FACULTAD DE INGENIERIA

Universidad de Buenos Aires

Análisis numérico I - 75.12/95.04

Trabajo práctico N° 1

Raíces de ecuaciones

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

10 de Octubre de 2019

Índice

Índice de figuras

Índice de cuadros

1. Enunciado

1.1. Resumen enunciado

Este TP consiste en realizar un pequeño estudio del movimiento de un ascensor, figura 1.1, intentando utilizar valores realistas para los parámetros a fin de obtener las expresiones para la posición, $x_{(t)}$, la velocidad, $v_{(t)}$ y la aceleración, $a_{(t)}$. Luego estas expresiones se utilizarán para aplicar un método numérico de nuestra implementación, Newton-Raphson en este caso.

1.2. Planteo

El planteo implica asumir ciertas cosas acerca del ascensor, realizando el análisis para el movimiento ascendente, en particular se debe analizar, la altura de un piso, que asumimos ser igual para todos los pisos, siendo en todos por lo tanto el mismo problema, la masa de la cabina y los pasajeros, el tiempo de viaje máximo (se da a máxima carga), la máxima aceleración, y la máxima velocidad, asumimos también por simplicidad que la fuerza que se imprime a la cabina varía linealmente con el tiempo, esto es planteado así en el enunciado. Todas estos parámetros se obtuvieron de normas locales,[5], e internacionales, [6], y de algunos ejemplos de ascensores residenciales comerciales.

En general las normas especifican para la máxima aceleración posible $8\text{m}/\text{s}^2$, la masa de cada persona debe tomarse en promedio como de 75kg, la masa de la cabina depende del ascensor en cuestión, se tomó un modelo para 12 personas, con una cabina de masa igual a 450kg, esto determina la masa máxima de carga en 1350kg y la mínima en los 450kg de la cabina vacía. Se toman $t_f = 3\text{s}$ como el máximo tiempo de tránsito entre pisos, caso que se daría a máxima carga, y se toma la altura entre pisos como, $h_f = 4\text{m}$, el cual es un valor posible en ascensores residenciales, dentro de todo un rango posible.
Otras condiciones a tener en cuenta en el planteo, son que se parte de la posición inicial $x_{(0)} = 0\text{m}$, se llega a la posición final, $x_{(t_f)} = h_f$ y además para la velocidad debe ser, $v_{(0)} = 0\text{m/s}$ y $v_{(t_f)} = 0\text{m/s}$

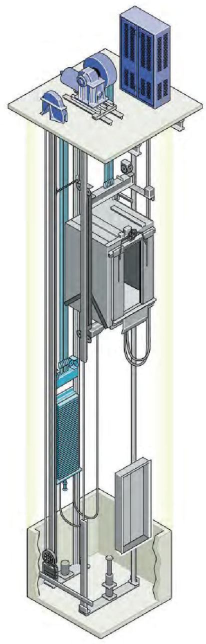

Figura 1.1: Ascensor electromecánico.

De la física del problema se obtienen las expresiones, (1.1), (1.2) y (1.3), para la posición, velocidad y aceleración, respectivamente.

$$x_{(t)} = A \cdot t^3 + B \cdot t^2 + C \cdot t + D \tag{1.1}$$

$$v_{(t)} = 3 \cdot A \cdot t^2 + 2 \cdot B \cdot t + C \tag{1.2}$$

$$a_{(t)} = 6 \cdot A \cdot t + 2 \cdot B \cdot \tag{1.3}$$

Las constantes A, B, C y D se obtienen de las condiciones y parámetros mencionados anteriormente para el caso de máxima carga, $n_{personas} = 12$, para el caso extremo de mínima carga se asigna la máxima aceleración y se ajusta el tiempo de transito e iterativamente se obtienen las expresiones correspondientes a media carga respetando los parámetros elegidos. Planteando para media carga, se obtiene el siguiente sistema de ecuaciones:

$$\begin{cases} 4 = & 27A + 9B \\ 0 = & 27A + 6B \\ 0 = & C \\ 0 = & D \end{cases} \tag{1.4}$$

Se obtiene $t_f = 3\text{s}$, $A = -0.2963\text{m/s}^3$ y $B = 1.3333\text{m/s}^2$, con lo que nos queda:

$$\boxed{\begin{cases} x_{cmax(t)} = & -0.2963\text{m/s}^3 \cdot t^3 + 1.3333\text{m/s}^2 \cdot t^2 \\ v_{cmax(t)} = & -0.8889\text{m/s}^3 \cdot t^2 + 2.6667\text{m/s}^2 \cdot t \\ a_{cmax(t)} = & -1.7778\text{m/s}^3 \cdot t + 2.6667\text{m/s}^2 \end{cases}} \tag{1.5}$$

Para el caso de mínima carga, $n_{personas} = 0$ (con la cabina vacía), asumiendo que la fuerza inicial es la misma y que la máxima aceleración debe ser de 8m/s^2, ahora para una masa total de 450kg, tenemos:

$$\begin{cases} 4 = & A \cdot t_f^3 + 4 \cdot t_f^2 \\ 0 = & 3A \cdot t_f^2 + 8 \cdot t_f \end{cases} \tag{1.6}$$

Se obtiene $A = -1.5396\text{m/s}^3$ y $t_f = 1.7321\text{s}$, con lo que nos queda:

$$\boxed{\begin{cases} x_{cmin(t)} = & -1.5396\text{m/s}^3 \cdot t^3 + 4\text{m/s}^2 \cdot t^2 \\ v_{cmin(t)} = & -4.6188\text{m/s}^3 \cdot t^2 + 8\text{m/s}^2 \cdot t \\ a_{cmin(t)} = & -9.2376\text{m/s}^3 \cdot t + 8\text{m/s}^2 \end{cases}} \tag{1.7}$$

Por último para el caso de carga media, $n_{personas} = 6$, nuevamente con la misma fuerza inicial y ahora con una masa total de 900kg, tenemos:

$$\begin{cases} 4 = & A \cdot t_f^3 + 2 \cdot t_f^2 \\ 0 = & 3A \cdot t_f^2 + 4 \cdot t_f \end{cases} \tag{1.8}$$

Se obtiene $A = -0.5443\text{m/s}^3$ y $t_f = 2.4495\text{s}$, con lo que nos queda:

$$\boxed{\begin{cases} x_{cmin(t)} = & -0.5443\text{m/s}^3 \cdot t^3 + 2\text{m/s}^2 \cdot t^2 \\ v_{cmin(t)} = & -1.6329\text{m/s}^3 \cdot t^2 + 4\text{m/s}^2 \cdot t \\ a_{cmin(t)} = & -3.2658\text{m/s}^3 \cdot t + 4\text{m/s}^2 \end{cases}} \tag{1.9}$$

1.3. Planteo extra - Dimensionamiento de los frenos del ascensor

Para el dimensionamiento del frenado del ascensor, partimos de suponer que el ascensor cae de la posición de reposo, moviéndose en caída libre, $a_{(t)} = g$, durante los 0.8s que le toma al sistema detectar el fallo, y dimensionando el frenado de tal forma de limitar la aceleración, el sistema solo logra detener totalmente el ascensor si este se encuentra a una altura mínima determinada, el sistema se completa con un sistema de topes hidráulicos o buffers, figura 1.2, que absorben la energía restante cuando la cabina del ascensor los golpea, este tipo de buffers son obligatorios en ascensores que desarrollan velocidades que cruzan un umbral establecido por normas.

Figura 1.2: Buffers hidráulicos.

Planteamos además la condición que el ascensor tarde 1.6s en detener la cabina, valor que permite lograr una aceleración máxima aceptable, también se plantea a máxima carga y que el frenado se hará a aceleración lineal. Con estás consideraciones, se tiene que se alcanza una velocidad de caída de -7.84meter/s antes de que se accione el sistema de frenado y cayendo en ese lapso 3.136m, estos valores permitirán diseñar el buffer hidráulico para el peor caso. Luego con estos valores y la condición de aceleración se plantea:

$$\begin{cases} a_{f(t)} = & A \cdot t + B \\ v_{f(t)} = & \frac{A}{2} \cdot t^2 + B \cdot t + C \\ x_{f(t)} = & \frac{A}{6} \cdot t^3 + \frac{B}{2} \cdot t^2 + C \cdot t + D \\ v_{(t_f)} = & 0 \\ a_{(t_f)} = & g \\ t_f = 1.6\mathrm{s} \end{cases} \tag{1.10}$$

De donde obtenemos:

$$\begin{cases} x_{cmin(t)} = & 1.0203\mathrm{m/s^3} \cdot t^3 - 7.84\mathrm{m/s^2} \cdot t - 3.136\mathrm{m} \\ v_{cmin(t)} = & 3.0625\mathrm{m/s^3} \cdot t^2 7.84\mathrm{m/s^2} \\ a_{cmin(t)} = & 6.125\mathrm{m/s^3} \cdot t \end{cases} \tag{1.11}$$

Donde la aceleración lineal implica una fuerza lineal por parte del motor para lograr este frenado.

1.4. Resolución del problema numérico

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. Todas las salidas numéricas se guardaron en archivos que luego fueron leídas en LaTeX usando paquetes para el proceso de archivos en formato **"CSV"**, los mismos permiten redondeo, presentación y hasta algunas operaciones sobre los datos, lo cual facilita el escribir el informe de tal manera de que no deba modificarse al modificar los datos, basta con compilar nuevamente. Las imágenes en forma similar, se guardaron por código desde **MATLAB** en formato **"PNG"** y luego se incorporaron desde LaTeX. Algo a mencionar es que se estimaron lo valores del orden de convergencia y la constante asintótica, para cada uno de el método numérico implementado.

2. Gráficos de las funciones obtenidas para media carga

A continuación se muestran las gráficas de las funciones de posición, velocidad y aceleración para media carga, $n_{personas} = 6$.

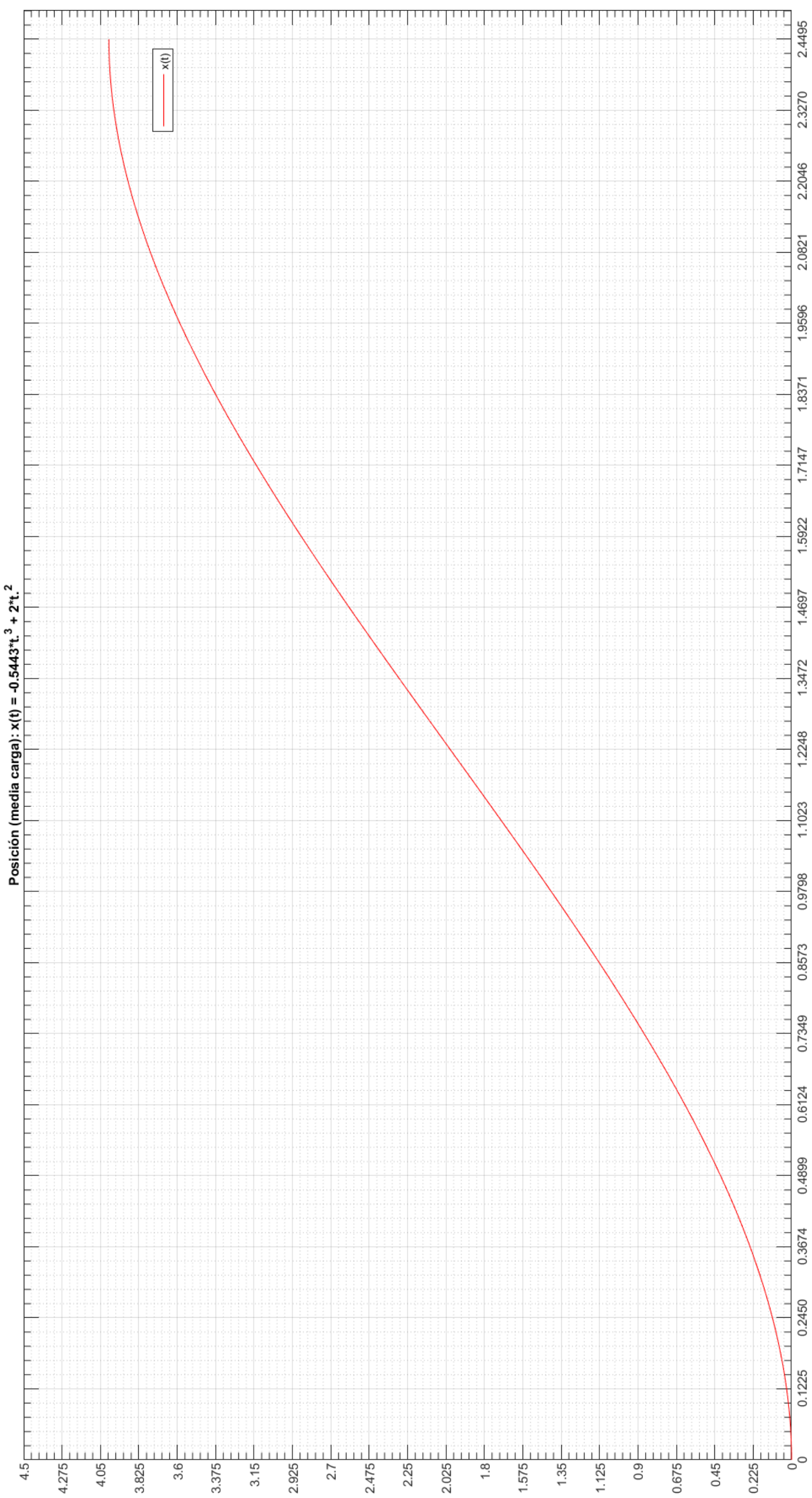

Figura 2.3: Posición en función del tiempo a media carga.

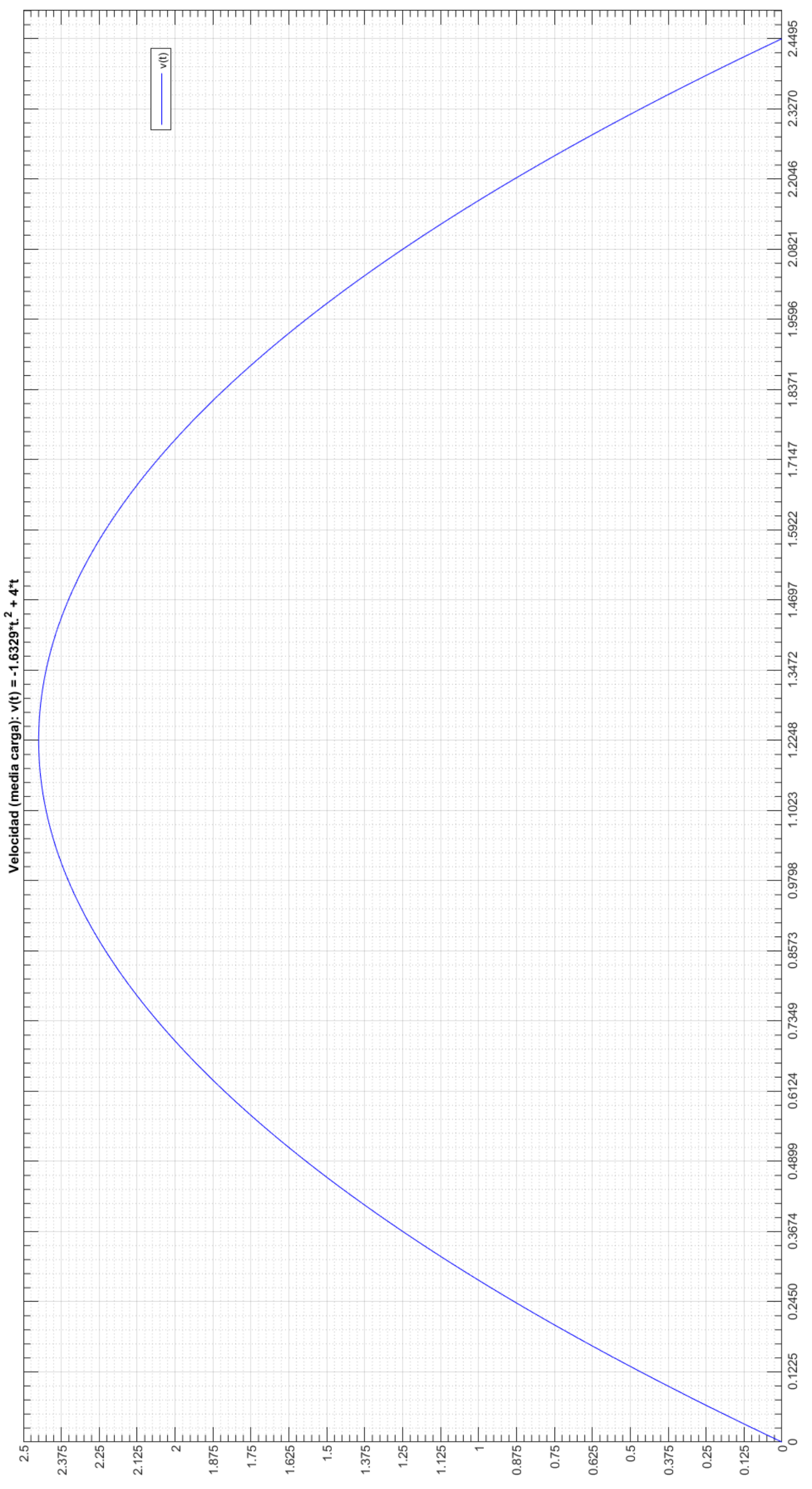

Figura 2.4: Valocidad en función del tiempo a media carga.

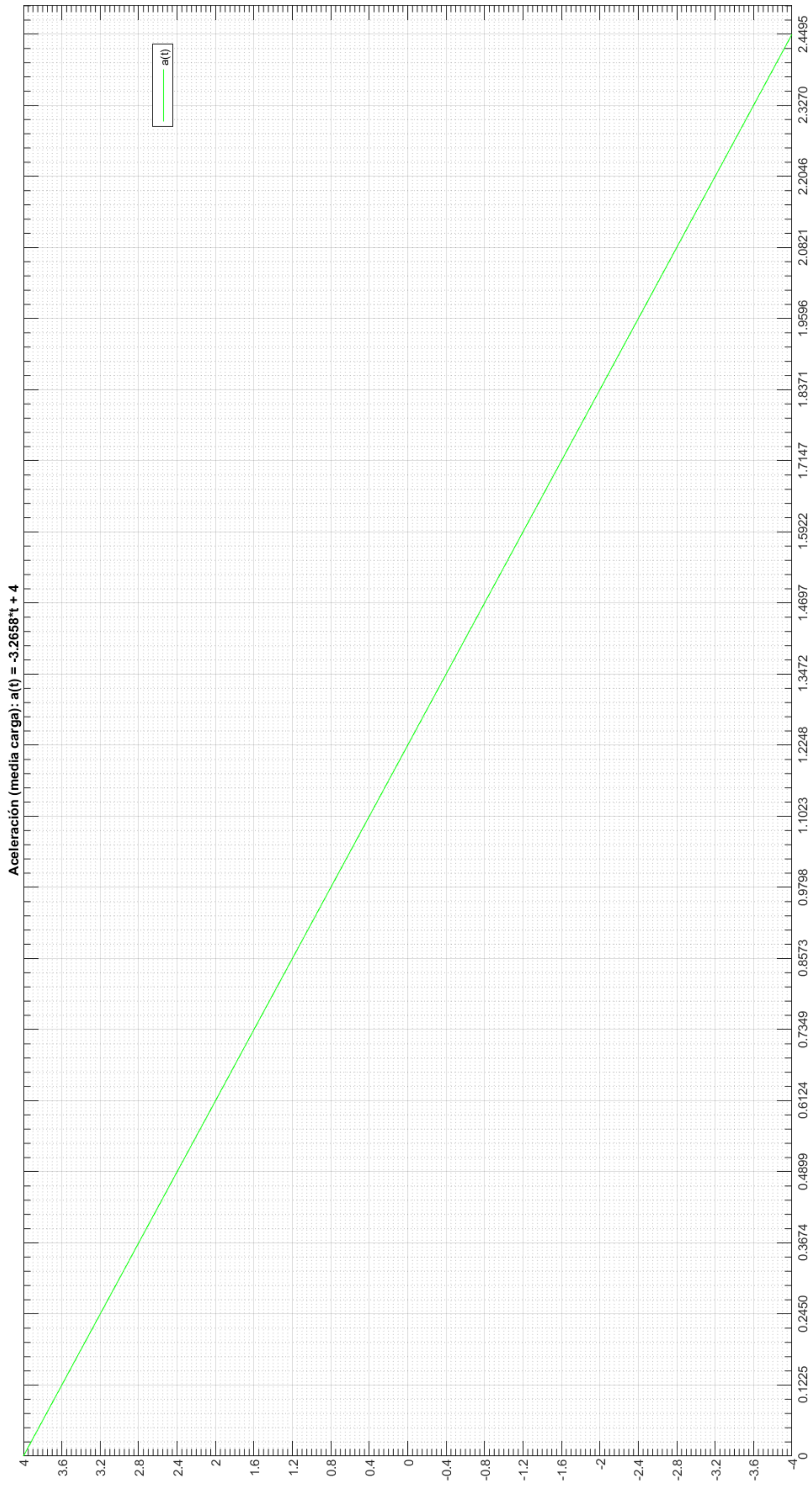

Figura 2.5: Aceleración en función del tiempo a media carga.

3. Implementación de los algoritmos

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

“tp1.m” (apéndice A.2.1): Script principal que se debe ejecutar para realizar todos los cálculos y generar los archivos de resultados.

“grafico.m” (apéndice A.2.2): Función que grafica una función entre dos valores dados.

“estimate_order.m” (apéndice A.2.3): Función que, dados las 4 últimas estimaciones de una raíz/-solución de un método numérico, estima el orden de convergencia y la constante asintótica del método, la estimación es mejor cuanto mayor es la precisión de las estimaciones usadas.

“method_newton.m” (apéndice A.2.4): Función que implementa el algoritmo del método de Newton-Raphson.

“method_bisection.m” (apéndice A.2.5): Función que implementa el algoritmo del método de bisección usado como arranque.

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** u **Octave** al ejecutar el script principal.

3.1. 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.

4. Método de Newton-Raphson

Para aproximar el valor de la raíz de $f(t)$ se plantea el método iterativo con la función:

$$g_{(t)} = t - \frac{f(t)}{f'(t)}$$

4.1. Demostración de la convergencia de la iteración a la raíz de $f(x)$

Para ver que el método de Newton-Raphson converge a la raíz del polinomio, dado que se elige una aproximación inicial lo suficientemente cercana, basta ver que se cumplan las hipótesis del teorema correspondiente, esto es, que la función sea derivable, esto es $f(t) \in \mathscr{C}^2$ y que la función derivada no se anule en la raíz, pero dado que nuestra función es un polinomio, se tiene que:

$$f(t) \in \mathscr{C}^\infty \Rightarrow f(t) \in \mathscr{C}^2$$

Por lo tanto la función es derivable, además se comprueba que su derivada no tiene raíces reales, por lo que nunca se anula para $t \in \mathbb{R}$, por lo tanto no se anula en particular para el intervalo de interés, $[0, t_f]$. Tenemos entonces que se cumplen las hipótesis del teorema, con lo que se puede garantizar que de elegir una aproximación inicial lo suficientemente cercana, el método convergerá a la raíz de $f(t)$, en la práctica simplemente se arranca el método de algún punto en el intervalo que contiene a la raíz y si no convergiese, bastaría con hacer algunos pasos de bisección para acercarse mas a la raíz buscada.

4.2. Orden de convergencia para el método de Newton-Raphson

α : orden de convergencia

λ : constante asintótica

Utilizamos las 4 últimas iteraciones extraídas de la tabla 4.1 para calcular aproximadamente el orden de convergencia y la constante asintótica:

$$\frac{|p_4 - p|}{|p_3 - p|^{\alpha}} \cong \frac{|p_4 - p_3|}{|p_3 - p_2|^{\alpha}} \cong \lambda \tag{4.1}$$

$$\frac{|p_3 - p|}{|p_2 - p|^{\alpha}} \cong \frac{|p_3 - p_2|}{|p_2 - p_1|^{\alpha}} \cong \lambda \tag{4.2}$$

Igualando las ecuaciones (4.1) y (4.2)

$$\frac{|p_3-p_2|}{|p_2-p_1|^{\alpha}} \cong \frac{|p_4-p_3|}{|p_3-p_2|^{\alpha}} \Rightarrow \left(\frac{|p_3-p_2|}{|p_2-p_1|}\right)^{\alpha} \cong \frac{|p_4-p_3|}{|p_3-p_2|} \Rightarrow \alpha \ln\left(\frac{|p_3-p_2|}{|p_2-p_1|}\right) \cong \ln\left(\frac{|p_4-p_3|}{|p_3-p_2|}\right) \Rightarrow$$

$$\Rightarrow \alpha \cong \frac{\ln\left(\frac{|p_4-p_3|}{|p_3-p_2|}\right)}{\ln\left(\frac{|p_3-p_2|}{|p_2-p_1|}\right)} \Rightarrow \alpha \cong \frac{\ln\left(\frac{0.0000000000000411}{0.0000000000000412}\right)}{\ln\left(\frac{0.0000000000000412}{0.0000000000000413}\right)} \cong \frac{0.0000000000000414}{0.0000000000000415} = 1.999900522446628 \cong 2$$

$$\Rightarrow \alpha \cong 2$$

Experimentalmente verificamos que el método de Newton-Raphson tiene orden de convergencia igual a 2. Reemplazamos en la ecuación (4.1) el valor de α calculado y determinamos el valor de la constante asintótica λ:

$$\lambda \cong \frac{|p_4 - p_3|}{|p_3 - p_2|^{\alpha}} \cong \frac{0.0000000000000411}{0.0000000000000416} \cong 0.268958171377874 \Rightarrow \lambda \cong 0.269$$

4.3. Iteración de Newton-Raphson

$$\begin{cases} p_{(n)} = p_{(n-1)} - \frac{f(p_{(n-1)})}{f'(p_{(n-1)})} \\ p_{(0)} = seed \end{cases} \tag{4.3}$$

Con $f_{(t)} = x_{(t)} - x_{(t_{a_{30\,\%}})}$

La expresión (4.3) representa el algoritmo implementado en **MATLAB** con el cual se obtuvo las tablas de la sección 4.4.

4.4. Salidas del algoritmo de Newton-Raphson para las funciones pedidas

Tolerancia pedida: 1×10^{-10}			
Iteración	Valor	Delta	Error relativo (%)
1	0.858240025980884	0.060322474019116	7.03
2	0.857370525910176	0.000869500070708	1.01×10^{-1}
3	0.857370322738697	0.000000203171480	2.37×10^{-5}
4	0.857370322738686	0.000000000000011	1.30×10^{-12}

Cuadro 4.1: Tabla de salida para el algoritmo de Newton-Raphson para la 1° función.

Resultado final para la solución, con una tolerancia de 1×10^{-10}, hallada después de 4 iteraciones:

$$\boxed{p = 0.8573703227 \pm 1 \times 10^{-10}}$$

Tolerancia pedida: 1×10^{-10}			
Iteración	Valor	Delta	Error relativo (%)
1	1.053151532920373	0.071848467079627	6.82
2	1.050005938275396	0.003145594644976	3.00×10^{-1}
3	1.050000000021146	0.000005938254251	5.66×10^{-4}
4	1.050000000000000	0.000000000021146	2.01×10^{-9}

Cuadro 4.2: Tabla de salida para el algoritmo de Newton-Raphson para la 2° función.

Resultado final para la solución, con una tolerancia de 1×10^{-10}, hallada después de 4 iteraciones:

$$p = 1.0500000000 \pm 1 \times 10^{-10}$$

Tolerancia pedida: 1×10^{-10}			
Iteración	Valor	Delta	Error relativo (%)
1	0.606834382915884	0.042703117084116	7.04
2	0.606218209598678	0.000616173317205	1.02×10^{-1}
3	0.606218065298354	0.000000144300324	2.38×10^{-5}
4	0.606218065298346	0.000000000000008	1.32×10^{-12}

Cuadro 4.3: Tabla de salida para el algoritmo de Newton-Raphson para la 3° función.

Resultado final para la solución, con una tolerancia de 1×10^{-10}, hallada después de 4 iteraciones:

$$p = 0.6062180653 \pm 1 \times 10^{-10}$$

Tolerancia pedida: 1×10^{-10}			
Iteración	Valor	Delta	Error relativo (%)
1	1.606428571428571	0.143571428571429	8.94
2	1.600012862859684	0.006415708568888	4.01×10^{-1}
3	1.600000000051704	0.000012862807980	8.04×10^{-4}
4	1.600000000000000	0.000000000051704	3.23×10^{-9}

Cuadro 4.4: Tabla de salida para el algoritmo de Newton-Raphson para la 4° función.

Resultado final para la solución, con una tolerancia de 1×10^{-10}, hallada después de 4 iteraciones:

$$p = 1.6000000000 \pm 1 \times 10^{-10}$$

5. Observaciones y conclusiones

Los métodos numéricos implementados en el caso propuesto tienen convergencias que en general coinciden con lo esperado en forma teórica, estos ordenes de convergencia y las correspondientes constantes asintóticas fueron estimadas y comparadas con los valores teóricos, para los casos donde hay una expresión disponible. Otra cosa que se puede analizar, es que las tolerancias pedidas tienen dos órdenes de magnitud de diferencia respecto a la anterior, con lo que se puede apreciar que los algoritmos que tienen convergencia lineal aproximadamente duplican las iteraciones necesarias para alcanzar la siguiente tolerancia, en cambio Newton-Raphson, que tiene convergencia cuadrática, solo incrementa en aproximadamente una iteración para alcanzarla.

6. Bibliografía

Referencias

[1] *Numerical Analysis (9th Edition)*
Author: Richard L. Burden
Author: J. Douglas Faires
Publisher: Brooks/Cole, Cengage Learning; 9th Edition (2011)
Copyright: © 2011, 2005, 2001, Brooks/Cole, Cengage Learning.
ISBN 13: 978-0-538-73351-9
ISBN 10: 0-538-73351-9
Website: https://www.cengagebrain.com.mx/shop

[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

[5] *Leyes y Normativas de ascensores en CABA*
Website: http://www.camaradeascensores.com.ar/index.php/leyes-y-normativas

[6] *Elevator World*
Website: https://www.elevatorworld.com/

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. tp1.m

```
% Implementa el TP1, declarando la variables necesarias y llamando a los
% respectivos scripts. Los gráficos son salvados en formato PNG en el
% correspodiente 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 TP1...');

x = cell(1, 3); % Punteros a función de posición para n/2, n y o pers.
v = cell(1, 3); % Punteros a función de velocidad para n/2, n y o pers.
a = cell(1, 3); % Punteros a función de aceleración para n/2, n y o pers.
f = cell(1, 3); % Punteros a función para aplicar N-R para n/2, n y o pers.
```

```
fn = cell(1, 3);% Nonmbres de las funciones.

% Declaro las funciones para media carga del ascensor.
x{1} = @(t) -0.5443*t.^3 + 2.*t.^2;
v{1} = @(t) -1.6329*t.^2 + 4*t;
a{1} = @(t) -3.2658*t + 4;

% Declaro las funciones para máxima carga del ascensor.
x{2} = @(t) 0.2963*t.^3 + 1.3333.*t.^2;
v{2} = @(t) -0.8889*t.^2 + 2.6667*t;
a{2} = @(t) -1.7778*t + 2.6667;

% Declaro las funciones para mínima carga del ascensor.
x{3} = @(t) -1.5396*t.^3 + 4*t.^2;
v{3} = @(t) -4.6188*t.^2 + 8*t;
a{3} = @(t) -9.2376*t + 8;

% Declaro las funciones para el freno del ascensor.
x{4} = @(t) 1.0208*t.^3 - 7.84*t - 3.136;
v{4} = @(t) 3.0625*t.^2 - 7.84;
a{4} = @(t) 6.125*t;


fn{1} = 'Media carga (6 personas)';
fn{2} = 'Máxima carga (12 personas)';
fn{3} = 'Mínima carga (0 personas)';

% Declaro un título para los gráficos de las funciones.
titlex = 'Posición (media carga): x(t) = -0.5443*t.^3 + 2*t.^2';
legendx = 'x(t)';

titlev = 'Velocidad (media carga): v(t) = -1.6329*t.^2 + 4*t';
legendv = 'v(t)';

titlea = 'Aceleración (media carga): a(t) = -3.2658*t + 4';
legenda = 'a(t)';

% Declaro la cantidad de puntos para los gráficos.
cant_points = 1E6;

% En el caso de Octave, una cantidad muy grande causa problemas
if (Is_Octave)
    cant_points = 1E3;
end

% 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.
seeds = 'seeds';
newton_prefix = 'newton_fun_';
newton_results_name = 'newton_final_results';
newton_convergence_name = 'newton_convergence_results';
grafico_x = 'grafico_funcion_x.png';
grafico_v = 'grafico_funcion_v.png';
grafico_a = 'grafico_funcion_a.png';

% Declaro los intervalos para las funciones.
ai = [0, 0, 0];
bi = [2.4495, 3, 1.7321];

% Declaro la tolerancia para las raíces/soluciones.
tol = 1E-10;

% Declaro la máxima cantidad de iteraciones admisibles para los algoritmos.
max_iter = 500;

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', env, version, OS);

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

%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% En el caso de MATLAB debo verificar que el paquete simbólico esté
% disponible, para el cálculo simbólico de la derivada, en Octave debo
% además cargar el paquete.
if (~Is_Octave) % Si estoy trabajando en MATLAB.

    if (~license('test', 'symbolic_toolbox'))
        fprintf(strjoin({'\nNo se encontró', ...
            ' el toolbox simbólico, necesario para la derivada', ...
```

```
            ' simbólica en el método de Newton-Raphson,', ...
            ' no se puede continuar.\n\n'}, ''));

        return;
    end %if

else            % Si estoy trabajando en Octave.

    loaded = 0;

    packs = pkg('list');

    for jj = 1:numel(packs)

        if (strcmp('symbolic', packs{jj}.name))
            pkg('load', packs{jj}.name);
            loaded = 1;
            break;
        end %if
    end %for

    if (~loaded)
        fprintf(strjoin({'\nNo se encontró', ...
            ' el paquete simbólico, necesario para la derivada', ...
            ' simbólica en el método de Newton-Raphson,', ...
            ' no se puede continuar.\n', ...
            ' Se puede instalar desde Octave forge con:\n', ...
            ' "pkg install -forge symbolic".\n\n'}, ''));

        return;
    end %if

end %if

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

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

fprintf(strjoin({'\nLas funciones para la posición, velocidad', ...
    ' y aceleración a media carga (n/2 = 6),', ...
    ' son respectivamente:\n\n'},''));
disp(x{1});
fprintf('\n');
disp(v{1});
fprintf('\n');
disp(a{1});

fprintf(strjoin({'Las unidades para estas funciones son', ...
    ' m, m/s y m/s^2, respectivamente.\n\n\n'},''));


fprintf(strjoin({'Generando un gráfico de cada función en el',...
    ' intervalo [%.5f, %.5f] para ver su forma general...\n\n'}, ''), ...
    ai(1), bi(1));

% Genero un primer gráfico de cada función para
% ver su forma general.
graphic_x_handle = grafico(x{1}, ai(1), bi(1), cant_points, titlex, ...
    legendx, false, [1 0 0], 75);

```

```
fprintf(...
    'Salvando el gráfico de la posición en un archivo "PNG"......');

% Salvo el gráfico en un archivo.
saveas(graphic_x_handle, fullfile(images_directory, ...
    grafico_x));

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

graphic_v_handle = grafico(v{1}, ai(1), bi(1), cant_points, titlev, ...
    legendv, false, [0 0 1], 75);

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

% Salvo el gráfico en un archivo.
saveas(graphic_v_handle, fullfile(images_directory, ...
    grafico_v));

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

graphic_a_handle = grafico(a{1}, ai(1), bi(1), cant_points, titlea, ...
    legenda, false, [0 1 0], 75);

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

% Salvo el gráfico en un archivo.
saveas(graphic_a_handle, fullfile(images_directory, ...
    grafico_a));

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


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

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

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

% Genero los nombres completos de los archivos de resultados .
```

```
newton_results_complete_name = ...
    fullfile(results_directory, ...
    strjoin({newton_results_name, '.csv'}, ''));

newton_convergence_complete_name = ...
    fullfile(results_directory, ...
    strjoin({newton_convergence_name, '.csv'}, ''));

% Genero el nombre completo para el archivo de semillas.
seeds_complete_name = ...
    fullfile(results_directory, ...
    strjoin({seeds, '.csv'}, ''));

if (exist(newton_results_complete_name, 'file'))
    delete(newton_results_complete_name);
end

if (exist(newton_convergence_complete_name, 'file'))
    delete(newton_convergence_complete_name);
end

if (exist(seeds_complete_name, 'file'))
    delete(seeds_complete_name);
end


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

% Calculo la cantidad de cifras significativas para la tolerancia.
cant_signif = ceil(-log10(tol));

fprintf('\n\n\nEjecutando los métodos numéricos...\n\n\n');


t_acel_30_percent = zeros(1, 3);

for ii = 1:3

    fprintf('****************************\nCalculando para %s:\n\n', ...
        fn{ii});


    fprintf(strjoin({'Hallamos el valor de tiempo en', ...
        ' el cual se alcanza el 30%% de la aceleración positiva. \n', ...
```

```
        'Por tratarse de una función lineal', ...
        ' el valor se halla trivialmente:\n\n'},''));

    % Calculo el tiempo para un 30% de la aceleración máxima (positiva).

    max_acel = a{ii}(0);

    ac_30p = @(t) a{ii}(t) - 0.3*max_acel;

    t_acel_30_percent(ii) = fzero(ac_30p, 0);


    fprintf(strjoin({'El 30%% de la aceleración',...
        ' se alcanza en t_acel_30_percent = %.', ...
        num2str(cant_signif) ,'f.\n\n'}, ''), ...
        t_acel_30_percent(ii));


    fprintf(strjoin({'Hallamos el valor de posición en', ...
        ' el tiempo hallado anteriormente. \n', ...
        'Con este valor construimos la función (f{', num2str(ii) ,'})', ...
        ' a la cuál le hallaremos la raíz por Newton-Raphson:\n\n'},''));

    f{ii} = @(t) x{ii}(t) - x{ii}( t_acel_30_percent(ii));


    disp(f{ii});


    %     graphic_x_handle = grafico(f{1}, ai(1), bi(1), cant_points, ...
    %         'Función a la cual aplicamos Newton-Raphson', ...
    %         legendx, false, [1 0.3 1], 75);


    % Busco la raíz por el método de bisección.
    fprintf(strjoin({'Ejecutando el método de bisección'...
        ' para f{', num2str(ii),'}, como arranque,', ...
        ' para la tolerancia %.1e...'}, ''), 0.1);

    fprintf('Listo\n\n');


    [~, seed, ~] = method_bisection(f{ii}, ai(ii), bi(ii), 0.1);


    fprintf(strjoin({'La semilla a usar para la búsqueda', ...
```

```
        ' de la raíz es: %.5f\n\n'}, ''), seed);

    seeds_table = [ai(ii), bi(ii), seed];

    % Guardo el intervalo y la semilla en un archivo.
    fprintf('Salvando las semillas en un archivo "CSV"......');

    % Guardo las semillas.
    dlmwrite(seeds_complete_name, ...
        seeds_table, 'precision','%.15f', '-append');

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


    % Busco la raíz por el método de Newton-Raphson.
    fprintf(strjoin({'Ejecutando el método de Newton-Raphson'...
        ' para f{', num2str(ii), ...
        '} y para la tolerancia %.1e...'}, ''), tol);

    % Ejecuto el método.
    [r0, delta, erel, ...
        table, success] = ...
        method_newton(f{ii}, seed, tol, max_iter);

    % Chequeo que el método alcanzó la precisón pedida en menos
    % de las máximas iteraciones.
    if (~success)
        fprintf(strjoin({'\nFalló el método de Newton-Raphson', ...
            ' para la función f{%d}.\n\n'}, ''), ii);

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

    % Extraigo la cantidad de iteraciones requeridas como
    % el largo de la tabla.
    iter_req = size(table, 1);

    % Armo el string para formatear la salida.
    format_str = sprintf(...
        strjoin({'Raíz hallada después de %%d iteraciones:'...
        '%%.%df +- %%.1e\n\n'}), ...
        cant_signif);

```

```
    % Muestro el resultado.
    fprintf(format_str, iter_req, r0, tol);


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

    % Guardo la tabla de las iteraciones.
    dlmwrite(fullfile(results_directory, ...
        strjoin({newton_prefix, int2str(ii), '.csv'}, '')),  ...
        table, 'precision','%.15f');

    % Armo la tabla de resultados finales con:
    % 1 - Índice.
    % 2 - Raíz hallada.
    % 3 - Tolerancia pedida.
    % 4 - Cantidad de cifras significativas correspondiente.
    % 5 - Delta obtenido.
    % 6 - Error relativo.
    % 7 - Cantidad de iteraciones requeridas.
    results = [ii, r0, tol, cant_signif, delta, erel, iter_req];

    % Guardo el archivo de resultados finales.
    dlmwrite(newton_results_complete_name, ...
        results, 'precision','%.15f', '-append');

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

    % Calculo el orden de convergencia y la constante asintótica
    % estimadas, en base a los últimos valores de la última tabla.

    fprintf('Estimando el orden de convergencia......');

    % Invoco al cálculo.
    [order, asintconst, ~, asintconstder, interm1, interm2, ...
        interm3, interm4, interm5, interm6] = ...
        estimate_order(f{ii}, table(iter_req -3, 2), ...
        table(iter_req - 2, 2), ...
        table(iter_req - 1, 2), table(iter_req, 2));

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

    % Muestro el valor estimado.
    fprintf(strjoin({'El orden de convergencia estimado', ...
```

```
        ' para el método de Newton-Raphson es %.8f.\n\n'}, ''), order);

    % Armo el array con los valores a guardar.
    conv_results = ...
        [order, asintconst, asintconstder, interm1, interm2, ...
        interm3, interm4, interm5, interm6];

    % Guardo los resultados de la estimación en un archivo.
    fprintf(strjoin({'Salvando los resultados de la estimación', ...
        ' del orden de convergencia en un archivo "CSV"......'}, ''));

    % Guardo los valores de las estimaciones.
    dlmwrite(newton_convergence_complete_name, ...
        conv_results, 'precision','%.15f', '-append');

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

end

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

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

fprintf('****************************\nCalculando para v{4}:\n\n');

fprintf(strjoin({'Para el caso del freno del ascensor', ...
    ' usamos Newton-Raphson para calcular el tiempo en el', ...
    ' que se alcanza velocidad 0, la función utilizada es:\n\n'},''));

disp(v{4});


% Busco la raíz por el método de bisección.
fprintf(strjoin({'Ejecutando el método de bisección'...
    ' para v{4}, como arranque,', ...
    ' para la tolerancia %.1e...'}, ''), 0.3);

fprintf('Listo\n\n');


[~, seed, ~] = method_bisection(v{4}, 0, 2, 0.3);


fprintf(strjoin({'La semilla a usar para la búsqueda', ...
```

```
    ' de la raíz es: %.5f\n\n'}, ''), seed);

seeds_table = [0, 2, seed];

% Guardo el intervalo y la semilla en un archivo.
fprintf('Salvando las semillas en un archivo "CSV"......');

% Guardo las semillas.
dlmwrite(seeds_complete_name, ...
    seeds_table, 'precision','%.15f', '-append');

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


% Busco la raíz por el método de Newton-Raphson.
fprintf(strjoin({'Ejecutando el método de Newton-Raphson'...
    ' para v{4} y para la tolerancia %.1e...'}, ''), tol);

% Ejecuto el método.
[r0, delta, erel, ...
    table, success] = ...
    method_newton(v{4}, seed, tol, max_iter);

% Chequeo que el método alcanzó la precisón pedida en menos
% de las máximas iteraciones.
if (~success)
    fprintf(strjoin({'\nFalló el método de Newton-Raphson', ...
        ' para la función v{4}.\n\n'}, ''));

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

% Extraigo la cantidad de iteraciones requeridas como
% el largo de la tabla.
iter_req = size(table, 1);

% Armo el string para formatear la salida.
format_str = sprintf(...
    strjoin({'Raíz hallada después de %%d iteraciones:'...
    '%%.%df +- %%.1e\n\n'}), ...
    cant_signif);

% Muestro el resultado.
```

```
fprintf(format_str, iter_req, r0, tol);


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

% Guardo la tabla de las iteraciones.
dlmwrite(fullfile(results_directory, ...
    strjoin({newton_prefix, int2str(4), '.csv'}, '')),  ...
    table, 'precision','%.15f');

% Armo la tabla de resultados finales con:
% 1 - Índice.
% 2 - Raíz hallada.
% 3 - Tolerancia pedida.
% 4 - Cantidad de cifras significativas correspondiente.
% 5 - Delta obtenido.
% 6 - Error relativo.
% 7 - Cantidad de iteraciones requeridas.
results = [ii, r0, tol, cant_signif, delta, erel, iter_req];

% Guardo el archivo de resultados finales.
dlmwrite(newton_results_complete_name, ...
    results, 'precision','%.15f', '-append');

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

% Calculo el orden de convergencia y la constante asintótica
% estimadas, en base a los últimos valores de la última tabla.

fprintf('Estimando el orden de convergencia......');

% Invoco al cálculo.
[order, asintconst, ~, asintconstder, interm1, interm2, ...
    interm3, interm4, interm5, interm6] = ...
    estimate_order(v{4}, table(iter_req -3, 2), ...
    table(iter_req - 2, 2), ...
    table(iter_req - 1, 2), table(iter_req, 2));

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

% Muestro el valor estimado.
fprintf(strjoin({'El orden de convergencia estimado', ...
    ' para el método de Newton-Raphson es %.8f.\n\n'}, ''), order);
```

```
% Armo el array con los valores a guardar.
conv_results = ...
    [order, asintconst, asintconstder, interm1, interm2, ...
    interm3, interm4, interm5, interm6];

% Guardo los resultados de la estimación en un archivo.
fprintf(strjoin({'Salvando los resultados de la estimación', ...
    ' del orden de convergencia en un archivo "CSV"......'}, ''));

% Guardo los valores de las estimaciones.
dlmwrite(newton_convergence_complete_name, ...
    conv_results, 'precision','%.15f', '-append');

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



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

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

% El script se ejecutó correctamente..
fprintf('\n\nEjecución del TP1 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. grafico.m

```
% Grafica la función pasada por parámetro con el título dado.
%
% Parámetros:
% f:              Puntero a la función a graficar.
% a:              Inicio del intervalo.
% b:              Fin del intervalo.
% cantpuntos:     Cantidad de puntos usados.
% tit:            Título del gráfico.
% leyenda:        Leyenda en el gráfico.
% raiz:           Muestra una raíz aproximada gráficamente, o no.
% color:          Color usado para trazar el gráfico.
% size_percent:   Porcentaje del tamaño de la pantalla a ocupar.
%
% Salidas:
% graphic_handle: Puntero al gráfico.
%
function [graphic_handle] = grafico(f, a, b, cantpuntos, titulo, ...
    leyenda, raiz, color, size_percent)

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

if (nargin < 8)                     % Aseguro de tener un tamaño.
    size_percent = 50;              % Valor por omisión.
end %if

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

% Genero los puntos de la variable x.
x = linspace(a, b, cantpuntos);

% Calculo los puntos de la función.
y = f(x);

% Calculo el tamaño y la posición de la imagen.
pict_size = size_percent/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]);

% Oculto la figura.
set(figure1, 'Visible', 'off');

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

% Retengo los ejes actuales.
hold(axes1,'on');

% Grafico.
hplot = plot(x, y, 'DisplayName', leyenda,'Color', color);

% Incoropor el título.
title(titulo);

% Muestro los ejes.
box(axes1, 'on');

ylims = ylim;

% Seteo el resto de las propiedades de los ejes.
set(axes1,'FontSize',7,'XGrid','on','XMinorGrid','on','XMinorTick','on',...
    'XTick',a:(b-a)/20:b,'YTick',ylims(1):(ylims(2)-ylims(1))/20:ylims(2),...
    'YGrid','on','YMinorGrid','on','YMinorTick','on');

% Genero una leyenda para el gráfico.
legend1 = legend(axes1, 'show');

set(legend1,...
    'Position',[0.84219042650382 0.76675052301329 ...
    0.0401041662630937 0.0197300099200053]);

if (raiz)

    % Busco la raíz aproximada, como el menor valor de la
    % función en valor absoluto dentro del vector y.
    root_index = 1;
    min_y = realmax;
    for i = 1:length(y)
        val = abs(y(i));
        if (val < min_y)
            root_index = i;
            min_y = val;
```

```
        end %if
    end %for

    if (~Is_Octave) % Si estoy trabajando en MATLAB.

        %Creo un DataTip para la raíz aproximada gráficamente.

        % Genero el título para el datatip.
        strtip = 'Raíz aproximada gráficamente:\n';

        % Actualizo la figura.
        drawnow update;

        % Inicio el modo cursor.
        cursorMode = datacursormode(figure1);

        % Seteo la función customizada usada para actualizar el datatip.
        set(cursorMode, 'UpdateFcn', @myUpdateFcn, 'Enable', 'on')

        % Genero el datatip.
        hDatatip = cursorMode.createDatatip(hplot);

        % set the datatip marker appearance
        set(hDatatip, 'Marker', 'x', 'MarkerSize', 10, 'MarkerFaceColor', ...
            'none', 'MarkerEdgeColor', 'r', 'OrientationMode', 'manual')

        % Set the data-tip orientation to top-right rather than auto
        set(hDatatip,'Orientation','bottomright');

        % Genero la pocisión para el datatip.
        pos = [x(root_index) y(root_index)];

        % Muevo el datatip a la posición deseada.
        % Para Matlab R2014a y anteriores es necesario lo siguiente.
        % set(get(hDatatip, 'DataCursor'), 'DataIndex', index, ...
        %     'TargetPoint', pos)
        set(hDatatip, 'Position', pos);

        % Actualizo los cursores.
        updateDataCursors(cursorMode);

        % Finalizo el modo cursor.
        set(cursorMode, 'enable', 'off');

    else            % Si estoy trabajando en Octave.

```

```
        % Actualizo la figura.
        drawnow;

        % Marco un punto sobre el gráfico.
        plot(x(root_index), y(root_index), 'rx');
    end %if

    %Pix_SS = get(0,'screensize');

    % ScreenWidth = Pix_SS(2);
    %
    % ScreenHeight= Pix_SS(2);
    %
    % set(figure1, 'Position', ...
    %     [0, 0, round(ScreenWidth/2), round(ScreenHeight/2)]);


end

% Muestro la figura.
set(figure1, 'Visible', 'on');

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

% Función customizada para los datatips.
% Imprime un datatip customizado solo en MATLAB.
    function outText = myUpdateFcn(~, event)
        point = get(event, 'Position');

        outText = {sprintf(strtip), ...
            sprintf('X: %.6f', point(1)), ...
            sprintf('Y: %.6f', point(2))};
    end %function

end %function
```

A.2.3. estimate_order.m

```
% Aproxima el orden de convergencia y la constante asintótica para un
% método numérico de busqueda de soluciones\raíces usando las últimas
% 4 aproximaciones encontradas.
% Se devuelven también los valores intermedios de los cálculos.
%
% Parámetros:
% f:               Puntero a la función a la que se aplicó el método.
% val1-val4:       Cuatro últimas aproximaciones del método.
%
% Salidas:
% order:           Orden de convergencia estimado.
% asintconst:      Constante asintótica estimada.
% asintconstder:   Constante asintótica estimada con la derivada de f.
% asintconstder2:  Constante asintótica estimada con la derivada 2da de f.
% interm1-interm6: Valores intermedios del cálculo.
%
%
function [order, asintconst, asintconstder, asintconstder2, ...
    interm1, interm2, interm3, ...
    interm4, interm5, interm6] = estimate_order(f, val1, val2, val3, val4)

% Calcula:
%
% order =
% log(abs(val4 - val3/val3 - val2))/log(abs(val3 - val2/val2 - val1))
%
% asintconst =
% abs(val4 - val3)/abs(val3 - val2)^order
%

interm1 = val4 - val3;

interm2 = val3 - val2;

interm3 = val2 - val1;

interm4 = log(abs(interm1/interm2));

interm5 = log(abs(interm2/interm3));

order = interm4/interm5;

interm6 = abs(interm2)^(round(order*10)/10);

```

```
asintconst = abs(interm1)/interm6;

% Calcula:
%
% asintconstder = abs(f'(val4))

syms x;                             % Defino x como variable simbólica.
func = f(x);                        % Defino func como la función puntero a f.
der_func = diff(func);              % Derivada simbólica respecto de x
der2_func = diff(der_func);         % Derivada 2da simbólica respecto de x



f_deriv = ...
    matlabFunction(der_func);      % Convierto la función derivada simbólica
                                   % en una función normal de MATLAB/Octave.

f_deriv2 = ...
     matlabFunction(...
     der2_func, 'vars', x);        % Convierto la función derivada 2da
                                   % simbólica en una función normal de
                                   % MATLAB/Octave.

if isempty(symvar(der2_func))
valaux = f_deriv2();
else
valaux = f_deriv2(val4);
end


asintconstder = abs(f_deriv(val4));

asintconstder2 = valaux/(2*f_deriv(val4));

end %function
```

A.2.4. method_newton.m

```matlab
% Implementa el método de Newton-Raphson.
%
% Parámetros:
% -----------
% f:        Puntero a la función a la cual hallarle la raíz.
% x0:       Aproximación inicial, la semilla.
% delta:    Diferencia deseada entre dos aprximaciones sucesivas.
% max_iter: Cantidad máxima de iteraciones a realizar, opcional.
% showwork: Decide si se muestra las iteraciones al calcularlas, es 0 o 1,
%           opcional.
%
% Salidas:
% --------
% r0:       Valor de la raíz hallada con la precisión pedida, si se logra.
% delta_r0: Delta obtenido para la raíz hallada.
% e_rel_r0: Error relativo para la raíz hallada.
% table:    Tabla de salida con las iteraciones, contiene el número de
%           iteración, el valor de la raíz, el delta y el error relativo.
% success:  Si el método fue exitoso, es 1 en caso de éxito y 0 en caso de
%           fallar.
%
% En caso de fallo, las salidas numéricas son NaN (Not a Number).
%
function [r0, delta_r0, e_rel_r0, table, success] = ...
         method_newton(f, x0, delta, max_iter, showwork)

if (nargin < 4)                  % Aseguro de tener corte si no se dió.
     max_iter = 1000;            % Valor por omisión.
end %if

if (nargin < 5)                  % Por omisión no muestro el progreso.
      showwork = 0;
end %if

delta = abs(delta);               % Aseguro que el error sea positivo.

syms x;                           % Defino x como variable simbólica.
func = f(x);                      % Defino func como la función puntero a f.
der_func = diff(func);            % Derivada simbólica respecto de x

f_deriv = ...
      matlabFunction(der_func);   % Convierto la función derivada simbólica
                                  % en una función normal de MATLAB/Octave.

```

```
p0 = x0;                         % Guardo la semilla.

table_iter = zeros(max_iter, 4); % Inicializo la tabla de salida.
                                 % Contiene:
                                 % - Número de iteración.
                                 % - Valor estimado de la raíz.
                                 % - Error.
                                 % - Error relativo (%).

if (showwork)                    % Muestro el inicio del proceso.
    fprintf(strjoin({'\nUsando el método',...
        'de Newton-Raphson para hallar la raíz...\n\n'}));
end %if

for idx = 1 : max_iter

    p = p0 - f(p0)/f_deriv(p0);  % Calculo la próxima aproximación.

    delta_iter = abs(p - p0);    % Calculo el error.

    e_relat_iter = ...
        delta_iter*100.0/p;      % Calculo el rror relativo.

    table_iter(idx,:) = [idx; p; ...
        delta_iter; ...
        e_relat_iter];           % Guardo la iteración en la tabla.

    if (showwork)                % Muestro un mensaje si corresponde.
        fprintf(strjoin({'Iteración %d, nuevo valor de x: %.15f,', ...
            'delta: %.15f, error relativo: %.15f\n\n'}), ...
            idx, p, delta_iter, e_relat_iter);
    end %if

    if (delta_iter < delta)      % Comparo el error con el deseado.

        r0 = p;                  % Guardo el valor de la raíz.

        delta_r0 = delta_iter;   % Guardo el delta de l raíz.

        e_rel_r0 = e_relat_iter; % Guardo el error relativo de la raíz.

        table = ...
            table_iter(1:idx,:); % Asigno la tabla recortada.

        success = 1;             % Indico que el método fue exitoso.

```

```
        if (showwork)            % Muestro un mensaje si corresponde.
            fprintf(strjoin({'\nLa raíz se halló exitosamente por', ...
                'el método de Newton-Raphson,', ...
                'después de %d iteraciones.\n\n'}), idx);
        end %if

        return;                  % Salgo de la función.
    end %if

    p0 = p;                      % Guardo el valor para la próxima iter.
end %for

r0 = NaN;                        % Guardo un valor no válido
                                 % para la raíz.

delta_r0 = NaN;                  % Guardo un valor no válido para el delta.

e_rel_r0 = NaN;                  % Guardo un valor no válido
                                 % para el error relativo.

table = [];                      % Guardo una tabla vacía.

success = 0;                     % Indico que el método falló.

if (showwork)                    % Muestro un mensaje si corresponde.
    error(strjoin({'\nFalló el método de Newton-Raphson', ...
        'después de %d iteraciones.\n\n'}), max_iter);
end %if

end %for
```

A.2.5. method_bisection.m

```
% Implementa la cantidad pasos de bisección pedidos.
%
% Parámetros:
% f:          Puntero a la función a la cual hallarle el intervalo
%             para la raíz.
% a0:         Límite inferior inicial.
% b0:         Límite superiro inicial.
% delta:      Diferencia deseada entre dos aprximaciones sucesivas.
% max_iter:   Cantidad máxima de iteraciones a realizar, opcional.
%
% Salidas:
% a:          Inicio del intervalo.
% b:          Fin del intervulo.
% success:    Si el método fue exitoso, es 1 en caso de éxito y 0 en caso
%             de fallar.
%
% En caso de fallo, las salidas numéricas son NaN (Not a Number).
%
function [a, b, success] = method_bisection(f, a0, b0, delta, max_iter)

if (nargin < 5)         % Aseguro de tener corte si no se dió.
    max_iter = 1000;    % Valor por omisión.
end %if

delta = abs(delta);     % Aseguro que el error sea positivo.


if ((a0 > b0) || (f(a0)*f(b0) > 0))

    success = 0;

    b = NaN;            % Asigno un valor no válido para el fin del
    % intervalo.

    a = NaN;            % Asigno un valor no válido para el inicio del
    % intervalo.

    return;
end

a = a0;
b = b0;


```

```
for ii = 1:max_iter

    delta_iter = b - a;

    if (delta_iter < delta)       % Comparo el error con el deseado.
        break;
    end

    center = (b + a)/2;

    if (f(center)*f(a) > 0)
        a = center;
    else
        b = center;
    end

end


success = 1;            % Indico éxito.

end %function
```

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 TP1...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


Las funciones para la posición, velocidad y aceleración a media carga (n/2 = 6), son respectivamente:

    @(t)-0.5443*t.^3+2.*t.^2


    @(t)-1.6329*t.^2+4*t


    @(t)-3.2658*t+4

Las unidades para estas funciones son m, m/s y m/s^2, respectivamente.


Generando un gráfico de cada función en el intervalo [0.00000, 2.44950] para ver su forma general...

Salvando el gráfico de la posición en un archivo "PNG"......Listo

Salvando el gráfico de la velocidad en un archivo "PNG"......Listo

Salvando el gráfico de la aceleración en un archivo "PNG"......Listo

Listo





Ejecutando los métodos numéricos...


****************************
Calculando para Media carga (6 personas):

Hallamos el valor de tiempo en el cual se alcanza el 30% de la aceleración positiva.
```

```
Por tratarse de una función lineal el valor se halla trivialmente:

El 30% de la aceleración se alcanza en t_acel_30_percent = 0.8573703227.

Hallamos el valor de posición en el tiempo hallado anteriormente.
Con este valor construimos la función (f{1}) a la cuál le hallaremos la raíz por Newton-Raphson:

    @(t)x{ii}(t)-x{ii}(t_acel_30_percent(ii))

Ejecutando el método de bisección para f{1}, como arranque, para la tolerancia 1.0e-01...Listo

La semilla a usar para la búsqueda de la raíz es: 0.91856

Salvando las semillas en un archivo "CSV"......Listo

Ejecutando el método de Newton-Raphson para f{1} y para la tolerancia 1.0e-10...Listo

Raíz hallada después de 4 iteraciones: 0.8573703227 +- 1.0e-10

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

Estimando el orden de convergencia......Listo

El orden de convergencia estimado para el método de Newton-Raphson es 1.99990052.

Salvando los resultados de la estimación del orden de convergencia en un archivo "CSV"......Listo

****************************
Calculando para Máxima carga (12 personas):

Hallamos el valor de tiempo en el cual se alcanza el 30% de la aceleración positiva.
Por tratarse de una función lineal el valor se halla trivialmente:

El 30% de la aceleración se alcanza en t_acel_30_percent = 1.0500000000.

Hallamos el valor de posición en el tiempo hallado anteriormente.
Con este valor construimos la función (f{2}) a la cuál le hallaremos la raíz por Newton-Raphson:

    @(t)x{ii}(t)-x{ii}(t_acel_30_percent(ii))

Ejecutando el método de bisección para f{2}, como arranque, para la tolerancia 1.0e-01...Listo

La semilla a usar para la búsqueda de la raíz es: 1.12500

Salvando las semillas en un archivo "CSV"......Listo
```

```
Ejecutando el método de Newton-Raphson para f{2} y para la tolerancia 1.0e-10...Listo

Raíz hallada después de 4 iteraciones: 1.0500000000 +- 1.0e-10

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

Estimando el orden de convergencia......Listo

El orden de convergencia estimado para el método de Newton-Raphson es 2.00012932.

Salvando los resultados de la estimación del orden de convergencia en un archivo "CSV"......Listo

****************************
Calculando para Mínima carga (0 personas):

Hallamos el valor de tiempo en el cual se alcanza el 30% de la aceleración positiva.
Por tratarse de una función lineal el valor se halla trivialmente:

El 30% de la aceleración se alcanza en t_acel_30_percent = 0.6062180653.

Hallamos el valor de posición en el tiempo hallado anteriormente.
Con este valor construimos la función (f{3}) a la cuál le hallaremos la raíz por Newton-Raphson:

    @(t)x{ii}(t)-x{ii}(t_acel_30_percent(ii))

Ejecutando el método de bisección para f{3}, como arranque, para la tolerancia 1.0e-01...Listo

La semilla a usar para la búsqueda de la raíz es: 0.64954

Salvando las semillas en un archivo "CSV"......Listo

Ejecutando el método de Newton-Raphson para f{3} y para la tolerancia 1.0e-10...Listo

Raíz hallada después de 4 iteraciones: 0.6062180653 +- 1.0e-10

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

Estimando el orden de convergencia......Listo

El orden de convergencia estimado para el método de Newton-Raphson es 1.99880274.

Salvando los resultados de la estimación del orden de convergencia en un archivo "CSV"......Listo

****************************
Calculando para v{4}:
```

```
Para el caso del freno del ascensor usamos Newton-Raphson para calcular el tiempo en el que se alcanza velocidad 0,

    @(t)3.0625*t.^2-7.84

Ejecutando el método de bisección para v{4}, como arranque, para la tolerancia 3.0e-01...Listo

La semilla a usar para la búsqueda de la raíz es: 1.75000

Salvando las semillas en un archivo "CSV"......Listo

Ejecutando el método de Newton-Raphson para v{4} y para la tolerancia 1.0e-10...Listo

Raíz hallada después de 4 iteraciones: 1.6000000000 +- 1.0e-10

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

Estimando el orden de convergencia......Listo

El orden de convergencia estimado para el método de Newton-Raphson es 1.99999853.

Salvando los resultados de la estimación del orden de convergencia en un archivo "CSV"......Listo



Ejecución del TP1 terminada.
```