

Módulo N° 1: Herramientas para operaciones numéricas y simbólicas

Contenidos

1	Herramientas para trabajo colaborativo en simulación de procesos dinámicos..	1
a.	Colaboratory.....	1
b.	GitHub.....	3
2	Solución numérica de ecuaciones.....	5
a.	Cálculo numérico.....	5
b.	Simulación empleando integración de Euler de un sistema lineal.....	7
c.	Ejemplo de integración usando Euler en un Motor CC	8
	Tiempo de muestreo adaptable.....	10
3	Solución analítica de ecuaciones diferenciales ordinarias.....	11
4	Herramientas de visualización	14
a.	FreeMat.....	14
b.	Octave.....	14
c.	Scilab	18
d.	Python.....	20
e.	R	21
5	Instructivo Symbolic en Octave	23
a.	Instalaciones	23
b.	Configuraciones	24
6	Comparación de resultados y ajustes	25

1 Herramientas para trabajo colaborativo en simulación de procesos dinámicos

a. Colaboratory

Una de las herramientas de control de código más completas es GitHub, <https://github.com>, y acompañada de Colaboratory <https://colab.research.google.com/> es una muy buena opción en los casos de simular algoritmos que se orientan a los

procesos dinámicos. Lo importante es que facilita el trabajo en equipo, ya que queda el rastro de qué es lo que ha contribuido cada integrante del proyecto.

A continuación, se muestra un ejemplo de uso para hallar una función de transferencia de un motor CC conociendo las entradas y salidas. Cabe aclarar, que el algoritmo fue desarrollado en Octave, y que luego fue ejecutado desde Colaboratory, donde se obtuvieron los resultados. Dichos resultados fueron copiados desde el mismo Colaboratory al GitHub, donde queda registrado el horario y todos los detalles.

El problema tratado requiere que se carguen los datos desde una planilla .xlsx, y para ello se hace clic debajo de Archivos, como se detalla en la Fig. 1. Una vez cargada, se debe poder ver, como se observa en la Fig. 2.

En la Fig. 2 se muestra el código que se requiere para correr el algoritmo desarrollado en Octave para obtener la función de transferencia del motor CC respecto de la tensión de entrada y del torque de carga. El comando

```
!apt install octave
```

genera una instalación de Octave en el entorno de ejecución de Colaboratory, y será borrado si se actualiza esa página web. Los demás comandos instalan las librerías empleadas por el .m que va a hacerse correr en ese entorno.

El comando

```
%%writefile ID_Chén.m
```

va a generar un archivo .m en el entorno con todo el texto que se escriba a continuación. Debe finalizar con un comando

```
save('ID_Chén.mat','-v7');
```

para que todos los resultados que se obtengan desde Octave sean almacenados en el entorno.

El comando

```
!octave -W ID_Chén.m
```

ejecuta al programa Octave para que corra al programa .m sin tratar de usar ventanas (tampoco plot), porque el cálculo se hará en el entorno de Colaboratory y no es posible generar ninguna ventana. Los resultados que devuelve el Octave están en el archivo .mat, y desde Python pueden cargarse y generarse las gráficas necesarias.

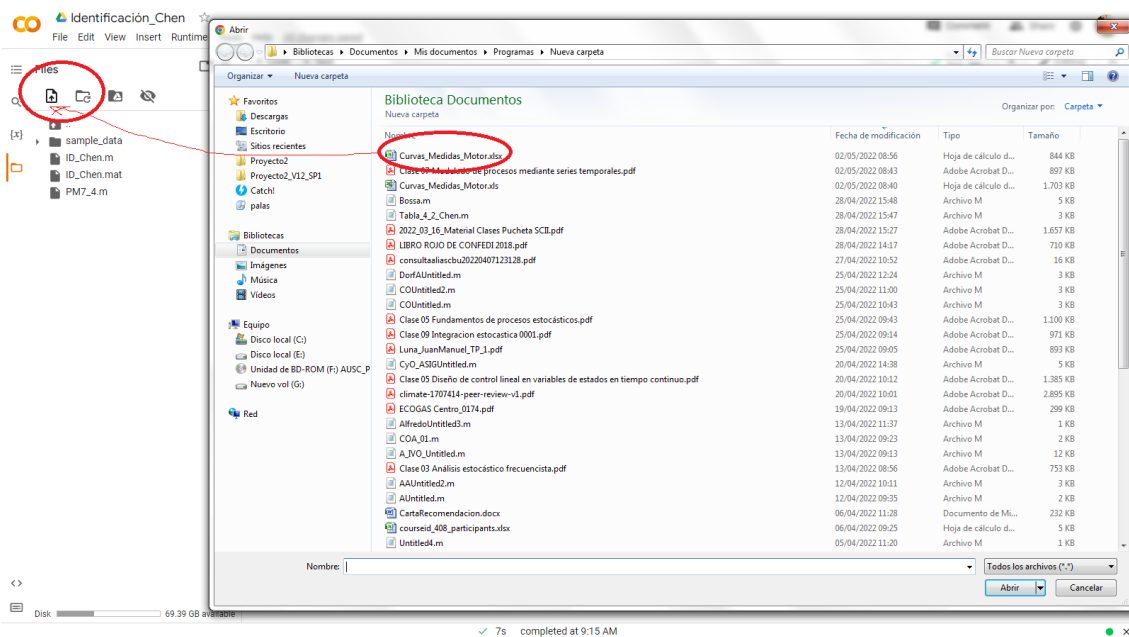


Fig. 1. Carga de una planilla xlsx en el entorno de trabajo de Colaboratory de Google.

```

[ ] !apt install octave

[ ] !apt install octave-signal

[ ] !apt install octave-io

[ ] !apt install octave-control

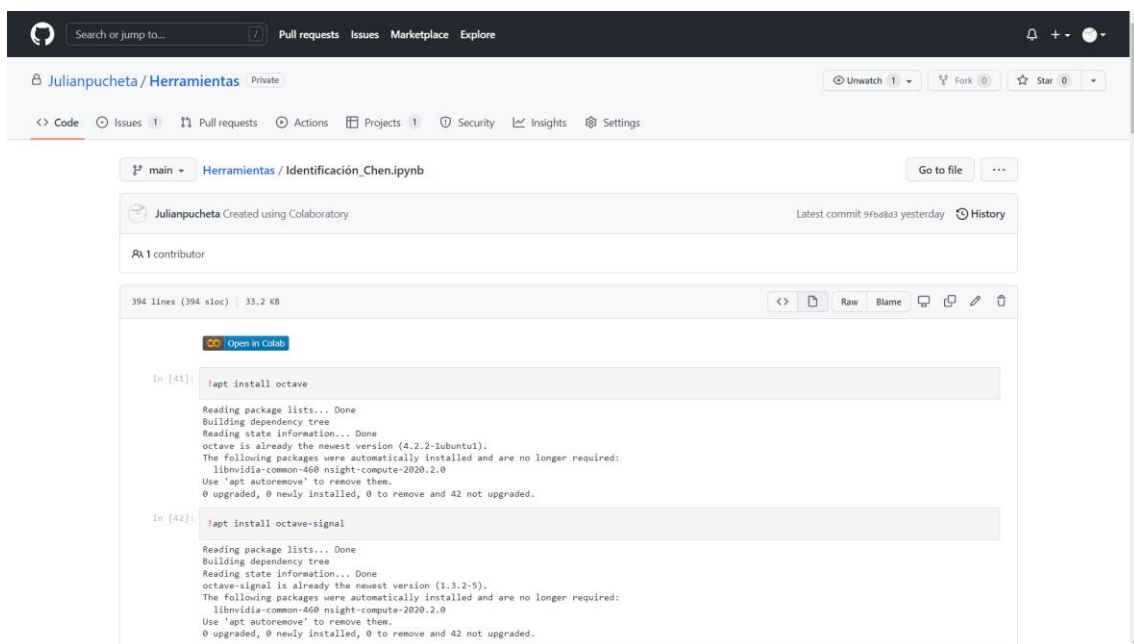
[ ] %%writefile ID_Chen.m
clc; clear all; close all;
pkg load control
pkg load signal
pkg load io
%{
-----
> A partir de las curvas de mediciones de las variables graficadas en la
Fig. 1-3, se requiere obtener el modelo del sistema considerando como
entrada un escalón de 12V, como salida a la velocidad angular, y a partir
de 0,1segundo se aplica un Tl aproximado de 7,5 10-2 Nm. En el archivo
Curvas_Medidas_Motor.xls están las mediciones, en la primer hoja los
valores y en la segunda los nombres. Se requiere obtener el modelo
dinámico, para establecer las constantes de la corriente.
-----
%}
tabla=xlsread('Curvas_Medidas_Motor.xls');
t_D=tabla(1,1);
y_D=tabla(1,2);
StepAmplitude=12; %12 V de entrada en Va
%Ar/va
ret = 0.0201;
t = 5e-4;
[val lugar] = min(abs(taret-t_D));%Busco en ret:t1
y_t=y_D(lugar);
t=t_D(lugar)-ret; %t1

```

Fig. 2. Código escrito en Colaboratory de Google. Nótese que lo atractivo es que puede trabajarse en línea desde varios usuarios a la vez, si se habilita la opción “Compartir” de arriba a la derecha.

b. GitHub

Todos los resultados pueden guardarse en GitHub, en éste caso se empleó el repositorio Herramientas, específicamente en https://github.com/Julianpucheta/Herramientas/blob/main/Identificaci%C3%B3n_Chen.ipynb Las capturas de las pantallas de los resultados que se almacenaron en GitHub están en la Fig. 3.



```

In [43]: !apt install octave-io

Reading package lists... Done
Building dependency tree
Reading state information... Done
octave-io is already the newest version (2.4.10-3).
The following packages were automatically installed and are no longer required:
  libnvidia-common-460 nvidia-compute-2020.2.0
Use 'apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 42 not upgraded.

In [44]: !apt install octave-control

Reading package lists... Done
Building dependency tree
Reading state information... Done
octave-control is already the newest version (3.0.0-5).
The following packages were automatically installed and are no longer required:
  libnvidia-common-460 nvidia-compute-2020.2.0
Use 'apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 42 not upgraded.

In [45]: %writefile ID_Chen.m
clc; clear all; close all;
pkg load control
pkg load signal
pkg load io
%{
-----
3- A partir de las curvas de mediciones de las variables graficadas en la
Fig. 1-3, se requiere obtener el modelo del sistema considerando como
entrada un escalón de 12V, como salida a la velocidad angular, y a partir
de 0,1segundo se aplica un TL aproximado de 7,5 10-2 Nm. En el archivo
Curvas_Medidas_Motor.xls están las mediciones, en la primer hoja los
valores y en la segunda los nombres. Se requiere obtener el modelo
dinámico, para establecer las constantes de la corriente.
-----
%}
tabla=load('Curvas_Medidas_Motor.xls');
t_Detabla(:,1);
y_Detabla(:,2);
StepAmplituden=12; %12 V de entrada en Va
%ur/va
ret = 0.0201;
t = 5e-4;
[valor lugar] = min(abs(t-ret-t_D)); %Busco en ret+t1
y_tny_D(lugar);
ret = 5e-4;
t = 5e-4;
[valor lugar] = min(abs(t-ret-t_D)); %Busco en ret+t1
y_tny_D(lugar);
t=t_D(lugar)-ret; %t1

[valor lugar] = min(abs(2*t-ret-t_D));
y_t2ny_D(lugar);
t2=t_D(lugar)-ret;

[valor lugar] = min(abs(3*t-ret-t_D));
y_t3ny_D(lugar);
t3=t_D(lugar)-ret;
% break
% Ksy(80)/U
k = 198.2488022/12;
% NCORROBORADOR DE PUNTOS
plot(tabla(:,1),tabla(:,2))
% hold on
% plot(t-ret,y_t,'o')
% plot(t2-ret,y_t2,'o')
% plot(t3-ret,y_t3,'o')

% break
%METODO DE CHEN
%Funcion de la forma G(s)=K*(s+T3)/[(s+T1+1).(s+T2+1)] luego se puede
%despreciar el cero
k1 = (1/StepAmplitude)*y_t/k-1;
k2 = (1/StepAmplitude)*y_t2/k-1;
k3 = (1/StepAmplitude)*y_t3/k-1;
b = 4*k1^3+k3^3*k1^2+k2^2-4*k2^3+k3^2+6*k1*k2*k3;
alfa1 = (k1*k2+k3-sqrt(b))/(2*(k1^2+k2));
alfa2 = (k1*k2+k3+sqrt(b))/(2*(k1^2+k2));
beta = (2*k1^3+3*k1*k2+k3-sqrt(b))/(sqrt(b));
T1 = (-t/log(alfa1));
T2 = (-t/log(alfa2));
Tl=real(T1)+2*real(T2); %importa sólo la parte real
T3 = -real(beta*(T1-T2)/T1);
sys_vastf(k,conv([T1 1],[T2 1]));
dt=3e-5;
t_s=0:dt:t_D(end-1);
u1=zeros(ret_t_Dt,1);
u2_Va=12*ones((.6-ret)/dt,1); %Va=12V
u1_Tzzeros(fix(.1000/dt)+1,1); %TL=0
u2_Tones(fix((.6-.100)/dt),1);
u_Va=[u1;u2;u3];
% plot(t_s,u_Va);title('Tensión de entrada')
u_T=[u1;u2;u3];
% plot(t_s,u_T);title('Torque de entrada')
[y1,t1,ent]=lsim(sys_va, u_Va, t_s, [0,0]);
% figure
% plot(tabla(:,1),tabla(:,2))
% hold on
% plot(t1,y1,'k')
% plot(t-ret,y_t,'o')
% plot(t2-ret,y_t2,'o')
% plot(t3-ret,y_t3,'o')
% break
%ur/TL
% t_t1 = 0.1002-ret_t1;
% y_t_t1 = 160.549509;
ret_t1 = 0.1+2e-4;
t_t1=2e-4;
[valor lugar] = min(abs(t_t1-ret-t_D));
y_t_t1ny_D(lugar);
t_t1t_D(lugar)-ret_t1;
% t2_t1 = 0.1005-ret_t1;
% y_t2_t1 = 101.4371121;
[valor lugar] = min(abs(2*t_t1-ret-t_D));
y_t2_t1ny_D(lugar);
t2_t1t_D(lugar)-ret_t1;
% t3_t1 = 0.1008-ret_t1;
% y_t3_t1 = 72.4383423;
[valor lugar] = min(abs(3*t_t1-ret-t_D));
y_t3_t1ny_D(lugar);
t3_t1t_D(lugar)-ret_t1;
% 7,5 10-2
TL=7.5e-2;% TL:Amplitud del escalon de Torque de entrada
% Ksy(80)/U
k_t1 = -(46.2-198)/TL;

% NCORROBORADOR DE PUNTOS
plot(tabla(:,1),-tabla(:,2)+k)
% hold on
% plot(t_t1-ret_t1,y_t_t1,'o')
% plot(t2_t1-ret_t1,y_t2_t1,'o')
% plot(t3_t1-ret_t1,y_t3_t1,'o')

%METODO DE CHEN
% % k1_t1 = (1/TL)*y_t_t1/k_t1-1;
% % k2_t1 = (1/TL)*y_t2_t1/k_t1-1;
% % k3_t1 = (1/TL)*y_t3_t1/k_t1-1;
yid_1a=(y_t_t1-198.2)
yid_2a=(y_t2_t1-198.2)
yid_3a=(y_t3_t1-198.2)
% plot(tabla(:,1),tabla(:,2));hold on;%plot(t_t1,yid_1,'o',t2_t1,yid_2,'o',t3_t1,yid_3,'o')
% plot(t_t1-ret_t1,yid_1,'o',t2_t1-ret_t1,yid_2,'o',t3_t1-ret_t1,yid_3,'o')

k1_t1 = (1/TL)*yid_1/k_t1-1;
k2_t1 = (1/TL)*yid_2/k_t1-1;
k3_t1 = (1/TL)*yid_3/k_t1-1;

```



Fig. 3. Capturas del GitHub que almacena los resultados del Colaboratory Google corriendo un .m para identificación de una planta a partir de series de datos provistas en un xlsx. Este código está disponible on line en [17].

2 Solución numérica de ecuaciones

a. Cálculo numérico

En la simulación de procesos dinámicos, se requiere el cálculo numérico para evidenciar el comportamiento que puede tener el sistema real que es representado por las ecuaciones diferenciales. Aquí se va a analizar a ecuaciones diferenciales que pueden

aproximarse mediante ecuaciones diferenciales ordinarias (EDO). Por lo que la solución de ecuaciones diferenciales ordinarias pasa a ser el objeto de estudio inicial, y específicamente el tipo de soluciones que es frecuente de resolver en sistemas de control, del tipo

$$\dot{x} = a \cdot x(t); x(0) = x_0 \quad (2-1)$$

donde a es un coeficiente y x_0 es la condición inicial de la función. Su solución analítica exacta es

$$x_t = x_0 e^{a \cdot t}. \quad (2-2)$$

No obstante, para tratar ecuaciones generales, del tipo

$$\dot{x} = f(t, x(t)); x(t_0) = x_0 \quad (2-3)$$

no siempre es simple encontrar la solución analítica exacta. Por lo que, para problemas de Ingeniería, conviene dominar a los métodos numéricos.

En lo que sigue se detallará el método de Euler directo, que será empleado en el desarrollo del curso. Éste método es la base de los demás métodos. Sea entonces, la ecuación diferencial (2-3), si se considera un intervalo temporal donde va a analizarse la solución y éste es dividido en N partes iguales, se puede escribir

$$x_{n+1} = x_n + h \cdot f(t_n, x_n); x_n|_{n=0} = x_0 \quad (2-4)$$

donde h es el intervalo temporal de integración, con $n=0,1,2 \dots N-1$.

Dado que éste método presenta algunos inconvenientes en ciertos casos, se han desarrollado modificaciones que tratan de mejorar su desempeño. Una modificación es conocida como Euler regresivo (backward), que consiste en valuar la función en el final del intervalo de tiempo, es decir, se modifica a la (2-4) como

$$x_{n+1} = x_n + h \cdot f(t_{n+1}, x_{n+1}); x_n|_{n=0} = x_0 \quad (2-5)$$

pero ahora hay que resolver una ecuación algebraica en cada intervalo de tiempo. Así, para acelerar el procedimiento de cálculo, se han propuesto modificaciones como la del punto medio, que consiste en valuar a la función en el medio del intervalo

$$x_{n+1} = x_n + h \cdot f\left(t_n + \frac{1}{2}h, x_n + \frac{1}{2}h \cdot f(t_n, y_n)\right); x_n|_{n=0} = x_0 \quad (2-6)$$

que conduce a la familia de métodos Runge–Kutta.

En adelante se va a emplear el algoritmo detallado en la expresión (2-4), para resolver diferentes casos pertinentes a los procesos en ingeniería y al control automático.

Como ejemplo, se van a comparar diferentes resultados de la solución numérica para los programas computacionales cuando $a=-1$ en la EDO (2-1). Ahora las expresiones vistas tienen que evaluarse para la función propuesta, donde la (2-4) reescribe como

$$x_{n+1} = x_n + h \cdot a \cdot x_n; x_n|_{n=0} = x_0 \quad (2-7)$$

la (2-5) como

$$x_{n+1} = x_n + h \cdot a \cdot x_{n+1}; x_n|_{n=0} = x_0 \quad (2-8)$$

y la (2-6)

$$x_{n+1} = x_n + h \cdot a \left(x_n + \frac{1}{2} h \cdot a x_n \right). \quad (2-9)$$

Empleando por ejemplo Wolfram Alpha

[1],

<https://www.wolframalpha.com/input?i=forward+euler+method+vs.+backward+euler+for+x%27%3D+-x%2C+x%280%29%3D1>

se obtiene la captura indicada en la Fig. 4. Nótese que la respuesta a la instrucción de comparar dos métodos numéricos, incluye una serie de resultados para diversos métodos que sirven como referencia de exactitud. Además, para éste caso se asigna el intervalo temporal entre 0 y 3, con una cantidad de pasos intermedios de 10.

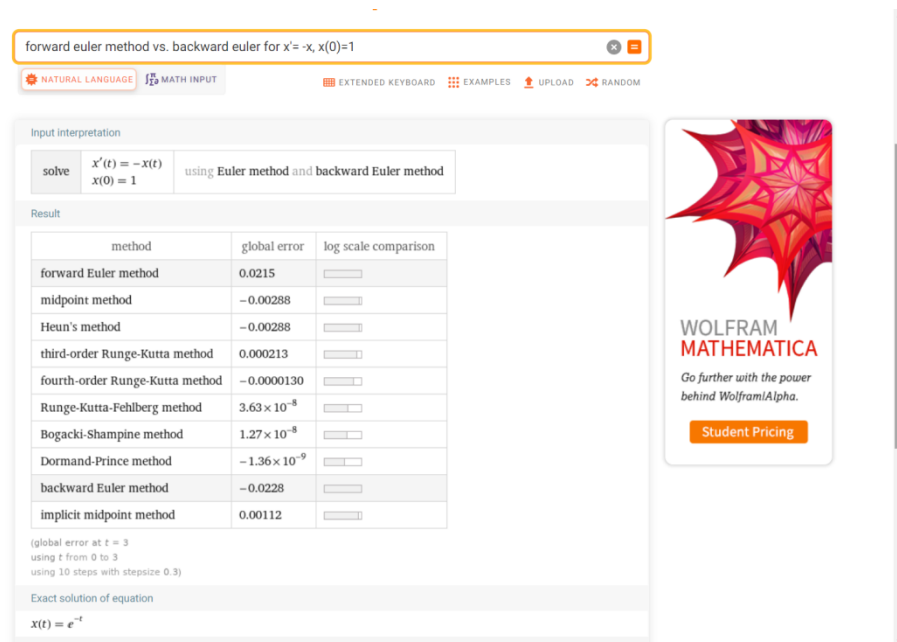


Fig. 4. Respuesta del sitio

[1] para la comparación de emplear diferentes métodos en la solución de la ecuación diferencial ordinaria siendo $a=-1$ con $0 \leq t \leq 3$ y 10 pasos intermedios. Están sombreados los métodos progresivos y regresivos de Euler.

b. Simulación empleando integración de Euler de un sistema lineal

Para simular un proceso lineal mediante integración de Euler, es necesario establecer dos tiempos, uno es el tiempo de integración o intervalo de muestreo y otro es el tiempo de simulación, es decir, la duración de todo el cálculo.

Sabiendo que la solución del sistema lineal es

$$x(t) = e^{At} x(t_0) + \int_0^t e^{A(t-s)} B u(s) ds. \quad (2-10)$$

con una acción de control dada, se procede a discretizar el tiempo según las constantes de tiempo involucradas, que son los autovalores de A y en especial la de mayor rapidez que es la que está más a la izquierda en el plano complejo.

Empleando el teorema de Cayley Hamilton se sabe que

$$e^{At} = \alpha_0(t)I + \alpha_1(t)A + \alpha_2(t)A^2 + \dots + \alpha_{n-1}(t)A^{n-1} = \sum_{k=0}^{n-1} \alpha_k(t)A^k \quad (2-11)$$

por lo que sólo se deben encontrar las funciones temporales $\alpha_i(t)$, que son n incógnitas. Se puede multiplicar miembro a miembro la igualdad (2-11) por un autovector correspondiente a un autovalor λ_1 y se obtiene

$$e^{\lambda_1 t} = \alpha_0(t) + \alpha_1(t)\lambda_1 + \alpha_2(t)\lambda_1^2 + \dots + \alpha_{n-1}(t)\lambda_1^{n-1} \quad (2-12)$$

Aquí se puede ver que para saber cuál es el valor de t cuando la exponencial sea igual a 0,95 si éste es el autovalor más a la izquierda en el semiplano izquierdo para obtener el tiempo de integración. Por otro lado, si el autovalor es el más cercano al semiplano derecho, corresponde a la dinámica más lenta y por lo tanto sirve para definir cuál es el tiempo de simulación total, ya que con un valor de cinco veces mayor es suficiente.

c. Ejemplo de integración usando Euler en un Motor CC

Por ejemplo, sean las matrices del sistema lineal de un motor de corriente continua, donde x_1 está asociada a la corriente y x_2 a la velocidad angular, dado por

$$A = \begin{bmatrix} -152000 & -23100000 \\ 1 & 0 \end{bmatrix}, B = \begin{bmatrix} 1 \\ 0 \end{bmatrix}, C = [0 \quad 3.57 \cdot 10^9] \quad (2-13)$$

Nótese que los autovalores de A son $\lambda_1 = -1,5185 \cdot 10^5$, $\lambda_2 = -152,1259$. Aquí se puede encontrar el tiempo al que corresponde el 95% de la dinámica más rápida $\exp(\lambda_1 t)$ es $t_R = \ln(.95) / \lambda_1$

$$t_R = \frac{\ln(0,95)}{\lambda_1} = 3,3779 \cdot 10^{-7} \quad (2-14)$$

de donde se selecciona el tiempo de integración menor que éste t_R calculado, por ejemplo, tres veces menor, como $1 \cdot 10^{-7}$.

Luego, para calcular el tiempo de simulación en el cual prácticamente ya no hay más transitorios, se toma la constante de tiempo más lenta y se calcula el tiempo para que la $\exp(\lambda_2 t)$ llegue al 5%, éste tiempo sería

$$t_L = \frac{\ln(0,05)}{\lambda_2} = 0,0197 \quad (2-15)$$

se puede tomar, por ejemplo, el triple de éste valor t_L , 0,06 seg.

En la Tabla 1 se detalla el código para simular el desarrollo de las variables de estado con la respuesta al escalón, es decir $u=1$. En la Tabla 2 está el detalle del modelo lineal del motor CC. En la Fig. 5 se muestra la evolución de las variables, nótese las diferentes dinámicas que presentan cada una.

```
clear;
close all; X=[0; 0]; ii=0; t_etapa=1e-7;
for t=0:t_etapa:.06
    ii=ii+1;
    X=modmotor(t_etapa, X, 1);
    x1(ii)=X(1); x2(ii)=X(2);
end
t=0:t_etapa:.06;
subplot(2,2,1); plot(x1,x2, '+'); title('Plano de fases'); xlabel('x_1'); ylabel('x_2');
subplot(2,2,2); plot(t, [0 3.57e9]*[x1; x2], '+'); xlabel('Tiempo [Seg.]');
ylabel('y_t'); title('Salida y');
subplot(2,2,3); plot(t,x1, '+'); title('x_1');
xlabel('Tiempo [Seg.]');
subplot(2,2,4); plot(t,x2, '+'); title('x_2');
xlabel('Tiempo [Seg.]');
```

Tabla 1. Código de integración de Euler para el sistema lineal del motor de CC.

```
function [X]=modmotor(t_etapa, xant, accion)
h=1e-7;
A=[-152000    -23100000
    1         0];
B=[1
    0];
C=[0 3.57e9];
x=xant; u=accion;
for ii=1:t_etapa/h
    xp=A*x+B*u;
    x=x+xp*h;
end
X=x;
```

Tabla 2. Código del modelo del sistema lineal del motor de CC.

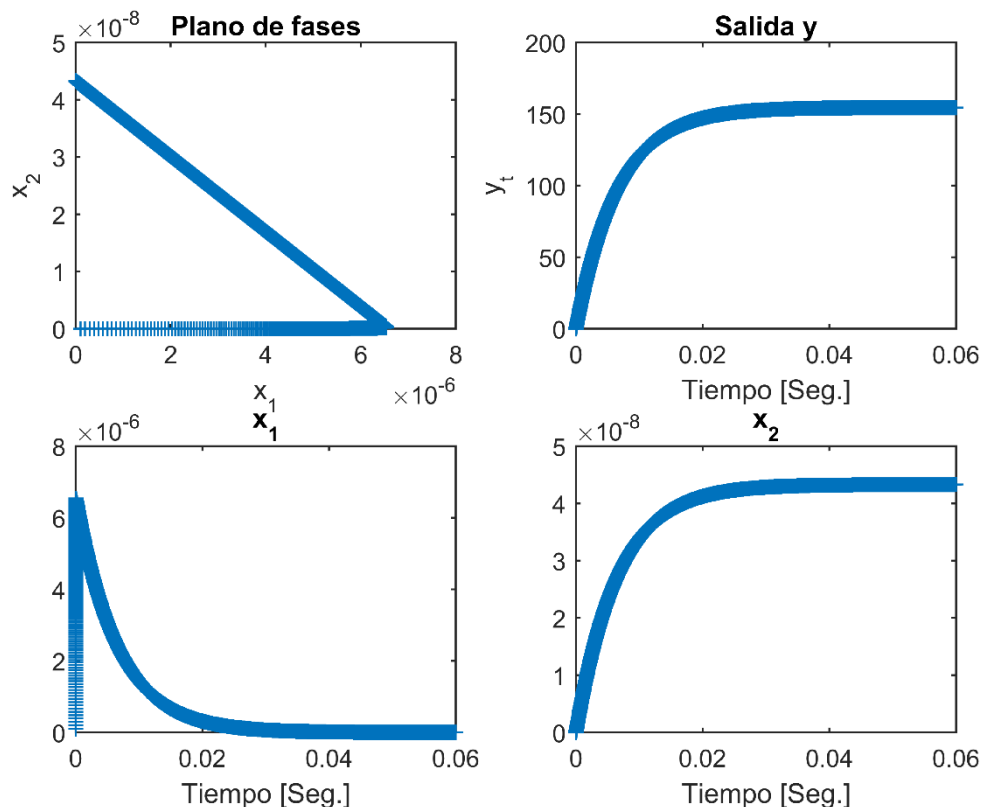


Fig. 5. Evolución de la respuesta al escalón obtenida para el motor.

Tiempo de muestreo adaptable

Puede notarse que el hecho de que una dinámica sea mucho más rápida que la otra puede tenerse en cuenta en el cálculo. Es decir, se puede pensar que el tiempo de etapa (o tiempo de muestreo) en el primer tramo del tiempo de simulación sea de $1 \cdot 10^{-7}$ y luego sí pasar a un tiempo de muestreo de $t_L/200$, por ejemplo. Se puede plantear un arreglo de intervalos de muestreo de 200 valores para cada intervalo posible. En el caso del motor CC serían dos valores para T_0 , uno es $1 \cdot 10^{-7}$ y el otro es $1 \cdot 10^{-4}$. Esto agiliza mucho la simulación, y no se pierde detalle en la dinámica rápida. El código para modificar el intervalo de etapa o tiempo de muestreo está en la Tabla 3, y la gráfica en la Fig. 6.

```
clear;close all;
X=[0; 0];ii=0;
t_R=1e-7;t_L=0.06;t_sim=0;T_F=t_L;
T0=[t_R*ones(200,1); (t_L/200)*ones(200,1)];
while t_sim<T_F
    ii=ii+1;
    t_sim=t_sim+T0(ii);
    t(ii)=t_sim;
    X=modmotor(T0(ii), X, 1);
    x1(ii)=X(1);x2(ii)=X(2);
end
subplot(2,2,1);plot(x1,x2,'+');title('Plano de fases');xlabel('x_1');ylabel('x_2');
subplot(2,2,2);plot(t,[0 3.57e9]*[x1;x2],'+');xlabel('Tiempo [Seg.]');
ylabel('y_t');title('Salida y');
subplot(2,2,3);plot(t,x1,'+');title('x_1');
xlabel('Tiempo [Seg.]');
subplot(2,2,4);plot(t,x2,'+');title('x_2');
xlabel('Tiempo [Seg.]');
```

Tabla 3. Código para simulación del modelo del sistema lineal del motor de CC con tiempo de integración adaptado.

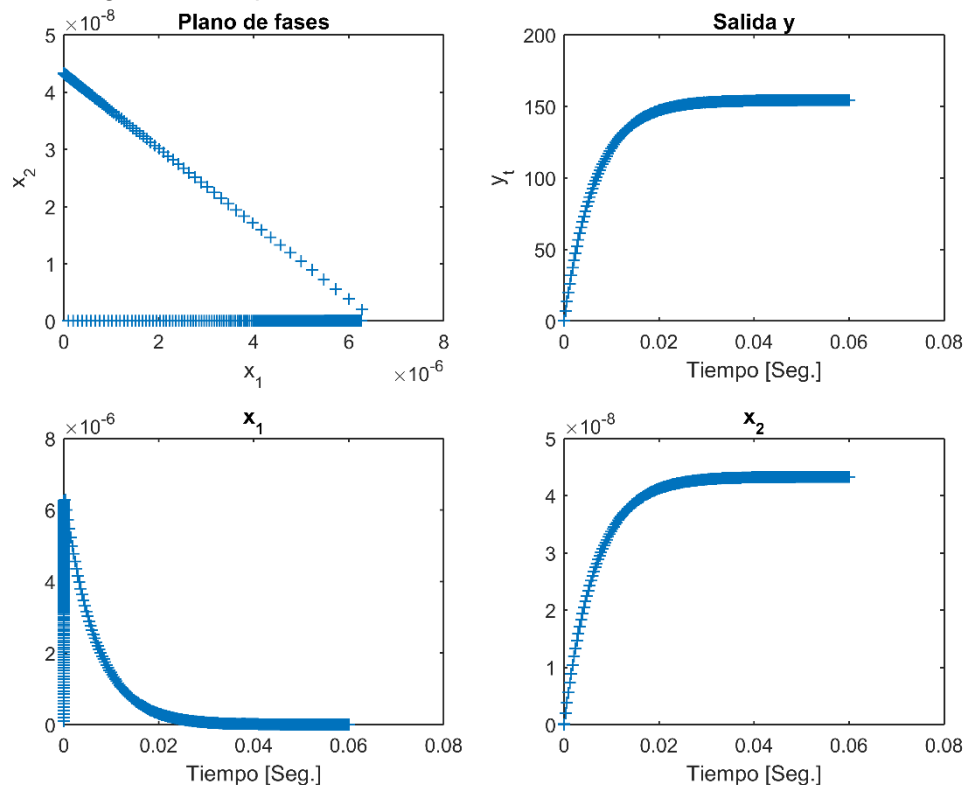


Fig. 6. Evolución de la respuesta al escalón obtenida para el motor CC con tiempo de muestreo adaptado a cada dinámica.

Nótese que el algoritmo es sencillo, pero lo que es importante es *visualizar* el resultado con el objetivo de mostrar la potencialidad de la solución que se está evaluando y proponer métodos de mediciones de errores de ser necesario.

3 Solución analítica de ecuaciones diferenciales ordinarias

Dada la necesidad de resolver un sistema de ecuaciones diferenciales para obtener su expresión simbólica, puede emplearse el WolframAlpha

[1], pero es más versátil el Octave.

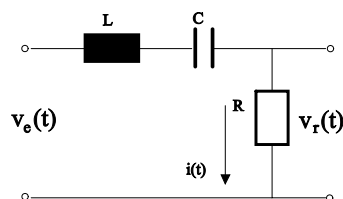


Fig. 3-1. Esquemático del circuito RLC.

Por ejemplo, para el sistema eléctrico de la Fig. 3-1, se requiere obtener la evolución temporal de la corriente para una entrada escalón de tensión. Las ecuaciones que describen el funcionamiento son

$$\begin{cases} \frac{di}{dt} = -\frac{R}{L} i - \frac{1}{L} v_c + \frac{1}{L} v_e \\ \frac{dv_c}{dt} = \frac{1}{C} i \end{cases} \quad (3-1)$$

<https://www.wolframalpha.com/input?i=i%28t%29%27%3D-i%28t%29+-+v%28t%29%2B+UnitStep%280%29%2C+v%28t%29%27%3D+i%28t%29%2C+v%280%29%3D0%2C+i%280%29%3D0>

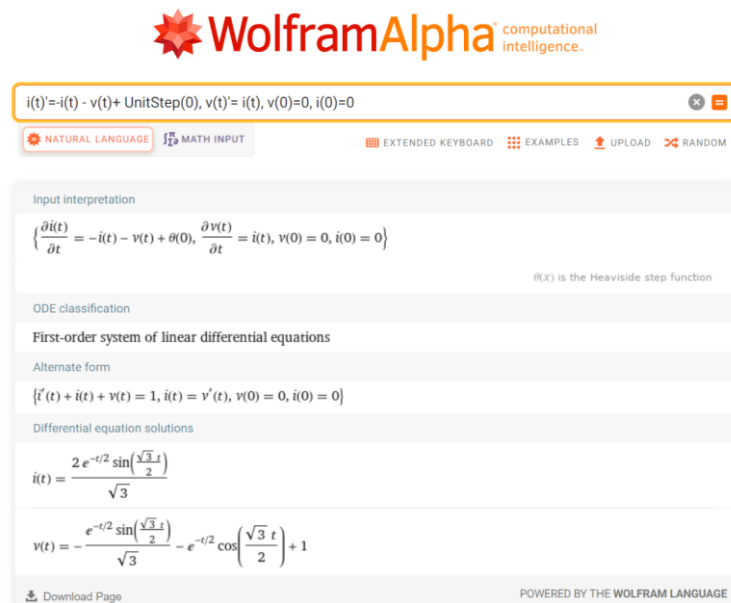


Fig. 2. Resultado que devuelve WolframAlpha

[1] cuando se introduce el texto $i(t)' = -i(t) - v(t) + \text{UnitStep}(0)$, $v(t)' = i(t)$, $v(0)=0$, $i(0)=0$.

```
% #R=2.2e3;L=10e-6;C=100e-9;
clc;clear all;
pkg load symbolic
syms ii(t) v(t)
ode2=diff(v)==ii;
% ode1=diff(ii)==-R/L*ii-1/L*v+1/L*heaviside(0);
ode1=diff(ii)==-ii-v+heaviside(0,1);%heaviside(0) devuelve .5
cond1 = ii(0) == 0;
cond2 = v(0) == 0;
conds = [cond1;cond2];
odes = [ode1; ode2];
Sol=dsolve(odes,ii(0) == .0,v(0) == .0);
simplify(Sol{1})
simplify(Sol{2})
```

Tabla 4. Script de Octave para cálculo que devuelve la solución a la ecuación diferencial que circuito RLC de valores normalizados en 1.

```

ar Ventana Ayuda Noticias
Directorio actual: sr_Julian\FCEFYN UNC\Extension\2022 01\M1
Ventana de comandos
Symbolic pkg v2.9.0: Python communication link active, SymPy v1.4.
ans = (sym)

      -t
      ---
      2
      2
      2*\sqrt{3}*e
      *sin\frac{\sqrt{3}*t}{2}
      ii(t) = -----
      3

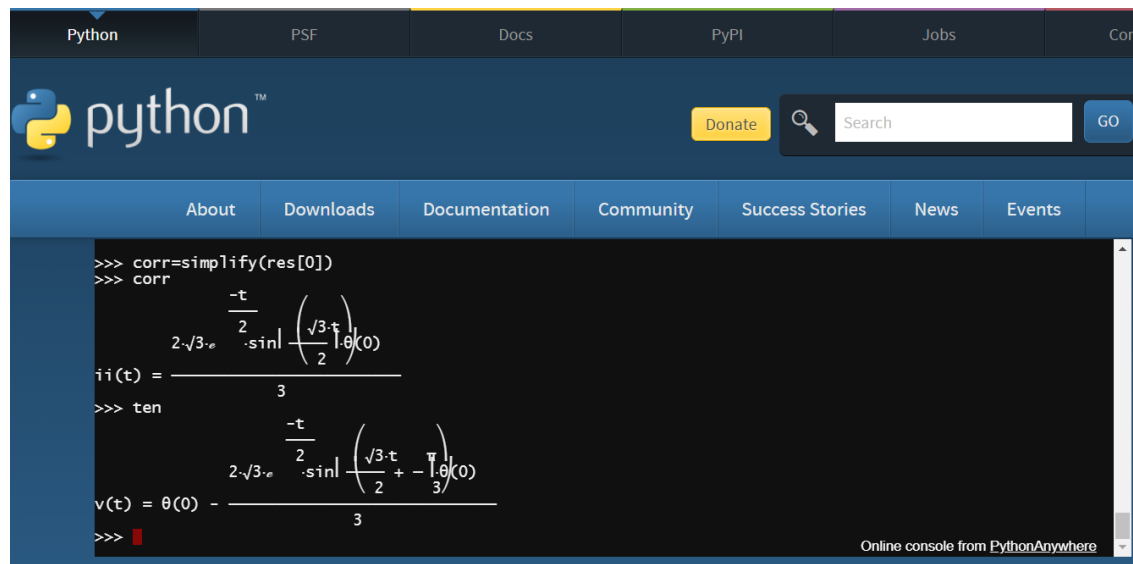
ans = (sym)

      -t
      ---
      2
      2
      2*\sqrt{3}*e
      *sin\frac{\sqrt{3}*t}{2}
      v(t) = 1 - -----
      3
>>

```

Fig. 3. Resultado de cálculo simbólico para solución de un sistema de ecuaciones diferenciales ordinarias en Octave, con valores R L C normalizados en 1.

Incluso en Python [15], si se usa su versión interactiva en línea, se puede emplear el código mostrado en la Tabla 5 y se obtendrá el resultado que se observa en la Fig. 4.



```

Python PSF Docs PyPI Jobs Cor
python
Donate Search GO
About Downloads Documentation Community Success Stories News Events
>>> corr=simplify(res[0])
>>> corr
      -t
      ---
      2
      2
      2*\sqrt{3}*e
      *sin\frac{\sqrt{3}*t}{2}
      ii(t) = -----
      3
>>> ten
      -t
      ---
      2
      2
      2*\sqrt{3}*e
      *sin\frac{\sqrt{3}*t}{2}
      v(t) = \theta(0) - -----
      3
>>>
Online console from PythonAnywhere

```

Fig. 4. Resultado de cálculo simbólico usando sympy de Python.

```

#https://scipy-lectures.org/packages/sympy.html
from sympy import *
import numpy as np
from sympy.functions.special.delta_functions import Heaviside
init_printing(use_unicode=True)
t=symbols('t')
ii=Function('ii')
v=Function('v')
ODE1=Eq(Derivative(ii(t),t),-ii(t)-v(t)+Heaviside(0))
ODE2=Eq(Derivative(v(t),t),ii(t))
eq=(ODE1,ODE2)
res=dsolve(eq,hint='all',ics={ii(0):0,v(0):0})
#Resultado en res[0] la corriente, y en res[1] la tensión
ten=simplify(res[1])

```

```
corr=simplify(res[0])
#Resultado en corr y ten
```

Tabla 5. Script para Python [15] cálculo que devuelve la solución a la ecuación diferencial que circuito RLC en R.

4 Herramientas de visualización

La visualización de los resultados mediante gráficas de las simulaciones depende del paquete de soft que se emplee. Aquí van a detallarse algunas opciones, que son muy divulgadas en la comunidad de codificación en software libre, como es FreeMat, Octave, Scilab, Python y R.

a. FreeMat

Un paquete de soft para simulaciones numéricas y cómputo es el ofrecido por los desarrolladores de Google, que se llama FreeMat [4]. Está en desarrollo, es open-source, pero es más limitado en el uso de recursos de simulación numérica comparado con Octave. En la Fig. 5 se muestra el caso de una simulación de un sistema dinámico empleando FreeMat. La gráfica es poco elegante, pero es fiel al cálculo.

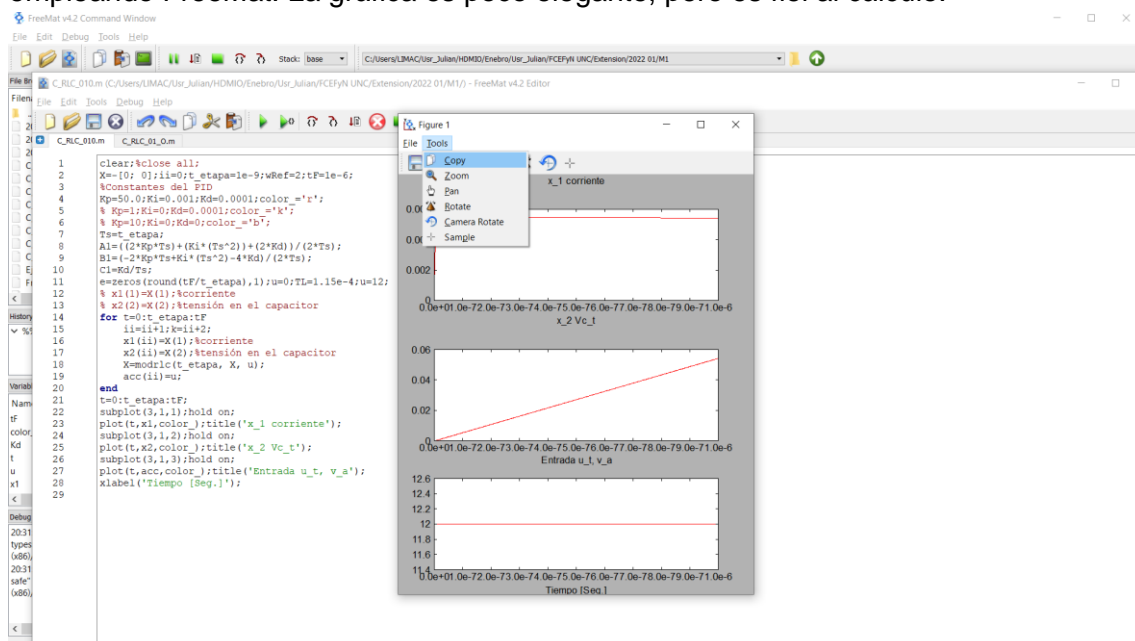


Fig. 5. Resultado de simulación numérica empleando FreeMat. Nótese que es poco elegante en el plot de curvas.

b. Octave

Un lenguaje muy intuitivo para realizar simulaciones de procesos dinámicos es la “programación .m” propuesto por Matlab® y la versión de código abierto más difundido es el Octave [5], que permite realizar gráficas elegantes y clara como muestra la Fig. 6. Existen paquetes para mejorar esa clase de gráficas, si se cargan los paquetes del Octave que facilitan la presentación visual, como es el

`graphics_toolkit("gnuplot")`. En la Fig. 7 se muestra el aspecto cuando se emplea éste paquete, pero desde Octave 6.0 no ha tenido continuidad y su uso está desaconsejado.

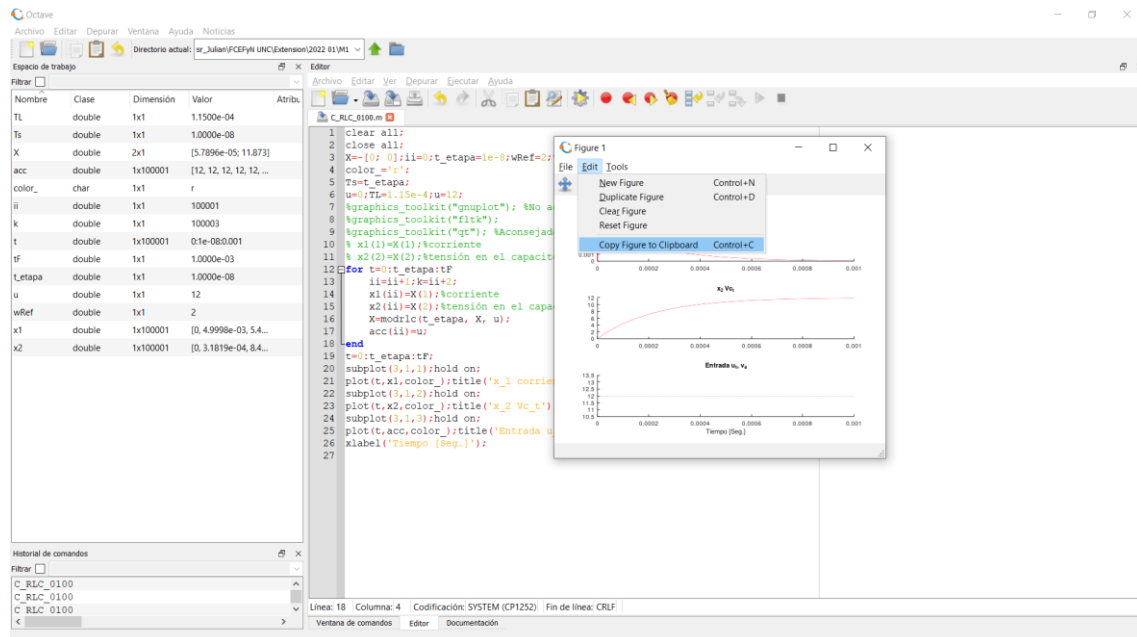


Fig. 6. Resultado de simulación numérica empleando Octave. Es completo el resultado que permite ésta instrucción `plot()`, que tiene varias posibilidades según los paquetes.

Mediante la instrucción `available_graphics_toolkits()` se obtienen los paquetes instalados en Octave para manipular las gráficas una vez realizadas:

```

ans =
{
    [1,1] = fltk
    [1,2] = gnuplot
    [1,3] = qt
}

```

Posteriormente, se puede emplear la instrucción `graphics_toolkit("gnuplot")`; para usar el paquete de herramientas que provee `gnuplot`. No obstante, Octave recomienda `fltk` o `qt`.

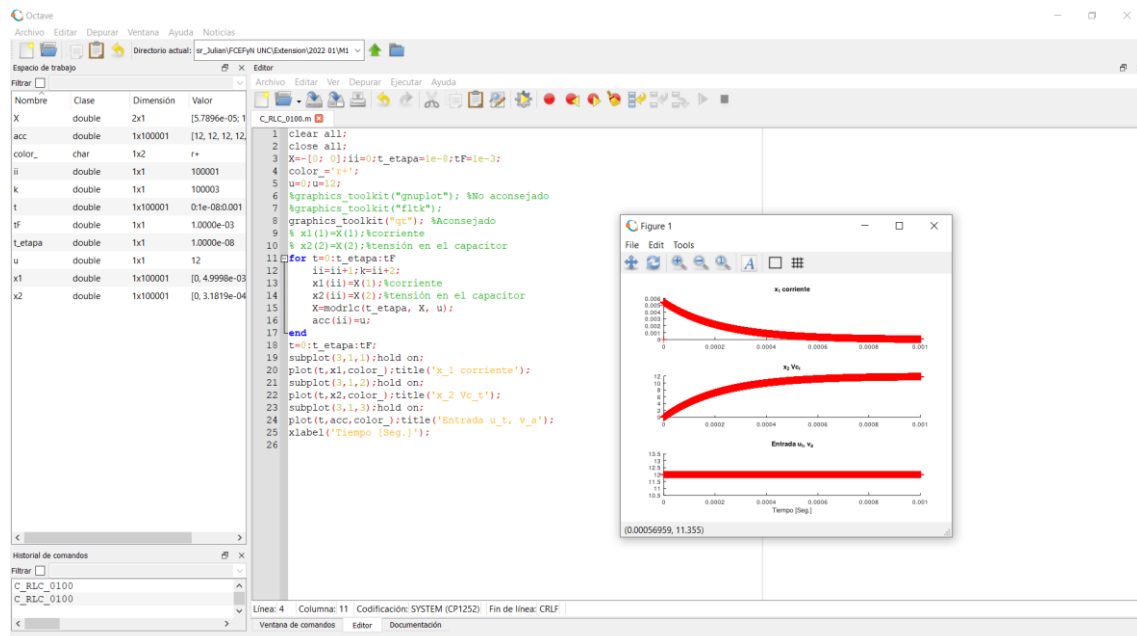


Fig. 9. Resultado de simulación numérica empleando Octave y su `graphics_toolkit("fltk")`. Aquí se cambió el indicador por defecto del plot "." por el "+" logrando que la línea del gráfico tenga un efecto de ser gruesa. En la ventana de la gráfica hay controles para manipular el gráfico agregando grillas, textos o cambiando la ubicación del plot.

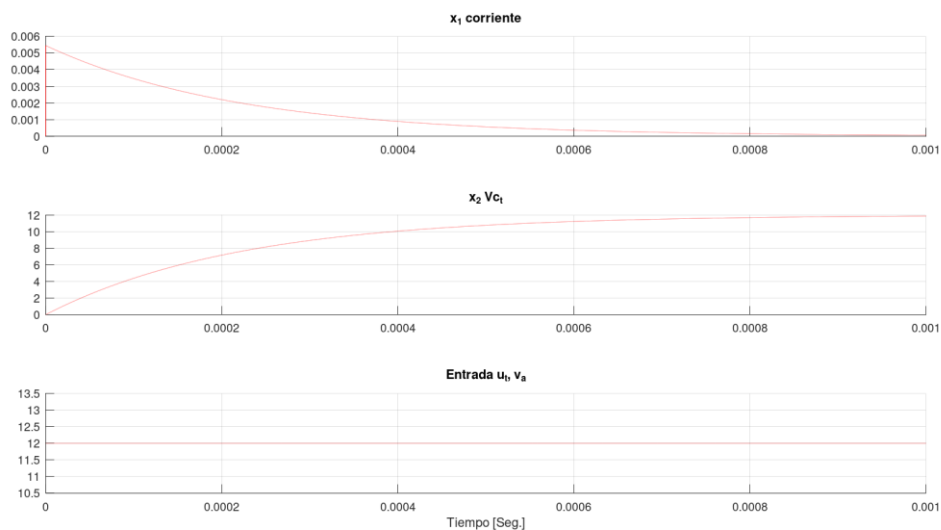


Fig. 10. Para `qt` o `fltk`, ésta es la imagen que se copia al clipboard usando Edit->Copy Figure to Clipboard cuando la imagen está maximizada, caso contrario, no se mantienen los tamaños de las fuentes.

```
clear all;
close all;
X=-[0; 0];ii=0;t_etapa=1e-8;wRef=2;tF=1e-3;
color_='r';
Ts=t_etapa;
%A1=((2*Kp*Ts)+(Ki*(Ts^2))+(2*Kd))/(2*Ts);
%B1=(-2*Kp*Ts+Ki*(Ts^2)-4*Kd)/(2*Ts);
```

```

%Cl=Kd/Ts;
%e=zeros(round(tF/t_etapa),1);
u=0;TL=1.15e-4;u=12;
%graphics_toolkit("gnuplot"); %No aconsejado
%graphics_toolkit("fltk");
%graphics_toolkit("qt"); %Aconsejado
% x1(1)=X(1);%corriente
% x2(2)=X(2);%tensión en el capacitor
for t=0:t_etapa:tF
    ii=ii+1;k=ii+2;
    x1(ii)=X(1);%corriente
    x2(ii)=X(2);%tensión en el capacitor
    X=modrlc(t_etapa, X, u);
    acc(ii)=u;
end
t=0:t_etapa:tF;
subplot(3,1,1);hold on;
plot(t,x1,color_);title('x_1 corriente');
subplot(3,1,2);hold on;
plot(t,x2,color_);title('x_2 Vc_t');
subplot(3,1,3);hold on;
plot(t,acc,color_);title('Entrada u_t, v_a');
xlabel('Tiempo [Seg.]');

%Funcion modelo
function [X]=modrlc(t_etapa, xant, accion)
h=1e-9;t_simul=1e-3;R=2.2e3;L=10e-6;C=100e-9;
A=[-R/L, -1/L;1/C,0];B=[1/L;0];%
% eig(A)
% 1./ans
C=[R 0];u=accion;
x=xant;
for ii=1:t_etapa/h
    xp=A*x+B*u;
    x=x+xp*h;
end
X=[x];%x1 corriente, x2 tensión

```

Tabla 6. Script para simulación de un circuito RLC en Octave.

c. Scilab

Es un programa open source de distribución libre [6], con mucha audiencia en el cómputo científico, y se asemeja mucho a la programación .m que tienen los programas antes descritos. Se pueden lograr gráficos elegantes, sólo es necesario dominar las características de las funciones y el Latex [7], por ejemplo para la misma simulación anterior, se tiene la Fig. 11.

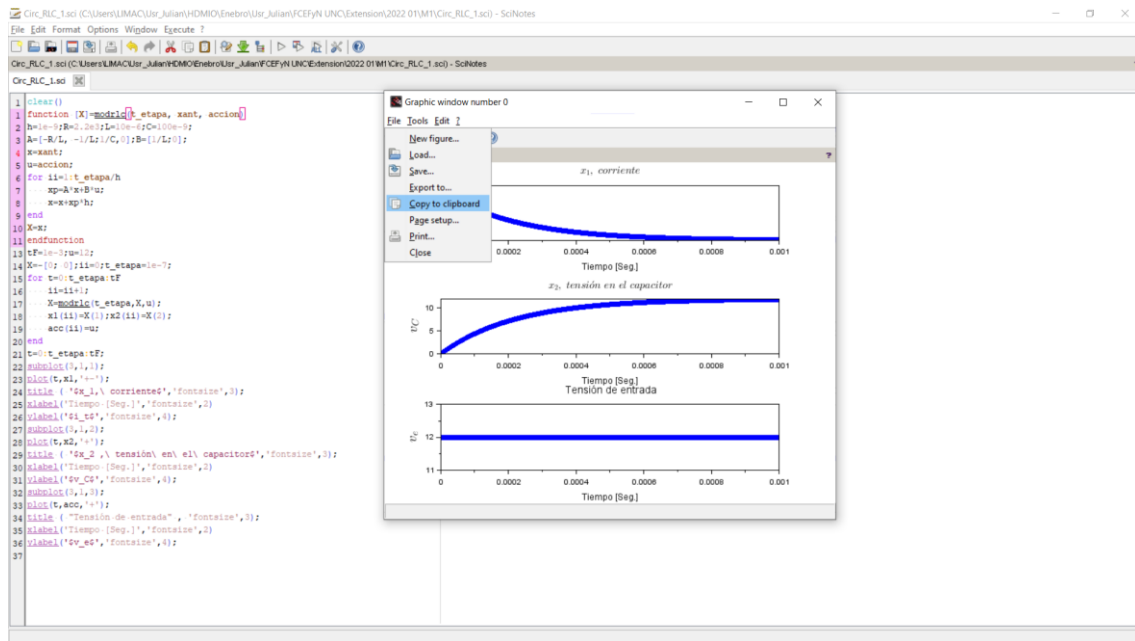


Fig. 11. Resultado de simulación numérica empleando Scilab [6].

```

0001 clear()
0002 function [X]=modrlc(t_etapa, xant, accion)
0003 h=1e-9;R=2.2e3;L=10e-6;C=100e-9;
0004 A=[-R/L, -1/L;1/C,0];B=[1/L;0];
0005 x=xant;
0006 u=accion;
0007 for ii=1:t_etapa/h
0008     xp=A*x+B*u;
0009     x=xp*h;
0010 end
0011 X=x;
0012 endfunction
0013 tF=1e-3;u=12;
0014 X=[0;0];ii=0;t_etapa=1e-7;
0015 for t=0:t_etapa:tF
0016     ii=ii+1;
0017     X=modrlc(t_etapa,X,u);
0018     x1(ii)=X(1);x2(ii)=X(2);
0019     acc(ii)=u;
0020 end
0021 t=0:t_etapa:tF;

```

```

0022 subplot(3,1,1);
0023 plot(t,x1,'+-');
0024 title(' $x_1$, corriente$', 'fontsize',3);
0025 xlabel('Tiempo [Seg.]', 'fontsize',2)
0026 ylabel('$i_t$', 'fontsize',4);
0027 subplot(3,1,2);
0028 plot(t,x2,'+');
0029 title(' $x_2$, tensión en el capacitor$', 'fontsize',3);
0030 xlabel('Tiempo [Seg.]', 'fontsize',2)
0031 ylabel('$v_C$', 'fontsize',4);
0032 subplot(3,1,3);
0033 plot(t,acc,'+');
0034 title(' "Tensión de entrada" ', 'fontsize',3);
0035 xlabel('Tiempo [Seg.]', 'fontsize',2)
0036 ylabel('$v_e$', 'fontsize',4);

```

Tabla 7. Script para simulación de un circuito RLC en Scilab.

d. Python

Otro programa que es muy difundido para la simulación de procesos dinámicos, es el Python, que tiene capacidad de hacer gráficas de simulaciones de procesos dinámicos, con varios cambios en la forma de escribir el código, como se puede observar en la Fig. 12 y en la Tabla 8. Hay que instalar algunos paquetes [8], además de los ya instalados para emplear la parte simbólica desde Octave detallado en el apartado 5, pero es bastante rápido para realizar el cómputo.

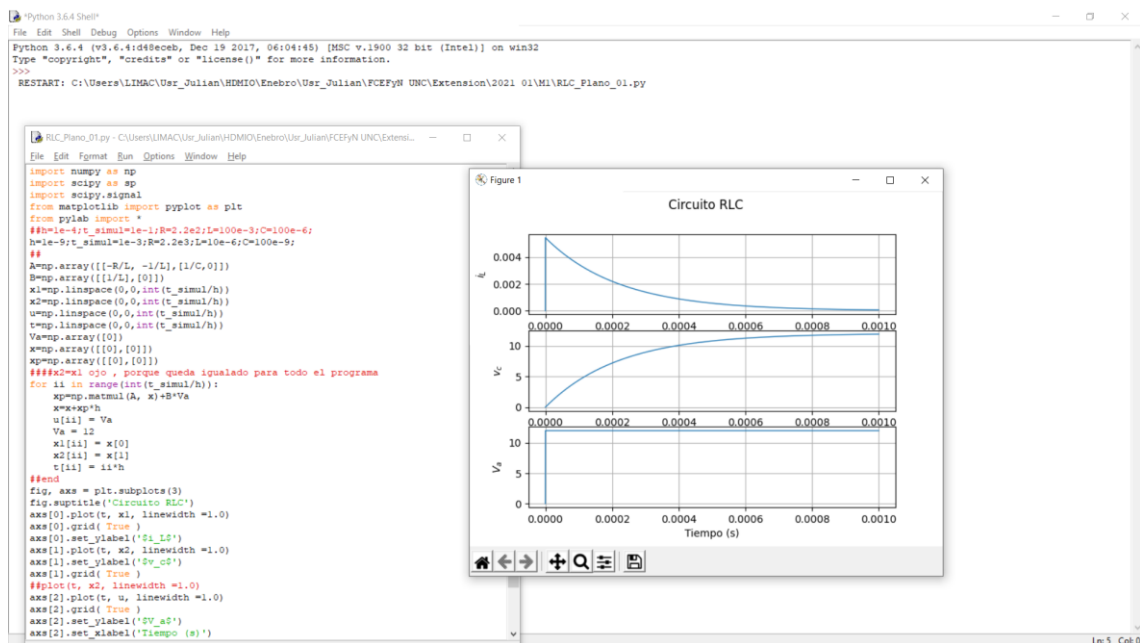


Fig. 12. Resultado de simulación numérica empleando Python [8].

```

import numpy as np
import scipy as sp
import scipy.signal
from matplotlib import pyplot as plt
from pylab import *
##h=1e-4;t_simul=1e-1;R=2.2e2;L=100e-3;C=100e-6;
h=1e-9;t_simul=1e-3;R=2.2e3;L=10e-6;C=100e-9;

```

```

##
A=np.array([[ -R/L, -1/L],[1/C,0]])
B=np.array([[1/L],[0]])
x1=np.linspace(0,0,int(t_simul/h))
x2=np.linspace(0,0,int(t_simul/h))
u=np.linspace(0,0,int(t_simul/h))
t=np.linspace(0,0,int(t_simul/h))
Va=np.array([0])
x=np.array([[0],[0]])
xp=np.array([[0],[0]])
###x2=x1 ojo , porque queda igualado para todo el programa
for ii in range(int(t_simul/h)):
    xp=np.matmul(A, x)+B*Va
    x=x+xp*h
    u[ii] = Va
    Va = 12
    x1[ii] = x[0]
    x2[ii] = x[1]
    t[ii] = ii*h
##end
fig, axs = plt.subplots(3)
fig.suptitle('Circuito RLC')
axs[0].plot(t, x1, linewidth =1.0)
axs[0].grid( True )
axs[0].set_ylabel('$i_L$')
axs[1].plot(t, x2, linewidth =1.0)
axs[1].set_ylabel('$v_c$')
axs[1].grid( True )
##plot(t, x2, linewidth =1.0)
axs[2].plot(t, u, linewidth =1.0)
axs[2].grid( True )
axs[2].set_ylabel('$V_a$')
axs[2].set_xlabel('Tiempo (s)')
##title('Motor CC')
grid( True )
show()

```

Tabla 8. Script para simulación de un circuito RLC en Python.

e. R

El programa R es de libre distribución, está creciendo en la comunidad de programación para aprendizaje automático y de inteligencia artificial. Se maneja por paquetes, que hay una gran cantidad y actualizada. En la Fig. 13 se muestra el aspecto del entorno integrado con una simulación del proceso dinámico que es un circuito RLC, y en [9] están los instructivos para instalarlo y mantenerlo.

Tiene varios paquetes de visualización de resultados [10] [11] [12] [13] [14].

La característica que presenta, respecto a los procesos dinámicos, es que una vez que se ha hecho el gráfico con un tiempo de simulación muy largo, y se repite el cómputo para un tiempo corto, el gráfico anterior no se borra. En la Tabla 9 se detalla el código empleado, teniendo en cuenta el directorio de trabajo para correr el programa y para guardar la impresión de la imagen, es por ello que difiere la captura de la pantalla mostrada en Fig. 13 y la gráfica impresa de la Fig. 14 por el script mostrado en Tabla 9.

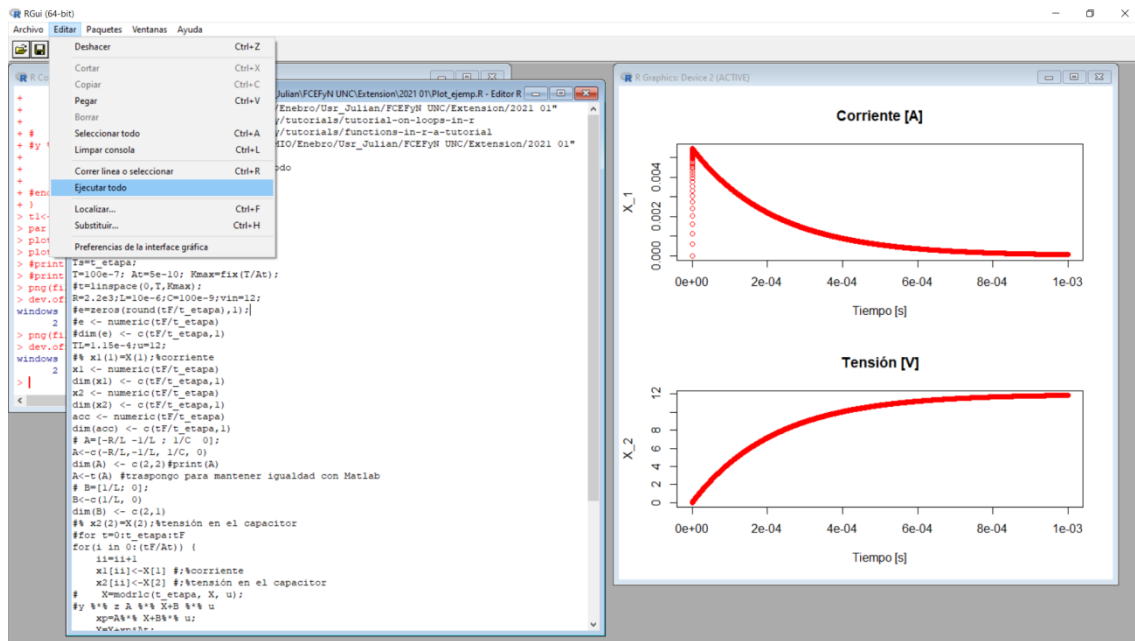


Fig. 13. Resultado de simulación numérica empleando R [9].

```
# "C:/Users/LIMAC/Usr_Julian/HDMI/O/Enebro/Usr_Julian/FCEfyN UNC/Extension/2022 01"
#https://www.datacamp.com/community/tutorials/tutorial-on-loops-in-r
#https://www.datacamp.com/community/tutorials/functions-in-r-a-tutorial
$ cd "C:/Users/LIMAC/Usr_Julian/HDMI/O/Enebro/Usr_Julian/FCEfyN UNC/Extension/2022 01"
$ R
rm(list=ls(all=TRUE)) #es BORRAR todo
#X=[0; 0];
X <- numeric(2)
dim(X) <- c(2,1)
ii=0;t_etapa=1e-9
tF=.5e-5;
Ts=t_etapa;
T=100e-7; At=5e-10; Kmax=round(T/At,0);
R=2.2e3;L=10e-6;C=100e-9;vin=12;
TL=1.15e-4;u=12;
#% x1(1)=X(1);%corriente
x1 <- numeric(tF/t_etapa)
dim(x1) <- c(tF/t_etapa,1)
x2 <- numeric(tF/t_etapa)
dim(x2) <- c(tF/t_etapa,1)
acc <- numeric(tF/t_etapa)
dim(acc) <- c(tF/t_etapa,1)
# A=[-R/L -1/L; 1/C 0];
A<-c(-R/L,-1/L, 1/C, 0)
dim(A) <- c(2,2)#print(A)
A<-t(A) #traspongo para mantener igualdad con Matlab
B<-c(1/L, 0)
dim(B) <- c(2,1)
for(i in 0:(tF/At)) {
  ii=ii+1
  x1[ii]<-X[1] #;%corriente
  x2[ii]<-X[2] #;%tensión en el capacitor
  xp=A%% X+B%% u;
  X=X+xp*At;
  acc[ii]<-u;
}
t1<-seq(0,tF,At) #seq(from, to, by= )
#para imprimir la imagen, sino es muy grande el archivo
#Acá hay que volver a asignar el directorio donde se almacena la imagen
#png(file="C:/Users/LIMAC/Usr_Julian/HDMI/O/Enebro/Usr_Julian/FCEfyN
UNC/Extension/2022 01/Imag_R_Tabla.png", width=1920, height=1080)
par(mfrow=c(2,1))
plot(t1,x1, main="Corriente [A]", col = "red", lwd = 1,xlab="Tiempo [s]", ylab="X_1")
plot(t1,x2, main="Tensión [V]", col = "red", lwd = 1,xlab="Tiempo [s]", ylab="X_2")
#print(t1)
#print(x1) dev.off() #Acá termina de imprimir
```

Tabla 9. Script para simulación de un circuito RLC en R.

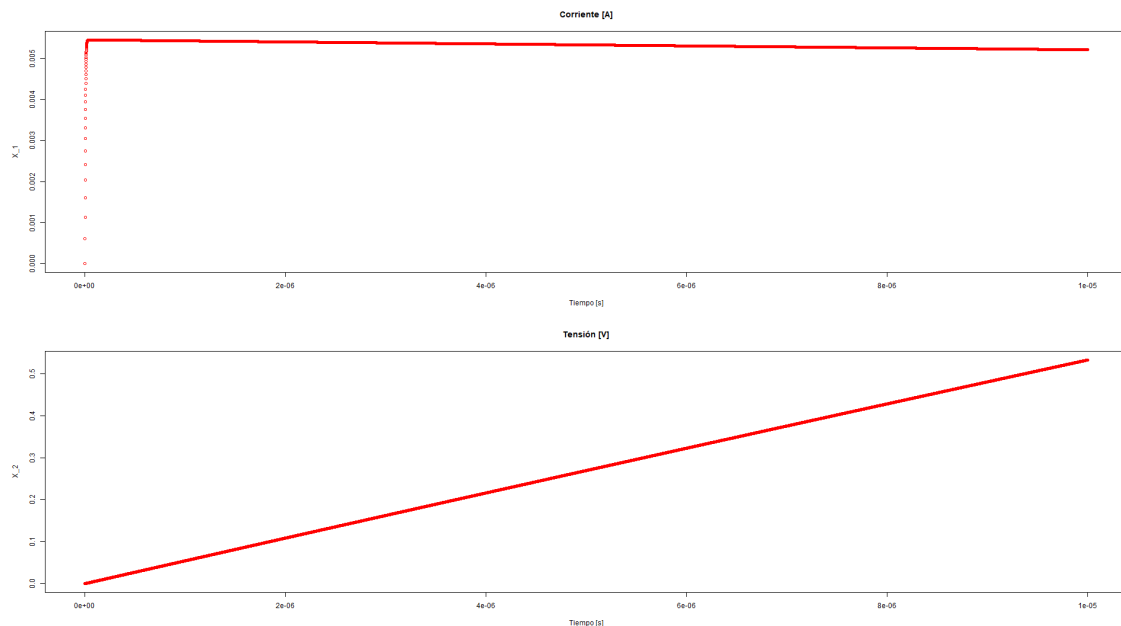


Fig. 14. Impresión del resultado del script de la Tabla 9 de simulación numérica empleando R [9].

5 Instructivo Symbolic en Octave

El Octave es una herramienta potente, y en lo que respecta a las herramientas simbólicas, usa los paquetes de Python OctsymPy. A continuación, se describe el proceso (sólo para Windows con el enlace de descarga del paquete que se ha instalado en [16]) en dos partes, una parte de Instalaciones, y una de Configuraciones.

a. Instalaciones

Va a ser necesario instalar Python, y su paquete simbólico. No es muy grande, así que es rápido.

1) De <https://www.python.org/> descargar Python 2.7.9 o superior para Windows e instalarlo con privilegios de Administrador.

3) Instalar [sympy](#) para [windows](#) que es la herramienta que tiene Python para manipulación simbólica.

```
>> pkg upgrade
```

 verifica si hay novedades en el paquete de Octave.

Octave emplea la aplicación [octsymPy](#) de Python que se instala como Sympy para Windows.

Instalar también mpmath <https://pypi.python.org/pypi/mpmath/0.17> para [Windows](#).

Debe descargarse el paquete [simbólico](#) para Octave para windows, y luego desde el prompt de Octave tipear

```
pkg install -forge symbolic-win-py-bundle-2.6.0
```

Se asume que Python y SymPy ya están instalados. Ahora es necesario conectarle las dependencias, para que Octave pueda llamar a las librerías que tiene Python. Se debe especificar el PATH y el PYTHONPATH.

```
>> pkg update
```

Sirve para verificar si hay actualizaciones en los paquetes instalados.

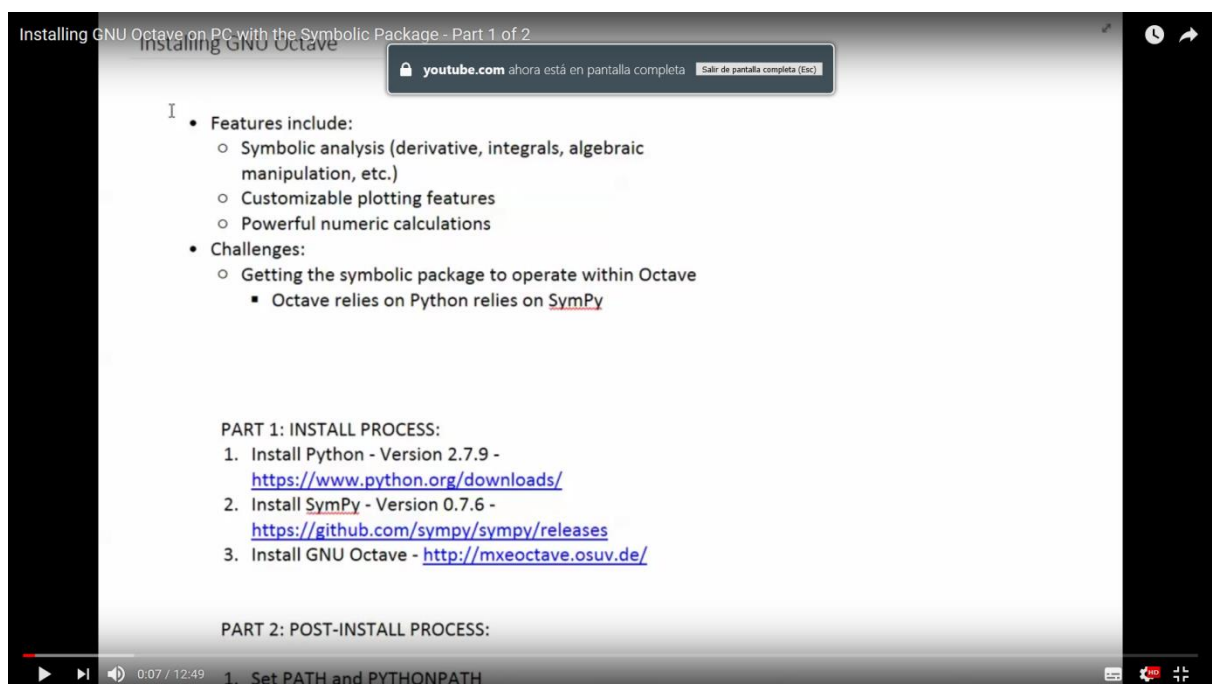
b. Configuraciones

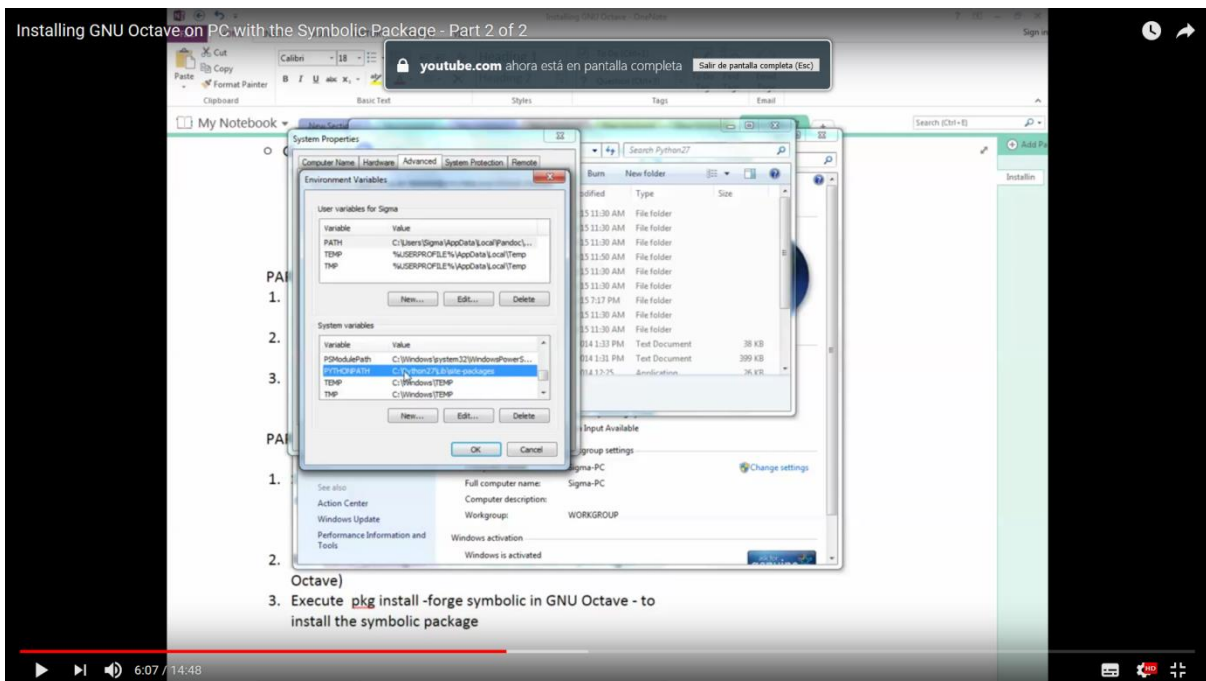
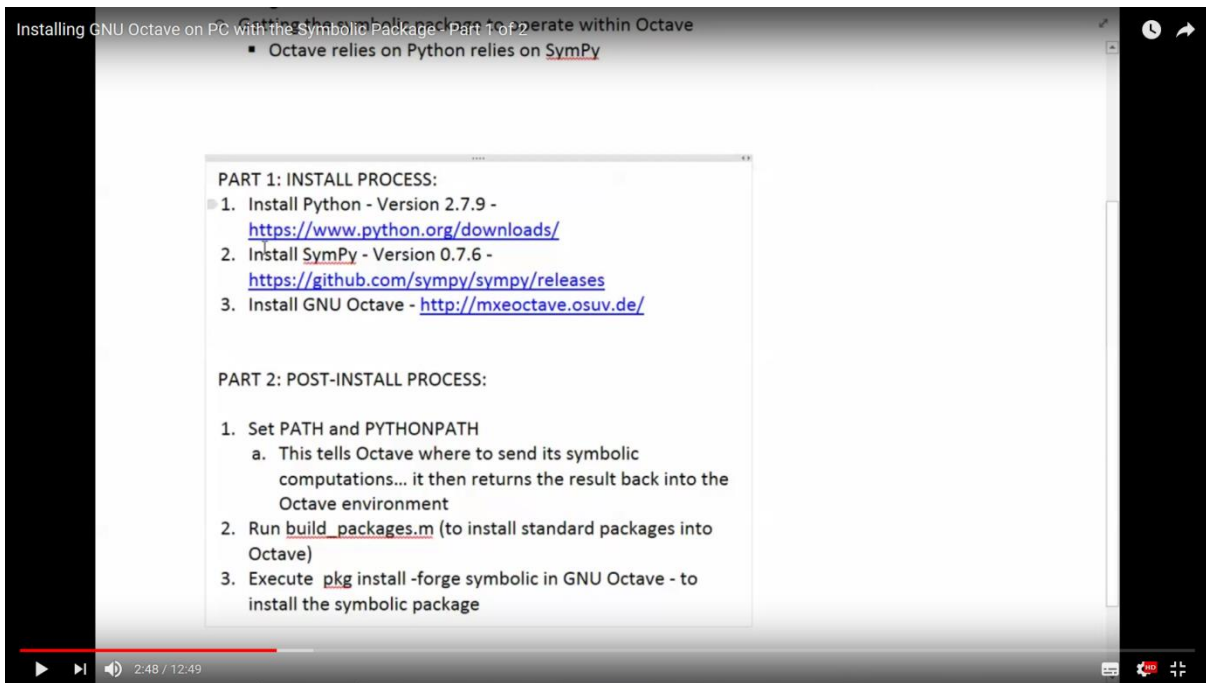
Video 1 <https://www.youtube.com/watch?v=KTvku6wvfQ4>

Video 2 <https://www.youtube.com/watch?v=8C9j2qlhTf0>

PYTHON PATH

C:\Users\Josef\AppData\Local\Programs\Python\Python36\





Nótese que debe reiniciarse el sistema una vez que se configura el Path.

6 Comparación de resultados y ajustes

Si se comparan los resultados empleando el Visual Studio, se obtienen resultados similares, pero considerando que la versión de Python sea la adecuada, se puede obtener un resultado como el de la Fig. 15 que muestra el resultado de un notebook.

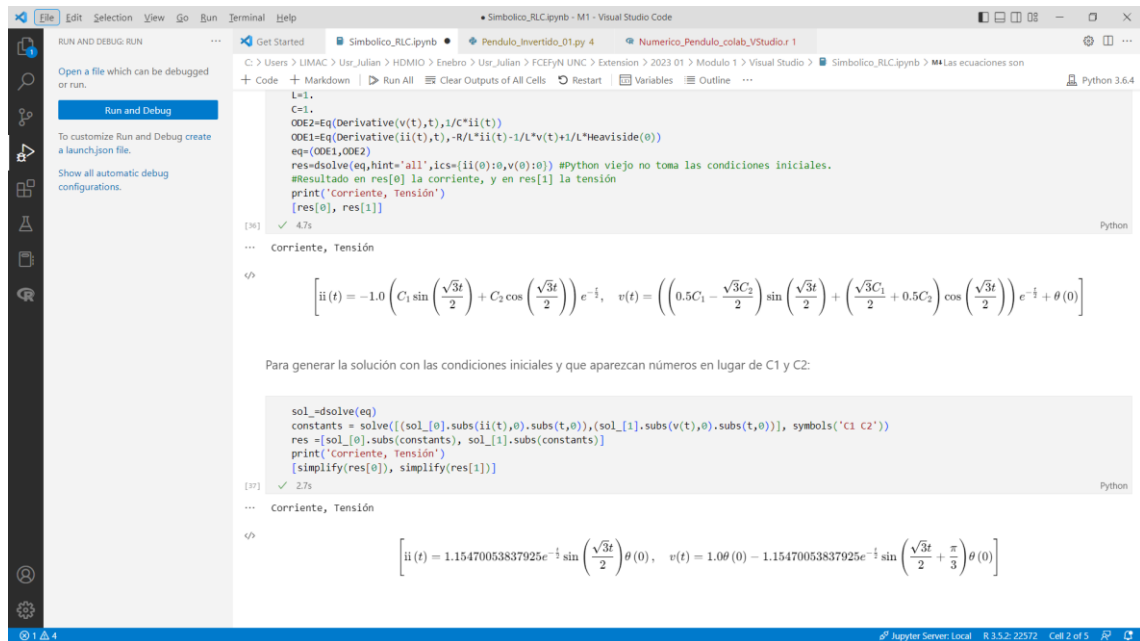
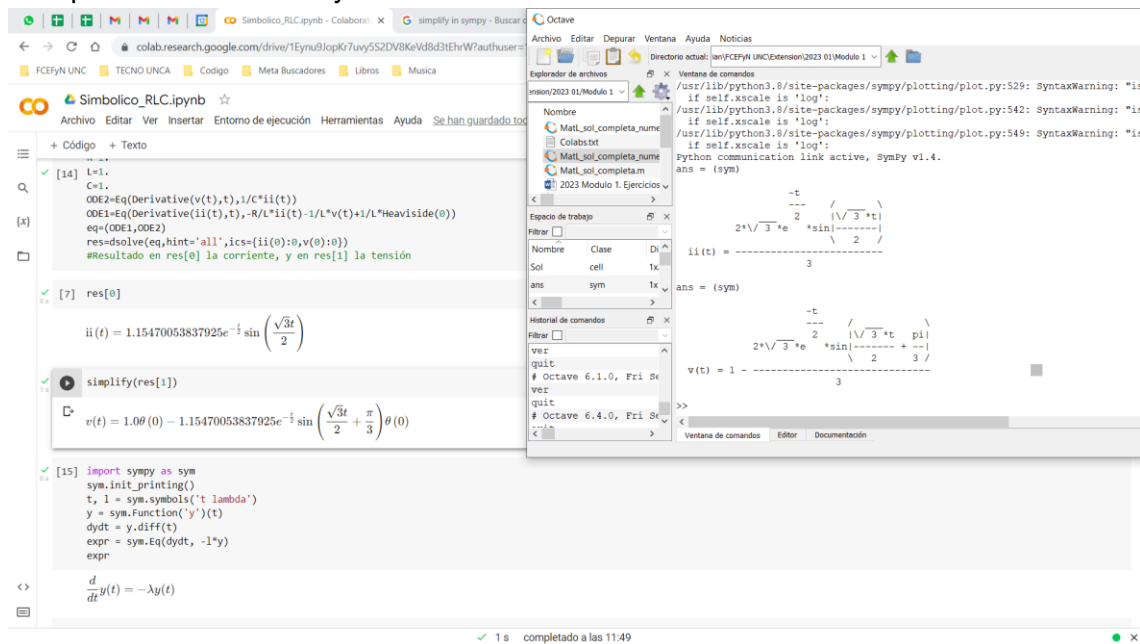
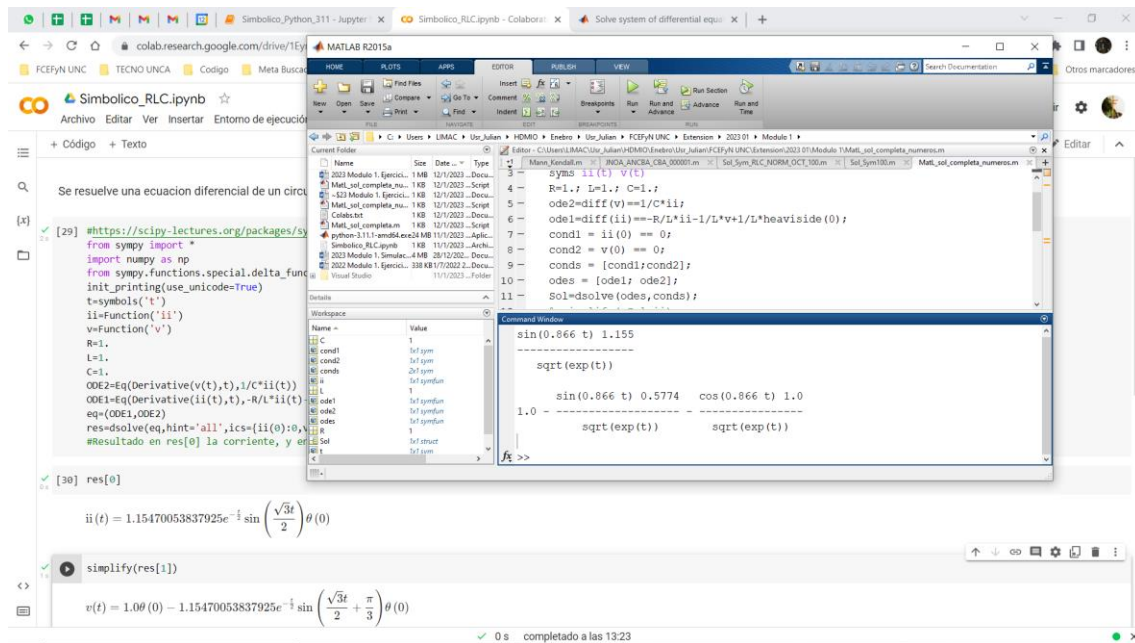


Fig. 15. Resultado de correr el script para Python 3.6.4., donde se deben hallar las constantes de integración.

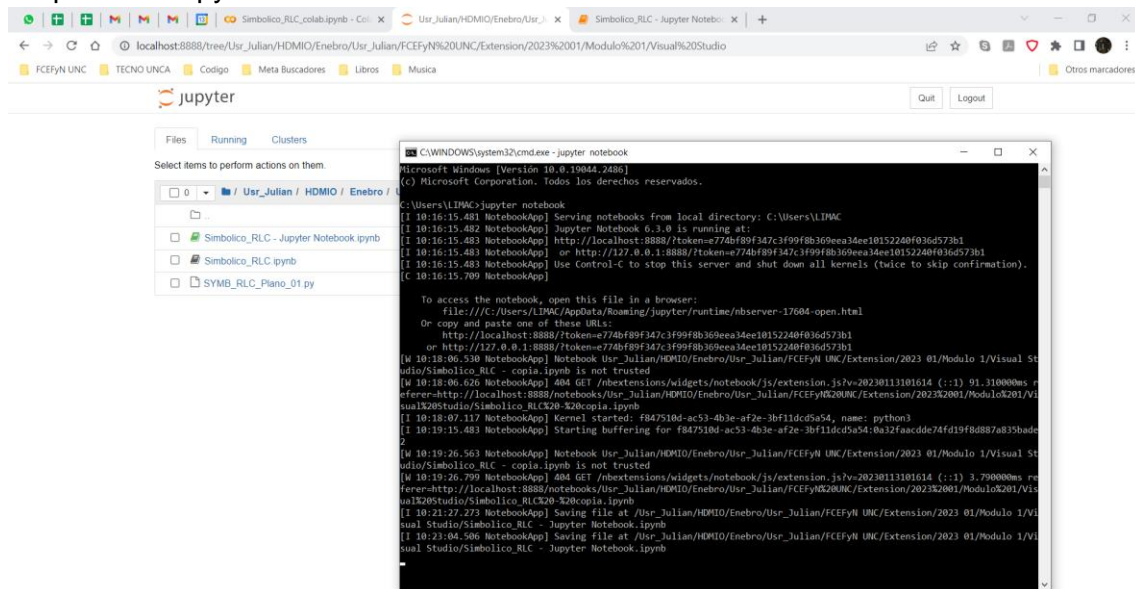
Comparación de Octave y Colabs:



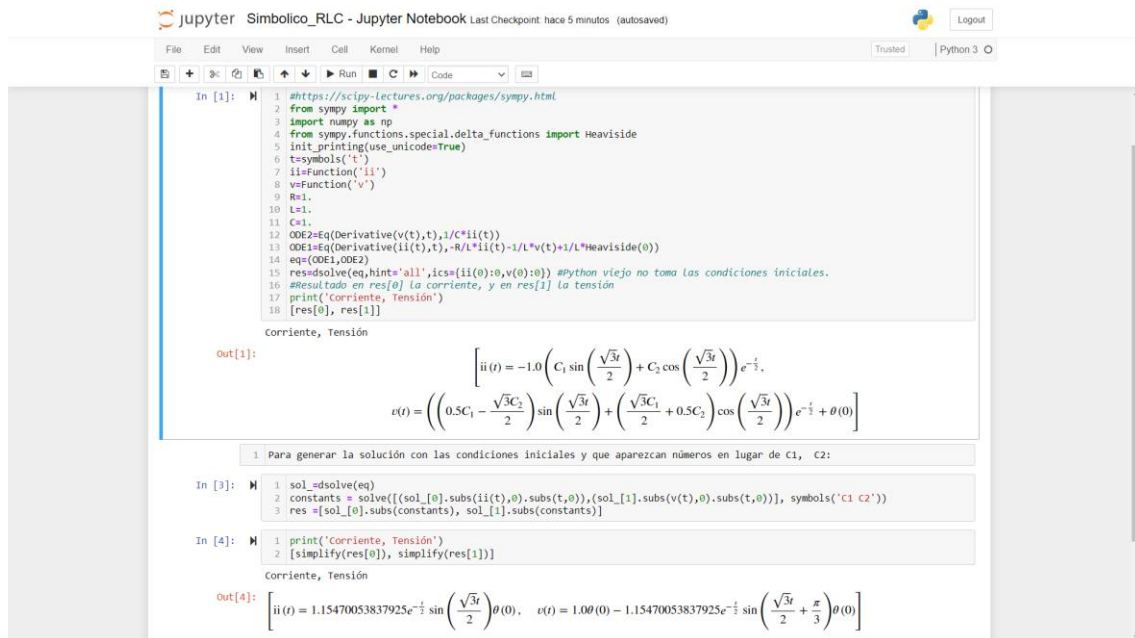
Igual que con Matlab:



Empleando Jupyter:



En la solapa correspondiente al navegador:



```

In [1]: 1 #https://scipy-lectures.org/packages/sympy.html
2 from sympy import *
3 import numpy as np
4 from sympy.functions.special.delta_functions import Heaviside
5 init_printing(use_unicode=True)
6 t=symbols('t')
7 ii=Function('ii')
8 v=Function('v')
9 R=1.
10 L=1.
11 C=1.
12 ODE2=Eq(Derivative(v(t),t),1/C*ii(t))
13 ODE1=Eq(Derivative(ii(t),t),-R/L*ii(t)-1/L*v(t)+1/L*Heaviside(0))
14 eq=(ODE1,ODE2)
15 res=dsolve(eq,hint='all',ics={ii(0):0,v(0):0}) #python viejo no toma las condiciones iniciales.
16 #Resultado en res[0] la corriente, y en res[1] la tensión
17 print('Corriente, Tensión')
18 [res[0], res[1]]

Corriente, Tensión
Out[1]: 
$$ii(t) = -1.0 \left( C_1 \sin\left(\frac{\sqrt{3}t}{2}\right) + C_2 \cos\left(\frac{\sqrt{3}t}{2}\right) \right) e^{-\frac{t}{2}},$$


$$v(t) = \left( \left( 0.5C_1 - \frac{\sqrt{3}C_2}{2} \right) \sin\left(\frac{\sqrt{3}t}{2}\right) + \left( \frac{\sqrt{3}C_1}{2} + 0.5C_2 \right) \cos\left(\frac{\sqrt{3}t}{2}\right) \right) e^{-\frac{t}{2}} + \theta(0)$$


1 Para generar la solución con las condiciones iniciales y que aparezcan números en lugar de C1, C2:

In [3]: 1 sol=dsolve(eq)
2 constants = solve([(sol[0].subs(ii(t),0).subs(t,0)), (sol[1].subs(v(t),0).subs(t,0))], symbols('C1 C2'))
3 res = [sol[0].subs(constants), sol[1].subs(constants)]

In [4]: 1 print('Corriente, Tensión')
2 [simplify(res[0]), simplify(res[1])]

Corriente, Tensión
Out[4]: 
$$ii(t) = 1.15470053837925e^{-\frac{t}{2}} \sin\left(\frac{\sqrt{3}t}{2}\right) \theta(0), \quad v(t) = 1.00(0) - 1.15470053837925e^{-\frac{t}{2}} \sin\left(\frac{\sqrt{3}t}{2} + \frac{\pi}{3}\right) \theta(0)$$


```

Nótese que hay diferencias en los resultados, ya que las versiones de programas empleados difieren, pero siempre puede lograrse verificar el resultado correcto.

Referencias:

- [1] <https://www.wolframalpha.com/>
- [2] <https://github.com/cbm755/octsympy>
- [3] <https://github.com/cbm755/octsympy/releases>
- [4] <https://es.wikipedia.org/wiki/FreeMat> , <http://freemat.sourceforge.net/>
- [5] <https://www.octave.org>.
- [6] <https://www.scilab.org/>
- [7] https://help.scilab.org/docs/6.1.0/en_US/index.html
- [8] <https://matplotlib.org/>
- [9] <https://CRAN.R-project.org>
- [10] <https://ggplot2.tidyverse.org/>
- [11] <https://cran.r-project.org/web/packages/colourpicker/index.html>
- [12] <https://jkunst.com/highcharter/>
- [13] <https://github.com/dreamRs/esquisse>
- [14] <https://plotly.com/>
- [15] <https://www.python.org/>
- [16] <https://drive.google.com/file/d/11T8NXh3aBqPHz6L28OesqASGreNysIU0/view>
- [17] https://github.com/Julianpucheta/OptimalControl/blob/main/ID_CHEN.ipynb