muy simple y compacto, y conformando a los mismos parámetros que toman las funciones incluidas en **MATLAB** u **Ocatve**, como ser **ode23b**, o **ode23s**, lo cual nos permitió utilizar las mismas inicialmente para ver los resultados esperados de nuestra implementación, luego de implementada, fue solo cuestión de reemplazar las llamadas a las funciones del entorno por la nuestra.

4. Primer caso

A continuación se muestran los gráficos para la serie de tiempos para los valores iniciales $x_1(0) = 5$, $x_2(0) = 5$, $x_3(0) = 30$ con $t \in [0; 50]$.

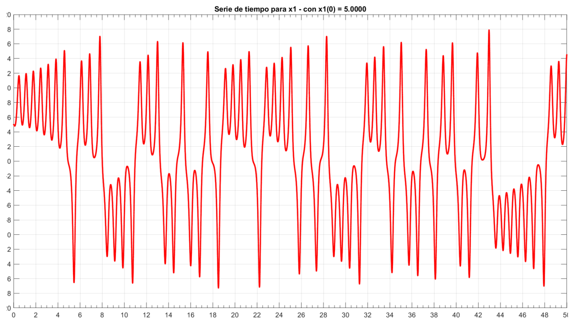

Figura 4.1: Gráfico de la serie de tiempo para x_1 con $x_1(0) = 5$.

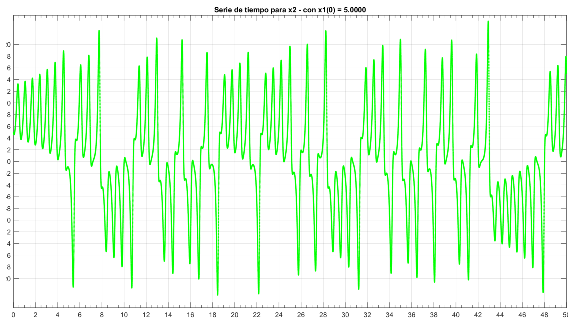

Figura 4.2: Gráfico de la serie de tiempo para x_2 con $x_1(0) = 5$.

Figura 4.3: Gráfico de la serie de tiempo para x_3 con $x_1(0) = 5$.

4.1. Observación del comportamiento

El comportamiento asintótico de las soluciones es oscilante, sin embargo no se discierne una periocidad en las mismas. Las oscilaciones observadas parecen ser entre valores que se repiten a lo largo del intervalo, siendo distintos para cada serie de tiempo.

5. Segundo caso

A continuación se muestran los gráficos para la serie de tiempos para los valores iniciales $x_2(0) = 5$, $x_3(0) = 30$ con $t \in [0; 50]$, pero ahora variando el valor incial de la primera función solución en los valores $x_1(0) = 5{,}01$, $x_1(0) = 5{,}001$ y $x_1(0) = 5{,}0001$.

5.1. Series de tiempo para x_1

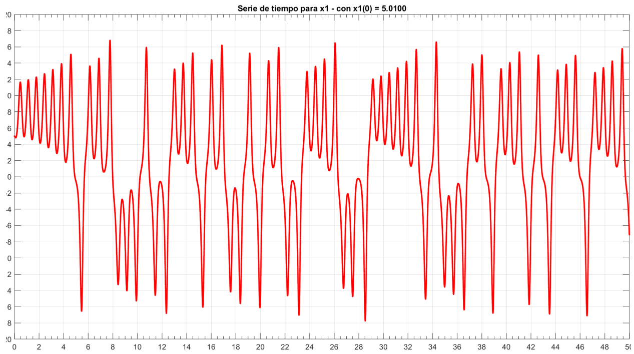

Figura 5.1: Gráfico de la serie de tiempo para x_1 con $x_1(0) = 5{,}01$.

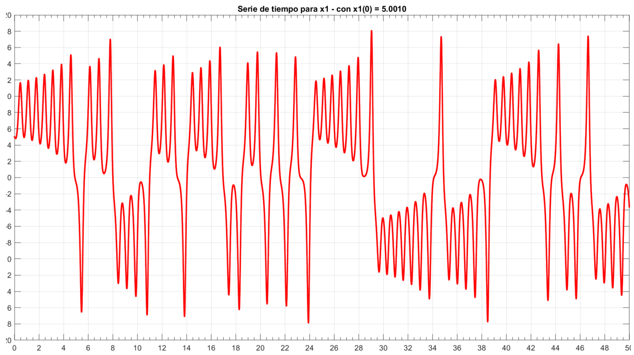

Figura 5.2: Gráfico de la serie de tiempo para x_1 con $x_1(0) = 5{,}001$.

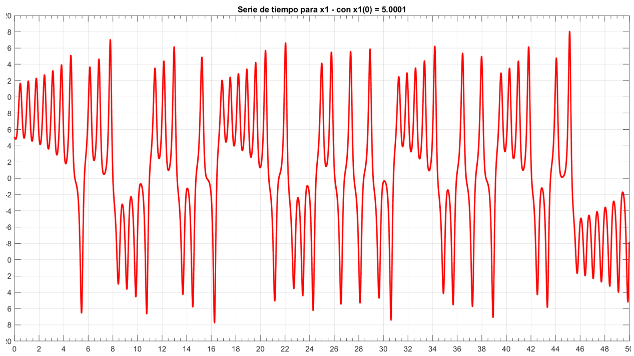

Figura 5.3: Gráfico de la serie de tiempo para x_1 con $x_1(0) = 5{,}0001$.

5.2. Series de tiempo para x_2

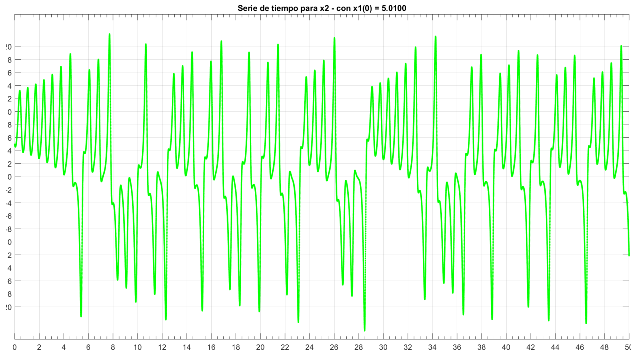

Figura 5.4: Gráfico de la serie de tiempo para x_2 con $x_1(0) = 5{,}01$.

Figura 5.5: Gráfico de la serie de tiempo para x_2 con $x_1(0) = 5{,}001$.

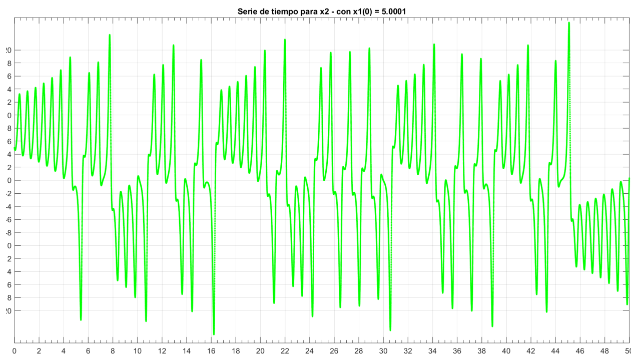

Figura 5.6: Gráfico de la serie de tiempo para x_2 con $x_1(0) = 5{,}0001$.

5.3. Series de tiempo para x_3

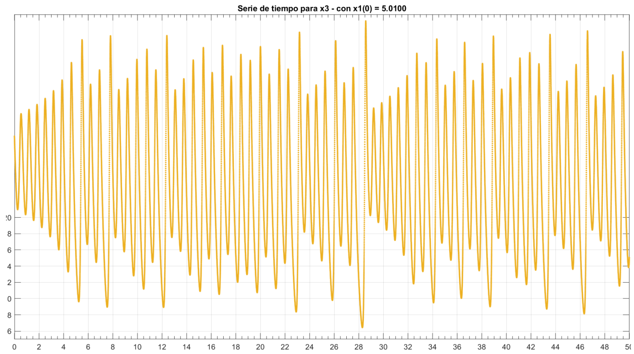

Figura 5.7: Gráfico de la serie de tiempo para x_3 con $x_1(0) = 5{,}01$.

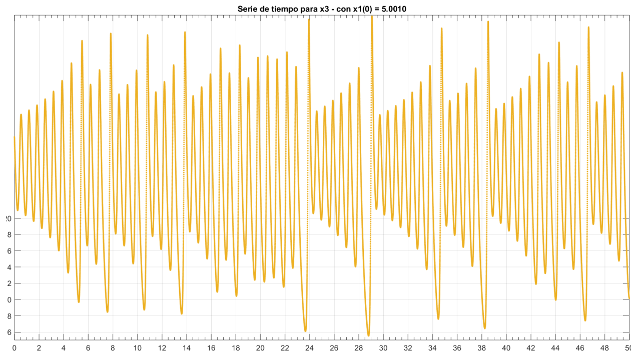

Figura 5.8: Gráfico de la serie de tiempo para x_3 con $x_1(0) = 5{,}001$.

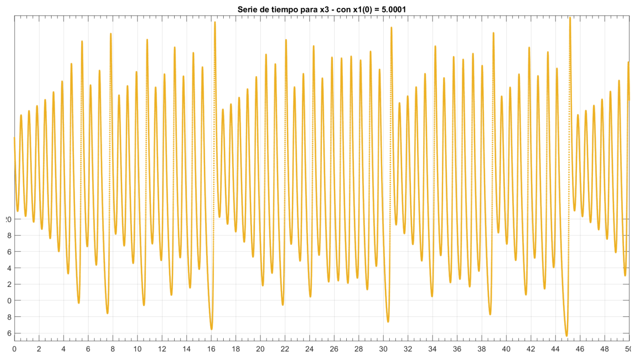

Figura 5.9: Gráfico de la serie de tiempo para x_3 con $x_1(0) = 5{,}0001$.

5.4. Observación del comportamiento

Se puede observar que la serie de tiempos es muy sensible a las variaciones de las condiciones inicales, con solo cambiar en muy poco solo uno de los valores inciales se observa que a pesar de que las series de tiempo siguen siendo oscilantes, los valores de las oscilaciones y su exacta forma es completamente distinta en cada caso.

6. Otros casos

A continuación se muestran los gráficos para la serie de tiempos para los valores iniciales $x_2(0) = 5$, $x_3(0) = 30$ con $t \in [0; 50]$, pero ahora variando el valor incial de la primera función solución en los valores $x_1(0) = 5{,}01$, $x_1(0) = 5{,}001$ y $x_1(0) = 5{,}0001$.

6.1. Series de tiempo para x_1

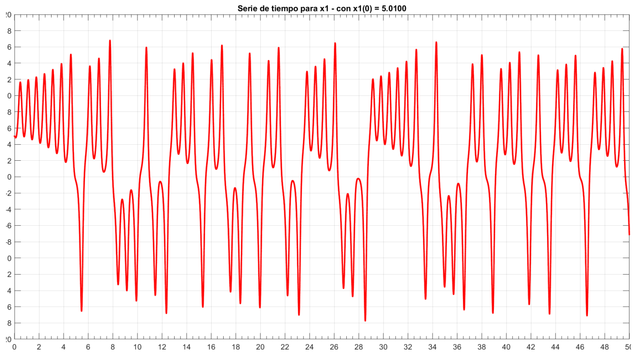

Figura 6.1: Gráfico de la serie de tiempo para x_1 con $x_1(0) = 5{,}01$.

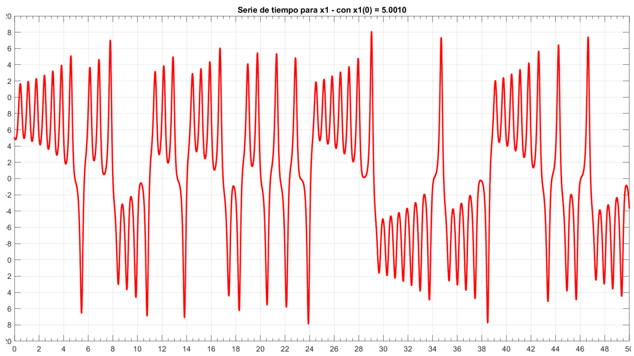

Figura 6.2: Gráfico de la serie de tiempo para x_1 con $x_1(0) = 5{,}001$.

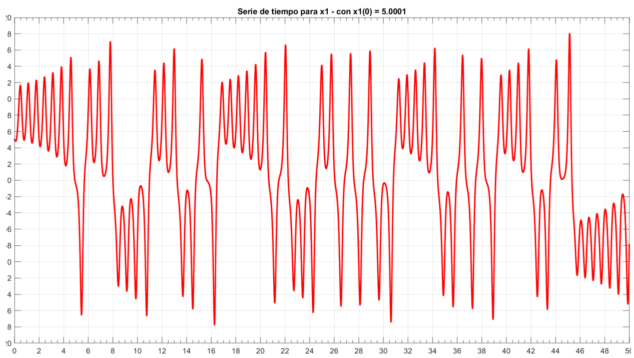

Figura 6.3: Gráfico de la serie de tiempo para x_1 con $x_1(0) = 5{,}0001$.

6.2. Series de tiempo para x_2

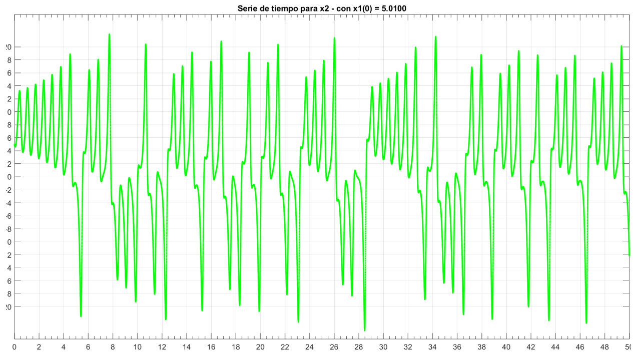

Figura 6.4: Gráfico de la serie de tiempo para x_2 con $x_1(0) = 5{,}01$.

Figura 6.5: Gráfico de la serie de tiempo para x_2 con $x_1(0) = 5{,}001$.

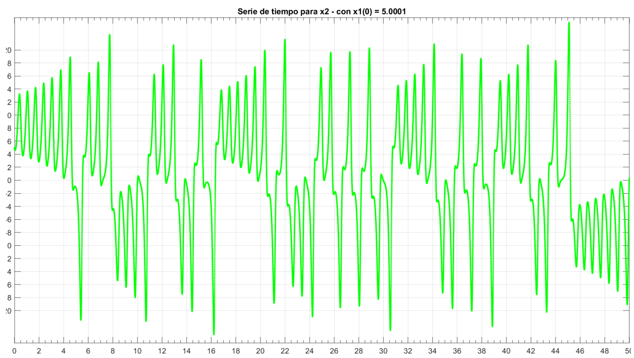

Figura 6.6: Gráfico de la serie de tiempo para x_2 con $x_1(0) = 5{,}0001$.

6.3. Series de tiempo para x_3

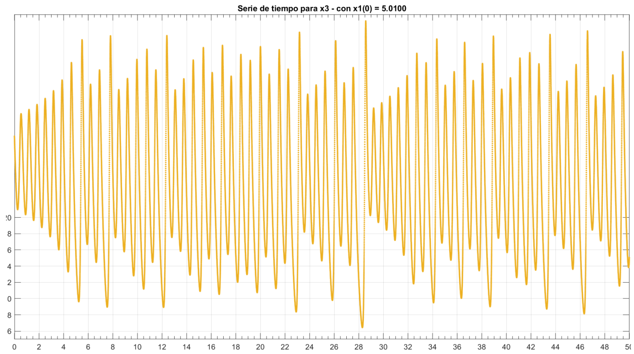

Figura 6.7: Gráfico de la serie de tiempo para x_3 con $x_1(0) = 5{,}01$.

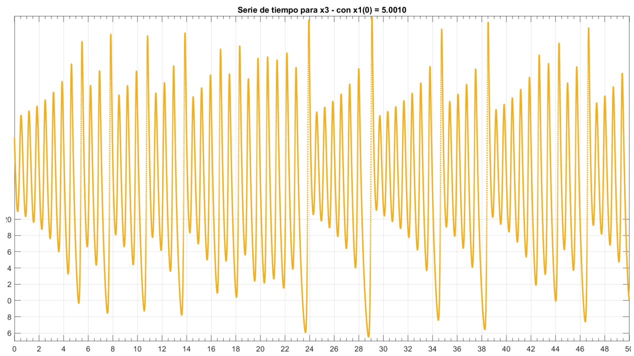

Figura 6.8: Gráfico de la serie de tiempo para x_3 con $x_1(0) = 5{,}001$.

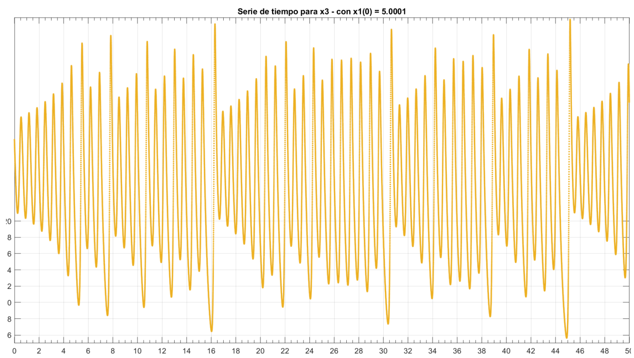

Figura 6.9: Gráfico de la serie de tiempo para x_3 con $x_1(0) = 5{,}0001$.

6.4. Observación del comportamiento

Se puede observar que la serie de tiempos es muy sensible a las variaciones de las condiciones inicales, con solo cambiar en muy poco solo uno de los valores inciales se observa que a pesar de que las series de tiempo siguen siendo oscilantes, los valores de las oscilaciones y su exacta forma es completamente distinta en cada caso.

7. Observaciones y conclusiones

El método numérico implementado en el caso propuesto se comporta en general como es esperado, sirvió la comparación con los métodos proveídos por **MATLAB** u **Octave** y también tener un problema planteado para el cual la solución se conoce de antemano. El algoritmo de Runge-Kutta fue implementado en forma matricial, lo cual a pesar de ser mas complicado de pensar inicialmente, permite lograr un código mas simple y entendible.

8. Bibliografía

Referencias

[1] *Álgebra lineal y ecuaciones diferenciales con MATLAB (1er Edición)*
Author: M. Golubitsky
Author: Dellnitz, M.
Publisher: INTERNATIONAL THOMSON EDITORES Edición (2001)
Copyright: © 2001 INTERNATIONAL THOMSON EDITORES.
ISBN 13: 978-9-706-86040-8
Website: https://www.marcialpons.es/libros

[2] *The PgfplotsTable Package*
Author: Dr. Christian Feuersänger
Copyright: © 2018, Christian Feuersänger.
Website: https://sourceforge.net/projects/pgfplots/

[3] *The Listings Package*
Author: Carsten Heinz
Author: Brooks Moses
Copyright: © 1996–2004, Carsten Heinz; © 2006–2007, Brooks Moses.
Website: http://www.ctan.org/pkg/listings

[4] *The Listingsutf8 Package*
Author: Heiko Oberdiek
Copyright: © 2007, Heiko Oberdiek.
Website: http://www.ctan.org/pkg/listingsutf8

Apéndices

A. Código fuente

A.1. Consideraciones para el código

El código está escrito en **MATLAB**, se trato de hacerlo ordenado y con todos los comentarios necesarios, así como también se hizo un uso consistente del indentado con tabulaciones a 4 espacios. La presentación del código en el informe se hizo directamente desde los fuentes hacia LaTeX usando el package **"listingsutf8"** [4], que es una extensión con soporte para **UTF8** del paquete **"listings"** [3], este paquete produce una salida formateada y con coloreado del código y también permite el agregado de números de líneas, el código fuente en **MATLAB** es soportado directamente, la salida que se produce es muy buena, pero no es perfecta, ya que cuestiones como el tabulado o el ancho total de las líneas pueden producir problemas, en caso de que alguno de estos problemas hagan confuso o incomprensible el código por favor remitirse a los fuentes originales.

A.2. Archivos fuente de MATLAB

A.2.1. tp2.m

```
% Implementa el TP2, declarando la variables necesarias y llamando a las
% respectivas funciones. Los gráficos son salvados en formato PNG en el
% correspondiente directorio del informe, también los resultados numéricos
% son salvados en el correspondiente directorio del informe, donde el
% código de Latex los levanta automáticamente para generar el archivo
% compilado final del informe.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Limpio todas las variables globales.
clear all;

% Cierro todos los gráficos.
close all;

% Determino si estoy trabajando en MATLAB u Octave.
Is_Octave = (5 == exist('OCTAVE_VERSION', 'builtin'));

% Determino el OS en el que estoy trabajando.
if (ismac)
    OS = 'Mac';
    % Mac plaform.
    % En general al igual que en Linux y otros Unix, se usa UTF-8, pero
    % no es necesariamente así.
elseif (isunix)
    OS = 'Linux';
    % Linux plaform.
    % En general se usa UTF-8, con lo que bastaría con codificar los
    % scripts en UTF-8 para que la codificación sea correcta.
elseif (ispc)
    OS = 'Windows';
    % Windows platform.
    % Esto es necesario mayormente para Octave en Windows, en MATLAB
    % CP1252 es el default. Los scripts deberían estar codificados
    % en CP1252.

    if (Is_Octave)
        major = int8(str2double(substr(OCTAVE_VERSION, 1, ...
            index(OCTAVE_VERSION, ".") - 1)));

        if (major >= 5)
            [~, ~] = system ('chcp 65001');
        else
```

```
            [~, ~] = system('chcp 1252');
        end

    else
        [~, ~] = system('chcp 1252');
    end

else
    OS = 'Sistema desconocido';
end %if

% Limpio la línea de comando, para poder capturar la salida limpia.
clc;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%

% Defino el archivo para la captura de la salida del script.
output_file = './salida.txt';

% Detengo la captura de la salida del script.
diary off;

% Borro el archivo si existía previamente.
if (exist(output_file, 'file'))
    delete(output_file);
end

% Borro el archivo si existía previamente.
% if (exist('diary', 'file'))
%     delete('diary');
% end

% Inicio la captura de la salida.
diary on;

% Inicio la ejecución.
fprintf('Inicializando las variables globales para el TP2...');

% Declaro el directorio para las imágenes.
images_directory = fullfile('..', 'informe', 'img');

% Declaro el directorio para los archivos ".csv".
results_directory = fullfile('..', 'informe', 'results');

```

```
% Declaro los nombres de los archivos a guardar.
grafico_respuesta_prefix = 'grafico_respuesta';
grafico_1 = '_1';
grafico_2 = '_2.png';
solutions_prefix = 'valores_respuesta';
respuesta_1 = '_1';
respuesta_2 = '_2';

% Tiempo final.
tend = 20;

% Cantidad de filas para Romberg.
romberg_levels = 10;

% En el caso de Ocatve reduzco los nivels porque es menos eficiente y
% tarda demasiado.
if (Is_Octave)
    romberg_levels = 6;
end % if

fprintf('Listo\n\n');

if (~Is_Octave) % Si estoy trabajando en MATLAB.
    env = 'MATLAB';
else            % Si estoy trabajando en Octave.
    env = 'Octave';
end %if

fprintf('Trabajando en: %s %s sobre %s.\n\n\n\n', env, version, OS);


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%

fprintf('Creando el directorio para las imágenes...');

% Creo el directorio para las imágenes y verifico que exista.
[~, ~, ~] = mkdir(images_directory);
success = (7 == exist(images_directory, 'dir'));

% Chequeo que el directorio para las imágenes exista.
if (~success)
    fprintf(strjoin({'\nNo se pudo crear ', ...
        'el directorio para las imágenes.\n\n'}, ''));

    exit;
```

```
else
    fprintf('Listo\n\n');
end

fprintf('Creando el directorio para los resultados numéricos...');

% Creo el directorio para los archivos numéricos y verifico que exista.
[~, ~, ~] = mkdir(results_directory);
success = (7 == exist(results_directory, 'dir'));

% Chequeo que el directorio para los archivos numéricos exista.
if (~success)
    fprintf(strjoin({'\nNo se pudo crear' , ...
        'el directorio para los resultados.\n\n'}, ''));

    exit;
else
    fprintf('Listo\n\n');
end

% Seteo el formato de números por pantalla.
format long;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Cálculo para el primer juego de parámetros.

fprintf(strjoin({'Calculando la estimación de '...
    'la solución del sistema con m = 1Kg, l = 1m, ' ...
    'b = 0Ns/m, h = 0.2s, Theta0 = pi/6, Omega0 = 0...'}, ''));

% Resuelvo la ecuación del péndulo para los parámetros dados.
% tend, h, b, l, m, Theta_0, Omega_0
[t, x, ~] = pendulum(tend, 0.2, 0, 1, 1, pi/6, 0);

% Generación completa.
fprintf('Listo\n\n');

% Guardo los resultados en un archivo.
fprintf('Salvando los resultados en un archivo "CSV"......');

% Armo la tabla de resultados finales con:
% 1 - Tiempo.
% 2 - Theta.
% 3 - d(Theta)/dt.
results = [t', x(:,1), x(:,2)];
```

```

% Guardo la tabla de las iteraciones.
dlmwrite(fullfile(results_directory, ...
    strjoin({solutions_prefix, respuesta_1, '.csv'}, '')), ...
    results, 'precision', '%.15f');

% Guardado completo.
fprintf('Listo\n\n');

% Genero el gráfico de la  solución.
fprintf('Generando un gráfico de la solución...');

graphic_handle1 = ...
    plot_solution(t, x(:,1)', x(:,2)', 'Ángulo y velocidad', 100);

% Generación completa.
fprintf('Listo\n\n');

%%%
solution_complete_name = ...
    fullfile(images_directory, ...
    strjoin({grafico_respuesta_prefix, grafico_1, ...
    '.png'}, ''));

fprintf('Salvando el gráfico en un archivo "PNG"......');

% Salvo el gráfico en un archivo.
saveas(graphic_handle1, solution_complete_name);

% Salvado completo.
fprintf('Listo\n\n');
%%%

fprintf('Calculando la integral del módulo de la posición......');

% Genero la función del módulo interpolada usando spline.
fint = @(y) interp1(t, abs(x(:,1)'),  y, 'spline');

% Calculo la integral usando Romberg.
I_romb = romberg(fint, 0, tend, romberg_levels);

% Listo.
fprintf('Listo\n\n');

fprintf(strjoin({'El valor de la integral aproximada con Romberg del ' ...
    'módulo de la posición es: %.16f.\n\n'}, ''), ...
```

```
    I_romb(size(I_romb,1), size(I_romb,2)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Cálculo para el segundo juego de parámetros.

fprintf(strjoin({'Calculando la estimación de '...
    'la solución del sistema con m = 1Kg, l = 1m, ' ...
    'b = 0.5Ns/m, h = 0.2s, Theta0 = pi/6, Omega0 = 5*pi/6...'}, ''));

% Resuelvo la ecuación del péndulo para los parámetros dados.
% tend, h, b, l, m, Theta_0, Omega_0
[t, x, s] = pendulum(tend, 0.2, 0.5, 1, 1, pi/6, 5*pi/6);

% Generación completa.
fprintf('Listo\n\n');

% Guardo los resultados en un archivo.
fprintf('Salvando los resultados en un archivo "CSV"......');

% Armo la tabla de resultados finales con:
% 1 - Tiempo.
% 2 - Theta.
% 3 - d(Theta)/dt.
results = [t', x(:,1), x(:,2)];

% Guardo la tabla de las iteraciones.
dlmwrite(fullfile(results_directory, ...
    strjoin({solutions_prefix, respuesta_2, '.csv'}, '')), ...
    results, 'precision', '%.15f');

% Guardado completo.
fprintf('Listo\n\n');

% Genero el gráfico de la  solución.
fprintf('Generando un gráfico de la solución...');

graphic_handle2 = ...
    plot_solution(t, x(:,1)', x(:,2)', 'Ángulo y velocidad', 100);

% Generación completa.
fprintf('Listo\n\n');

%%%
solution_complete_name = ...
    fullfile(images_directory, ...
```

```
    strjoin({grafico_respuesta_prefix, grafico_2, ...
    '.png'}, ''));

fprintf('Salvando el gráfico en un archivo "PNG"......');

% Salvo el gráfico en un archivo.
saveas(graphic_handle2, solution_complete_name);

% Salvado completo.
fprintf('Listo\n\n');
%%%

fprintf('Calculando la integral del módulo de la posición......');

% Genero la función del módulo interpolada usando spline.
fint = @(y) interp1(t, abs(x(:,1)'),  y, 'spline');

% Calculo la integral usando Romberg.
I_romb = romberg(fint, 0, tend, romberg_levels);

% Listo.
fprintf('Listo\n\n');

fprintf(strjoin({'El valor de la integral aproximada con Romberg del ' ...
    'módulo de la posición es: %.16f.\n\n'}, ''), ...
    I_romb(size(I_romb,1), size(I_romb,2)));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Cálculo para parámetros dados por el usuario.

while (1)

    % Ingreso los valores iniciales.

    % Masa.
    m = 1;

    % Longitud del hilo.
    l = 1;

    % Rozamiento.
    b = 0.5;

    % Paso.
    h = 0.001;
```

```

    % Ángulo inicial.
    Theta_0 = pi/6;

    % Velocidad angular inicial.
    Omega_0 = 5*pi/6;

    % En el caso de Octave, un valor muy chico causa problemas
    if (Is_Octave)
        h = 0.01;
    end

    answer = questdlg(strjoin({'¿Desea resolver el sistema', ...
        ' para otros parámetros?'}, ''), ...
        'Pregunta', 'Si', 'No', 'No');

    if ( ~strcmp('Si', answer))
        break;
    end %if

    % Pido los datos para resolver el problema.
    answer = inputdlg({'Masa del péndulo [Kg]','Largo del hilo (m)', ...
        'Rozamiento [Kg/s]', 'Paso [s]', 'Ángulo inicial [rad]', ...
        'Velocidad angular inicial [rad/s]'},...
        'Parámetros del sistema', ...
        [1 50; 1 50; 1 50; 1 50; 1 50; 1 50], ...
        {num2str(m, 16) num2str(l, 16) num2str(b, 16) num2str(h, 16) ...
        num2str(Theta_0, 16) num2str(Omega_0, 16)});

    % Valido los datos.
    if (isempty(answer))

        dlg = errordlg('Parámteros inválidos', 'Error', 'modal');

        uiwait(dlg);

        continue;

    end % if

    m = str2double(answer{1});

    l = str2double(answer{2});

    b = str2double(answer{3});

```

```
    h = str2double(answer{4});

    Theta_0 = str2double(answer{5});

    Omega_0 = str2double(answer{6});

    % Valido los datos ingresados.
    if isnan(h) || isnan(b) || isnan(l) || isnan(m) || isnan(Theta_0) || ...
            isnan(Omega_0) || (b < 0) || (l <= 0) || (m <= 0) || ...
            ((Theta_0 == 0) && (Omega_0 == 0))

        dlg = errordlg('Parámteros inválidos', 'Error', 'modal');

        uiwait(dlg);

        continue;

    end %if

    fprintf(strjoin({'Calculando la estimación de '...
        'la solución del sistema...'}, ''));

    % Resuelvo la ecuación del péndulo para los parámetros dados.
    [t, x, s] = pendulum(tend, h, b, l, m, Theta_0, Omega_0);

    if (~s)
        continue;
    end % fi

    % Listo.
    fprintf('Listo\n\n');

    if exist('graphic_handle', 'var') == 1
        % Cierro el gráfico previo.
        close(graphic_handle);
    end

    % Genero el gráfico de la  solución.
    fprintf('Generando un gráfico de la solución...');

    % Genero el gráfico de las soluciones.
    graphic_handle = plot_solution(t, x(:,1)', x(:,2)', ...
        'Ángulo y velocidad', 100);

    % Listo.
    fprintf('Listo\n\n');
```

```
    fprintf('Calculando la integral del módulo de la posición......');

    % Genero la función del módulo interpolada usando spline.
    fint = @(y) interp1(t, abs(x(:,1)'),  y, 'spline');

    % Calculo la integral usando Romberg.
    I_romb = romberg(fint, 0, tend, romberg_levels);

    % Listo.
    fprintf('Listo\n\n');

    fprintf(strjoin({'El valor de la integral aproximada con Romberg del ' ...
        'módulo de la posición es: %.16f.\n\n'}, ''), ...
        I_romb(size(I_romb,1), size(I_romb,2)));

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
% Guardo la salida del programa.

% El script se ejecutó correctamente..
fprintf('\n\nEjecución del TP2 terminada.\n\n');

% Detengo la captura de la salida del script.
diary off;

% Copio el archivo de salida.
fprintf('Copiando el archivo de salida......');

[status, ~] = copyfile('diary', output_file);

% Chequeo que la copia se realizó correctamente.
if (~status)
    fprintf(strjoin({'\nFalló la copia', ...
        'del archivo de salida.\n\n'}));

    return;
else %if
    fprintf('Listo\n\n');
end %if

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

A.2.2. pendulum.m

```
% Resuelve el sistema del péndulo en el intervalo [0, tend] y con los
% parámetros dados, o unos tomados por default.
%
% Parámetros:
% -----------
% tend:      Final del intervalo en el cual aproximar las funciones
%            solución del sistema (Se asume el inicio en 0).
% h:         Paso deseado para el cálculo de las aproximaciones, el valor
%            finalmente usado puede que sea menor para acomodarse al
%            intervalo de aproximación.
% b:         Coeficiente de rozamiento viscoso.
%            [Este parámetro es opcional, el default es 0 Kg/s].
% l:         Longitud del hilo.
%            [Este parámetro es opcional, el default es 1 m].
% m:         Masa del péndulo.
%            [Este parámetro es opcional, el default es 1 Kg].
% Theta_0:   Ángulo inicial.
%            [Este parámetro es opcional, el default es Pi/6 rad].
% Omega_0:   Velocidad angular inicial.
%            [Este parámetro es opcional, el default es 0 rad/s].
%
% Salidas:
% --------
% t:         Vector con los valores de la variable independiente donde se
%            aproximó las m funciones solución del sistema en el intervalo.
% x:         Matriz de dimensión Nx2, donde N es la cantidad de puntos
%            de aproximación de las funciones dentro del intervalo los
%            valores de las filas corresponden a las aproximaciones de las
%            funciones solución en los valores de la variable independiente
%            correspondientes al mismo índice en el array t.
%
function [t, x, s] = pendulum(tend, h, b, l, m, Theta_0, Omega_0)

if (nargin < 2)
    fprintf(strjoin({'\nError: ',...
        'Insuficientes ', ...
        'parámetros.\n'}, ''));
    t = [];
    x = [];
    s = 0;
    return;
end %if

if (tend < 0)
```

```
    fprintf(strjoin({'\nError: ',...
        'Tiempo final ', ...
        'inválido.\n'}, ''));
    t = [];
    x = [];
    s = 0;
    return;
end %if

if (h >= tend/10) || (h <= 0)
    fprintf(strjoin({'\nError: ',...
        'Paso ',...
        'inválido.\n'}, ''));
    t = [];
    x = [];
    s = 0;
    return;
end %if

if (nargin < 3)                   % Aseguro de tener el valor si no se dió.
    b = 0;                        % Valor por omisión.
end %if

if (nargin < 4)                   % Aseguro de tener el valor si no se dió.
    l = 1;                        % Valor por omisión.
end %if

if (nargin < 5)                   % Aseguro de tener el valor si no se dió.
    m = 1;                        % Valor por omisión.
end %if

if (nargin < 6)                   % Aseguro de tener el valor si no se dió.
    Theta_0 = Pi/6;               % Valor por omisión.
end %if

if (nargin < 7)                   % Aseguro de tener el valor si no se dió.
    Omega_0 = 0;                  % Valor por omisión.
end %if


if isnan(h) || isnan(b) || isnan(l) || isnan(m) || isnan(Theta_0) || ...
        isnan(Omega_0) || (b < 0) || (l <= 0) || (m <= 0) || ...
        ((Theta_0 == 0) && (Omega_0 == 0))
    fprintf(strjoin({'\nError: ',...
        'Parámetros del péndulo ',...
        'inválidos.\n'}, ''));
```

```
    t = [];
    x = [];
    s = 0;
    return;
end %if


g = 9.81;                          % Aceleración de la gravedad.


f = @(t,x) [x(2); ...              % Sistema de dos ecuaciones del péndulo.
    -(b/m)*x(2)- ...
    (g/l)*sin(x(1))];

[t, x] = rk4(f, [0 tend + h],...  % Llamo a Runge-Kutta de curato orden.
    [Theta_0, Omega_0], h);

s = 1;

end % function
```

A.2.3. rk2.m

```
% Implementa el método de Runge-Kutta de orden 4.
%
% Parámetros:
% -----------
% f:         Puntero a la función que toma un valor escalar y devuelve un
%            array de m valores correspondientes a las m funciones solución
%            del sistema a aproximar.
% tspan:     Array de dos valores con el valor inicial y final del
%            intervalo en el cual aproximar las funciones.
% y0:        Array de m valores iniciales para las funciones.
% step:      Paso deseado para el cálculo de las aproximaciones, el valor
%            finalmente usado puede que sea menor para acomodarse al
%            intervalo de aproximación.
%
% Salidas:
% --------
% t:         Vector con los valores de la variable independiente donde se
%            aproximó las m funciones solución del sistema en el intervalo.
% y:         Matriz de dimensión Nxm, donde N es la cantidad de puntos
%            de aproximación de las funciones dentro del intervalo y m la
%            cantidad de funciones del sistema, los valores de las filas
%            corresponden a las aproximaciones de las funciones solución
%            en los valores de la variable independiente correspondientes
%            al mismo índice en el array t.
%
function [t, y] = rk4(f, tspan, y0, step)

N = ceil((tspan(2) - tspan(1))/step); % Determino la cantidad de pasos.

h = (tspan(2) - tspan(1))/N;           % Adapto el paso al intervalo.

u = zeros(1, N);                       % Inicializo la variable
                                       % que contendrá los pasos
                                       % de la variable independiente.

u(1) = tspan(1);                       % Guardo el primer valor de
                                       % la variable independiente, que
                                       % corresponde al inicio del
                                       % intervalo.

m = length(y0);                        % Obtengo la cantidad de
                                       % funciones a estimar.

z = zeros(N, m);                       % Inicializo la matriz que
```

```
                                        % contendrá las estimaciones de
                                        % las m funciones solución.

z(1, : ) = y0;                          % Guardo los valores iniciales
                                        % en la matriz.

k = zeros(m, 4);                        % Inicializo la matriz que
                                        % contendrá los k de cada
                                        % iteración.

for i = 1:N-1

    k(:, 1) = f(u(i), z(i, :));         % Calulo de la primera columna
                                        % de la matriz k, corresponde
                                        % a los k1 de las m funciones.

    k(:, 2) = f(u(i) + (1/2)*h, ...     % Calulo de la segunda columna
        z(i, :) + (1/2)*h*k(:, 1)');    % de la matriz k, corresponde
                                        % a los k2 de las m funciones.


    k(:, 3) = f((u(i) + (1/2)*h), ... % Calulo de la tercera columna
        (z(i, :) + (1/2)*h*k(:, 2)')); % de la matriz k, corresponde
                                        % a los k3 de las m funciones.

    k(:, 4) = f((u(i) + h),...          % Calulo de la cuarta columna
        (z(i, :) + k(:, 3)'*h));        % de la matriz k, corresponde
                                        % a los k4 de las m funciones.

    z(i+1, :) = z(i, :) +   ...         % Calculo el próximo valor
        (1/6)*(k(:, 1)' + ...           % de la variable independiente.
        2*k(:, 2)' + 2*k(:, 3)' + ...
        k(:, 4)')*h;

    u(i + 1) = u(1) + i*h;              % Calculo el próximo valor de la
                                        % variable independiente.
end% for

t = u;                                  % Devuelvo los valores de la
                                        % variable independiente.

y = z;                                  % Devuelvo la matriz de
                                        % estimaciones de las m
                                        % funciones solución del sistema.

end%function
```

A.2.4. plot_solution.m

```
% Grafica el ángulo y la velocidad angular.
%
% Parámetros:
% -----------
% t:         Vector con los valores de la variable independiente donde se
%            aproximó las  funciones solución del sistema en el intervalo.
% Theta:     Array de los valores de ángulo correspodientes a la variable
%            independiente,t, a graficar.
% Omega:     Array de los valores de la velocidad angular
%            correspodientes a la variable independiente,t, a graficar.
% titulo:    Título para el gráfico.
% sz_perc:   Tamaño del gráfico en porcentaje de la pantalla.
%
% Salidas:
% --------
% graphic_handle: Puntero al gráfico.
%
function [graphic_handle] = plot_solution(t, Theta, Omega, titulo, sz_perc)

if (nargin < 5)                  % Aseguro de tener un tamaño.
    sz_perc = 50;                % Valor por omisión.
end %if

if (sz_perc < 10.0)              % Ajusto el tamaño de salida.
    sz_perc = 10.0;
elseif (sz_perc > 100.0)
    sz_perc = 100.0;
end % if

% Determino si estoy trabajando en MATLAB u Octave.
Is_Octave = (5 == exist('OCTAVE_VERSION', 'builtin'));

% Calculo el tamaño y la posición de la imagen.
pict_size = sz_perc/100;
pict_pos = (1 - pict_size)/2;

% Genero la figura, a un % del tamaño de la panatalla y centrada.
% No parece funcionar en Octave, pero no genera errores tampoco.
figure1 = figure('units', 'normalized', 'outerposition', ...
    [pict_pos pict_pos pict_size pict_size]);

% Seteo el color de fondo para el gráfico.
set(figure1, 'Color', [1 1 1]);

```

```
% Creo los ejes.
axes1 = axes('Parent', figure1);

% Activo eje izquierdo.
%yyaxis(axes1, 'left');

% Creo el gráfico de Theta.
% plot(axes1, t, Theta, 'Color', [1 0 0]);

% Grafico los datos.
[yyax, h1, h2] = plotyy(axes1, t, Theta, t, Omega);

% Seteo el color de los ejes de ordenadas.
set(yyax, {'ycolor'}, {[1 0 0]; [0 0 1]})

% Seteo el color de los gráficos.
set(h1, 'color', [1 0 0]);
set(h2, 'color', [0 0 1]);

% Seteo el label para y.
ylabel(yyax(1), '\Theta [rad]');

% Seteo el label para x.
xlabel(axes1, 'Tiempo [s]');

% Seteo el título.
title(axes1, titulo);

% Octave no soporta labels inclinados.
if (~Is_Octave)
    % Labels inclinados.
    xtickangle(axes1, 75);
end % if

% Límites para las abcisas.
xlim(axes1, [t(1) t(length(t))]);

% Determino los límites.
maxy = max([abs(min(Theta)) abs(max(Theta))]);

% Ticks para el eje y izquierdo.
yticks1 = (-1.1*maxy : 1.1*maxy/5 : 1.1*maxy);

% Límites para el eje y izquierdo.
ylim(yyax(1), [yticks1(1) yticks1(length(yticks1))]);

```

```
% Muestro la "caja" que contiene al gráfico.
box(axes1,'on');

% Seteo las propiedades del eje de abcisas.
set(axes1, 'FontSize',14, 'XGrid','on','XMinorTick','on','XTick',...
    (t(1): 0.5: t(length(t))),...
    'TickLabelInterpreter','tex');

% Seteo las propiedades del eje de ordenadas 1.
set(yyax(1), 'FontSize',14, 'XGrid','on','XMinorTick','on','XTick',...
    (t(1): 0.5: t(length(t))),...
    'YGrid','on','YMinorTick','on','YTick',...
    yticks1, 'TickLabelInterpreter','tex');

% Seteo el label para y.
ylabel(yyax(2), '\Omega [rad/s]');

% Determino los límites.
maxy = max([abs(min(Omega)) abs(max(Omega))]);

% Ticks para el eje y izquierdo.
yticks2 = (-1.1*maxy : 1.1*maxy/5 : 1.1*maxy);

% Límites para el eje y izquierdo.
ylim(yyax(2), [yticks2(1) yticks2(length(yticks2))]);

% Seteo las propiedades del eje de ordenadas 2.
set(yyax(2), 'FontSize',14,...
    'YGrid','on','YMinorTick','on','YTick',...
    yticks2, 'TickLabelInterpreter','tex');

% Fuerzo a que se muestre al gráfico de inmediato.
drawnow;

% Asigno el valor del handle del gráfico que devuelvo.
graphic_handle = figure1;
```

A.2.5. romberg.m

```
% Implementa el método de Romberg.
%
% Parámetros:
% -----------
% f:         Puntero a la función que cuya integral se desea aproximar.
% a:         Inicio del intervalo de integración.
% b:         Fin del intervalo de integración.
% p:         Número de filas.
% Salidas:
% --------
% I:         Valor de la integral aproximada.
function I = romberg(f, a, b, p)

I = zeros(p, p);
for k=1:p
    % llamo al método de los trapecios para n = 2^k.
    I(k,1) = trapezcomp(f, a, b, 2^(k-1));

    % Fórmula recursiva de Romberg.
    for j=1:k-1
        I(k,j+1) = (4^j * I(k,j) - I(k-1,j)) / (4^j - 1);
    end

end

end
```

A.2.6. trapezcomp.m

```
% Implementa el método de Trapecios.
%
% Parámetros:
% -----------
% f:         Puntero a la función que cuya integral se desea aproximar.
% a:         Inicio del intervalo de integración.
% b:         Fin del intervalo de integración.
% n:         Número de paneles.
% Salidas:
% --------
% In:        Valor de la integral aproximada.
function In = trapezcomp(f, a, b, n)

% Incialización.
h = (b-a)/n;
x = a;

%Regla compuesta.
In =f(a);
for k=2:n
    x  = x + h;
    In = In + 2. * f(x);
end
In = (In + f(b)) * h * 0.5;

end
```

B. Captura de la salida

B.1. Consideraciones para el código

El texto corresponde a la captura que se hace desde el script de **MATLAB**, la codificación es automáticamente convertida por LaTeX usando el package **"listingsutf8"**.

B.2. Archivo de captura de la salida

B.2.1. salida.txt

```
Inicializando las variables globales para el TP2...Listo

Trabajando en: MATLAB 9.7.0.1190202 (R2019b) sobre Windows.



Creando el directorio para las imágenes...Listo

Creando el directorio para los resultados numéricos...Listo

Calculando la estimación de la solución del sistema con m = 1Kg, l = 1m, b = 0Ns/m, h = 0.2s, Theta0 = pi/6, Omega0

Salvando los resultados en un archivo "CSV"......Listo

Generando un gráfico de la solución...Listo

Salvando el gráfico en un archivo "PNG"......Listo

Calculando la integral del módulo de la posición......Listo

El valor de la integral aproximada con Romberg del módulo de la posición es: 6.5419117322670095.

Calculando la estimación de la solución del sistema con m = 1Kg, l = 1m, b = 0.5Ns/m, h = 0.2s, Theta0 = pi/6, Omega

Salvando los resultados en un archivo "CSV"......Listo

Generando un gráfico de la solución...Listo

Salvando el gráfico en un archivo "PNG"......Listo

Calculando la integral del módulo de la posición......Listo

El valor de la integral aproximada con Romberg del módulo de la posición es: 2.7320644496497102.

Calculando la estimación de la solución del sistema...Listo

Generando un gráfico de la solución...Listo

Calculando la integral del módulo de la posición......Listo

El valor de la integral aproximada con Romberg del módulo de la posición es: 2.7378140579622841.
```

```
Ejecución del TP2 terminada.
```