Resolviendo ecuaciones diferenciales ordinarias con symbolic math toolbox™ (Matlab) y SymPy (Python)

G. Ortigoza y R. I. Ponce de la Cruz Herrera

Instituto de Ingeniería, Universidad Veracruzana,
S.S. Juan Pablo II s/n. Costa Verde, 94294, Boca del río, Veracruz, México.

Received 10 March 2023; accepted 8 May 2023

En este trabajo se muestran soluciones de ecuaciones diferenciales ordinarias (EDOS) obtenidas mediante el uso de dos paquetes simbólicos: Symbolic Math Toolbox™ (Matlab) y SymPy (Python). Las instrucciones básicas para obtener soluciones de ambos paquetes son explicadas paso a paso, mediante un grupo de ejemplos provenientes de un curso tradicional de ecuaciones diferenciales ordinarias. Se incluyen ecuaciones diferenciales que se resuelven con métodos tales como: variables separables, ecuaciones lineales, coeficientes indeterminados, variación de parámetros, series de potencias, transformada de Laplace y soluciones numéricas. Mediante el cómputo simbólico realizado con estos paquetes es posible obtener la solución de sistemas lineales, así como la visualización del campo de direcciones de una ecuación diferencial o de un sistema no lineal de ecuacioines diferenciales. El aporte de este trabajo es proveer al lector con una guía práctica que le permite iniciar el estudio de las ecuaciones diferenciales asistido por Symbolic Math Toolbox™ o SymPy. Entre los beneficios del uso de estas herramientas computacionales en las prácticas de docencia y/o aprendizaje se muestra como el uso de cómputo simbólico o numérico, nos ahorra esfuerzo en el cómputo de cálculos tediosos; permitiendo enfocar la atención en ideas y conceptos importantes como: la relación entre el modelo matemático y su contraparte física, comportamiento asintótico y análisis cualitativo de las soluciones.

Descriptores: Ecuaciones diferenciales ordinarias; solución simbólica; Matlab; SymPy; Python.

This paper shows solutions of ordinary differential equations (EDOS) obtained by using two symbolic packages: Symbolic Math Toolbox™ (Matlab) and SymPy (Python). The basic instructions to obtain solutions of both packages are explained step by step, through a group of examples from a traditional ordinary differential equations course. Differential equations that are solved with methods such as: separable variables, linear equations, indeterminate coefficients, variation of parameters, power series, Laplace transform, and numerical solutions are included. By means of the symbolic computation carried out with these packages it is possible to obtain the solution of linear systems, as well as the visualization of the direction field of a differential equation or of a non-linear system of differential equations. The contribution of this work is to provide the reader with a practical guide that allows him to start the study of differential equations assisted by Symbolic Math Toolbox™ or SymPy. Among the benefits of using these computational tools in teaching and/or learning practices, it is shown how the use of symbolic or numerical computation saves us effort in computing tedious calculations; allowing to focus attention on important ideas and concepts such as: the relationship between the mathematical model and its physical counterpart, asymptotic behavior and qualitative analysis of the solutions.

Keywords: Ordinary differential equations; solution symbolic; Matlab; SymPy; python.

DOI: https://doi.org/10.31349/RevMexFis.20.020209

1. Introducción

Recientemente ha crecido el interés por incorporar herramientas computacionales a las prácticas de docencia e investigación. Desde calculadoras gráficas [1] hasta paquetes de algebra computacional tales como: Maple, Mathematica, Matlab, Maxima, SageMath y SymPy [2], han dado motivo a la publicación de obras que abordan la física, el cálculo diferencial, el álgebra lineal, y en particular las ecuaciones diferenciales con el apoyo de Maple [3] , Mathematica [4] y Matlab [5, 6]. Cabe resaltar que estos tres paquetes son comerciales y de uso privativo (licencias educativas con un promedio de 200 dólares) por lo que en general, su uso en Latinoamérica está restringido a personas o instituciones que pueden adquirir las licencias. Por otra parte Maxima, SageMath y Sympy son de acceso libre (filosofía de software libre) lo que las convierten en atractivas opciones para ser utilizadas en las practicas de docencia e investigación.

Entre los trabajos previos que buscan incoorporar el uso de paquetes computacionales en la enseñanza de las ecuaciones diferenciales ordinarias, podemos mencionar el de Lozada *et al*. [7] quienes realizaron una revisión sistemática de la literatura en las metodologías para enseñar y aprender ecuaciones diferenciales ordinarias. En su revisión identifican una transición de la enseñanza tradicional hacia un enfoque cualitativo y numérico con métodos interactivos, modelación y el uso de la tecnología. Enfatizan la importancia de la participación del estudiante en la construcción de su propio aprendizaje. Amangeldievna [8] considera que para motivar las actividades de aprendizaje de los estudiantes se deben organizar los temas de estudio en una forma visual e interactiva. Brandi y Garcia [9] buscan motivar a los alumnos de ingeniería mediante problemas para ser modelados con ecuaciones diferenciales ordinarias. Aquí el problema elegido por el alum-

no debe ser explorado para crear un modelo que será resuelto mediante herramientas computacionales (preferentemente de uso libre). Por otra parte Coelho *et al.* [10] muestran el uso de un paquete diseñado con Wolfram Mathematica que implementa de forma interactiva cuatro modelos de ecuaciones diferenciales ordinarias, así el alumno puede variar los parámetros y observar los cambios en las soluciones analíticas y en sus representaciones gráficas. Kwon [11] aborda los esfuerzos por adaptar la Educación Matemática Realística a los cursos de ecuaciones diferenciales ordinarias. ésta teoría se enfoca en una reinvención guiada a través de una matematización de las estrategias informales de solución de los alumnos y sus interpretaciones mediante sus experiencias en el contexto de problemas reales. Concluyen que en los recientes esfuerzos para reformar los contenidos de los cursos de ecuaciones diferenciales se ha incrementado el uso de la tecnología por computadora para incluir el análisis cualitativo y las soluciones numéricas. Con respecto al análisis cualitativo y el uso de un software dinámico, Guerrero *et al.* [12] concluyen que a los estudiantes les resulta significativa la representación gráfica de las soluciones y esto les permite interpretar y recuperar información a partir del estudio del campo de direcciones. En lo que concierne a la visualización y análisis gráfico de las soluciones, en sus trabajos Kwon [13], Habre [14], Peres [15] y West [16] reportan la gran utilidad de las visualizaciones de las trajectorias en el plano fase así como las soluciones en series de tiempo, lo cual permite analizar comportamientos importantes aún cuando las soluciones analíticas no existan.

En este trabajo se presenta el uso de symbolic Matlab toolbox (Matlab versión 2022b) y SymPy (versión Python 3.8.5) para la solución de ecuaciones diferenciales ordinarias. Estos paquetes son herramientas didácticas valiosas, ya que además de proveer de visualizaciones y cómputo simbólico cuentan con funciones específicas para la solución analítica y numérica de ecuaciones diferenciales ordinarias (dsolve). El objetivo de este trabajo es brindar al lector una introducción práctica a los comandos básicos de Matlab y Sympy para el estudio de ecuaciones diferenciales ordinarias (edos) y presentar mediante varios ejemplos, algunas de las ventajas de la aplicación de estos paquetes a la enseñanza de las ecuaciones diferenciales ordinarias. Este trabajo es en cierta forma, una complementación de anteriores trabajos donde se resolvieron ecuaciones diferenciales ordinarias con Maple y Mathematica [17], así como maxima [18]. Por ello, se abordó el mismo conjunto de ejemplos representativos proveniente de un curso clásico de ecuaciones diferenciales ordinarias. El trabajo se organiza de la siguiente manera: en la Sec. 2 se describe la sintaxis de los comandos básicos para la solución simbólica de EDOS en Matlab y SymPy; en la Sec. 3 se presenta mediante una serie de ejemplos, el uso de estos comandos para resolver EDOS, a la vez se agregan cuestionamientos y comentarios dirigidos a iniciar la reflexión de los instructores y estudiantes sobre algunos temas importantes y finalmente, en la Sec. 4 presentan conclusiones a este trabajo.

2. Comandos básicos para resolver ecuaciones diferenciales en Symbolic toolbox matlab y Simpy

En esta sección mostramos el uso de algunos comandos básicos para resolver de forma simbólica ecuaciones diferenciales ordinarias en Symbolic toolbox Matlab y SymPy. La herramienta Symbolic Toolbox Matlab provee una lista extensiva de funciones para la manipulación simbólica de expresiones y ecuaciones. El cálculo simbólico y la manipulación analítica, pueden combinarse con los métodos numéricos para hallar soluciones con la ventaja de olvidarse de los manipulaciones algebraicas engorrosas y enfocándose en los puntos importantes de la implementación numérica.

Originalmente la herramienta Symbolic Toolbox Matlab usaba el paquete de cómputo Maplesoft symbolic para realizar las operaciones simbólicas y pasaba el resultado a Matlab.

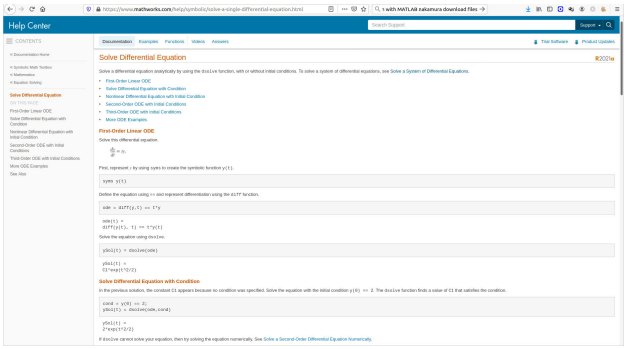

FIGURE 1. Symbolic toolbox ejemplos para resolver ecuaciones diferenciales del Help Center Matlab.

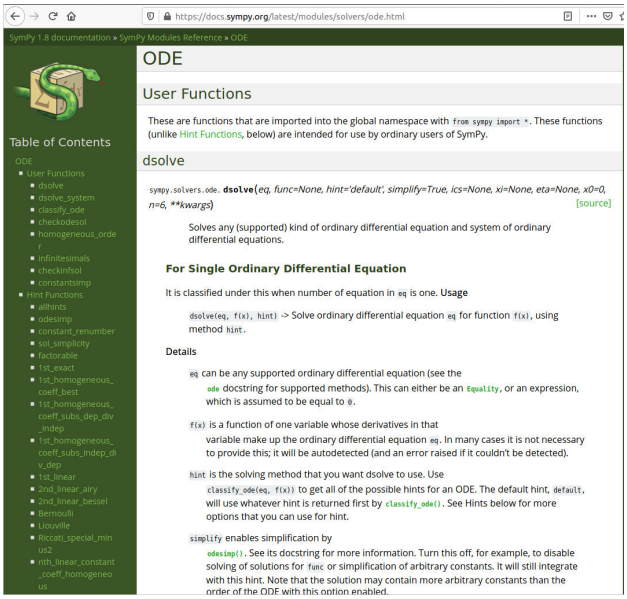

FIGURE 2. Página de documentación de SymPy 1.8, para resolver de forma simbólica ecuaciones diferenciales ordinarias.

Actualmente, a partir de 2008 la Symbolic Toolbox de Matlab usa el sistema de cómputo algebraico Mupad. En la Fig. 1 se muestra la página de internet de Help Center Matlab [19] con algunos ejemplos del uso de dsolve para resolver ecuaciones diferenciales ordinarias.

En la Fig. 2, vemos la página de documentación y referencias a funciones para la solución simbólica de ecuaciones diferenciales ordinarias con SymPy de Python [20]. Ambos programas cuentan con una función llamada dsolve la cual permite resolver de forma simbólica ecuaciones diferenciales ordinarias.

2.1. El comando dsolve en Symbolic toolbox matlab

Para resolver con Matlab usamos dsolve con la sintaxis:

```
S = dsolve(eqn)
S = dsolve(eqn,cond)
S = dsolve(___,Name,Value)
[y1,...,yN] = dsolve(___)
```

Los argumentos de entrada son `eqn`, la cual es una ecuación simbólica, `cond` es una expresión para la condición inicial, `Name,Value` es una pareja de argumentos para definir opciones de solución. Por ejemplo: 'ExpansionPoint' (punto donde se centra una solución en serie de potencias), 'IgnoreAnalyticConstraints' (realizar simplificaciones), 'Implicit' (hallar soluciones implícitas), etc. Mientras que las salidas son: `S, y1, \ldots ,yN`, las cuales representan la o las soluciones obtenidas. Si `dsolve` no puede hallar una solución de la ecuación diferencial, manda un mensaje de salida advirtiendo de ello.

2.2. El comando dsolve en SymPy Python

Para resolver con SymPy usamos dsolve con la sintaxis: `sympy.solvers.ode.dsolve(eq,func=None, hint='default',simplify=True,ics=None, xi=None, eta=None,x0=0,n=6,**kwargs)`. A continuación explicamos brevemente los argumentos de entrada, `eq` es una ecuación simbólica (puede ser definida como `Equality`, o una expresión igualada a cero). Donde `f(x)` es una función de una variable cuya derivada aparece en la ecuación diferencial, en la gran mayoría de casos no es necesario definirla pues se autodecta en la expresión de la ecuación simbólica. `hint` es el método de solución que el usuario desea que dsolve use para resolver la ecuación diferencial. Podemos usar `classify_ode(eq, f(x))` para obtener todas las posibles ayudas o sugerencias de los métodos usados para resolver la ecuación diferencial. simplify permite simplificar el resultado usando la función `odesimp()`. `xi` y `eta` son las funciones infinitesimales de la ecuación diferencial (son los infinitesimales del punto de transformaciones de Lie para las cuales la ecuación diferencial es invariante). Cuando no se especifican son calculadas automáticamente por la función `infinitesimals()`. Denotamos por `ics` las condiciones iniciales en la forma `{f(x0): x1, f(x).diff(x).subs(x, x2): x3}`. `x0` es el punto alrededor del cual la solución en serie de potencias será calculada. Donde `n` nos da el exponente hasta el cual la solución en serie de potencias será calculada.

3. Ejemplos de solución de ecuaciones diferenciales ordinarias

En esta sección mostramos cómo usar el comando dsolve en software Matlab o Symply para resolver diferentes tipos de problemas que se representan con ecuaciones diferenciales ordinarias. Denotamos con `>>` el prompt de Matlab y con `>>>` el prompt de python.

1. Ecuación logística:

$$\frac{dp}{dt} = ap - bp^2, \quad p(0) = p_0. \tag{1}$$

El problema de valor inicial para la ecuación logística es ampliamente usado como un modelo crecimiento poblacional. Donde a y b son los coeficientes vitales de la población $p(t)$ y p_0 es la población inicial. Iniciamos resolviendo con Matlab, para ello con el comando syms definimos variables simbólicas para los parámetros a, b, p_0 y la función $p(t)$:

```
>>syms a b p0 p(t)
```

A continuación, definimos la ecuación diferencial (ecuación simbólica) y la condición inicial:

```
>>ode = diff(p,t)= =a*p-b*p*p;
>>cond = p(0) = = p0;
```

y usamos dsolve para resolver la ecuación diferencial con su condición inicial:

`>>ySol(t) = dsolve(ode,cond)` obtenemos así:

```
ySol(t)=-(a*(tanh (atanh((a-2*b*p0)/a)-(a*t)/2))-1 ))/(2*b)
```

Definimos ahora los valores de los parámetros a, b y diferentes condiciones iniciales:

```
>>b=0.0001; a=0.03;p0=5;y1=subs(ySol(t));
p0=6;y2=subs(ySol(t));p0=7;y3=subs(ySol(t))
```

Note que `subs` se usa para substituir los valores de los parámetros en la solución y ahora procedemos a graficar nuestros resultados.

```
>>t=0:200;plot(t,subs(y1),t,subs(y2),t,subs(y3))
>>xlabel('tiempo t');ylabel('poblacion p(t)')
```

Veamos ahora cómo se resuelve con SymPy:
Primeramente importamos la librería SymPy a Python,

```
>>> import sympy as sp
```

Definimos las variables independiente y dependiente así como los parámetros de forma simbólica:

```
>>>t=sp.symbols('t')
>>>p=sp.Function('p')
>>>a=sp.symbols('a')
>>>b=sp.symbols('b')
>>>p0=sp.symbols('p0')
```

Ahora definimos la ecuación diferencial y la resolvemos con la condición inicial usando dsolve:

```
>>>ode=sp.Eq(sp.Derivative(p(t),t)-a*p(t)+b*p(t)*p(t),0)
>>>sol = sp.dsolve(ode,p(t),ics={p(0):p0})
```

Obteniendo así la solución:

```
Eq(p(t), a*exp(a*(t + log(b*p0/(-a + b*p0))/a))/
(b*(exp(a*(t + log(b*p0/(-a + b*p0))/a)) - 1)))
```

Definimos a continuación los valores de a, b y p0, y evaluamos las respectivas soluciones (una para cada condición inicial):

```
>>>constants={a:0.03,b:0.0001,p0:5}
>>>sol1=sol.subs(constants)
>>>constants={a:0.03,b:0.0001,p0:6}
>>>sol2=sol.subs(constants)
>>>constants={a:0.03,b:0.0001,p0:7}
>>>sol3=sol.subs(constants)
```

La función `lambdify` permite definir una función a partir de la expresión simbólica:

```
>>>func1 = sp.lambdify(t,sol1.rhs,'numpy')
>>>func2 = sp.lambdify(t,sol2.rhs,'numpy')
>>>func3 = sp.lambdify(t,sol3.rhs,'numpy')
```

A continuación, importamos los módulos `scipy, matplotlib` y `pyplot` para graficar nuestras soluciones:

```
>>>import scipy
>>>tt = scipy.arange(0,200,1)
>>>pp1 = func1(tt)
>>>pp2 = func2(tt)
>>>pp3 = func3(tt)
>>>import matplotlib as mpl
>>>import matplotlib.pyplot as plt
>>>plt.plot(tt,pp1,'r',tt,pp2,'b',tt,pp3,'g')
>>>plt.xlabel('tiempo t')
>>>plt.ylabel('Poblacion p(t)')
>>>plt.show()
```

En la Fig. 3 mostramos las soluciones obtenidas con Matlab y Python.

Para extender el análisis del modelo, en este ejemplo podríamos utilizar las gráficas de las soluciones obtenidas para diferentes valores de los parámetros y así mostrar relaciones entre la parte física del modelo y su contraparte matemática representada por la ecuación diferencial. Planteamientos tales como: ¿Cómo se comportan las soluciones cuando t tiende a infinito?, ¿para diferentes valores iniciales se cortan las diferentes soluciones? ¿Cómo se comportan las soluciones si el parámetro **a** es mucho menor que **b**? ¿y si **b** es mucho menor que **a**? ¿Cuál es la interpretación física de estos parámetros? Para ilustrar, la Fig. 4 obtenida con Matlab nos muestra comportamientos diferentes de las soluciones para diferentes valores de los parámetros **a** y **b**, si $a >>> b$ tenemos crecimiento exponencial, mientras que si $b >> a$ tenemos decaimiento hiperbólico.

Cabe mencionar que SymPy (Python) cuenta con las funciones `classify_ode` y `checkodesol`. Estas funciones son de gran utilidad desde el punto de vista didáctico. La primera de ellas nos da una clasificación del tipo de ecuación diferencial y con ello, el método que podría usarse para resolverla, mientras que la segunda verifica si una función propuesta es solución de la ecuación diferencial. Veamos cómo funcionan estas funciones, para ello usamos el caso particular de la ecuación logística

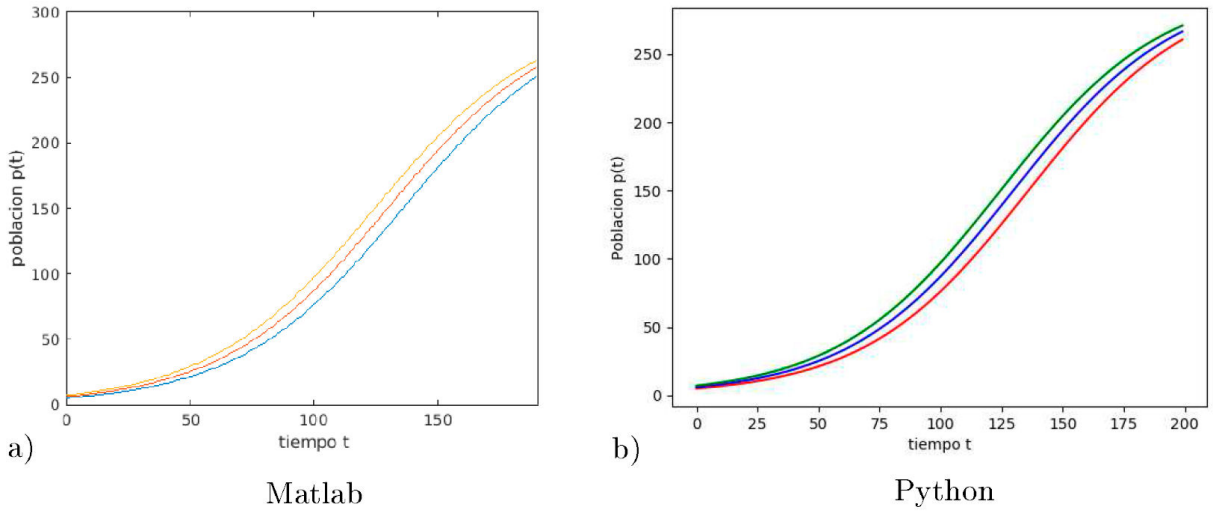

FIGURE 3. Gráficas de las soluciones de la ecuación logística para tres condiciones iniciales diferentes obtenidas con Matlab y Sympy.

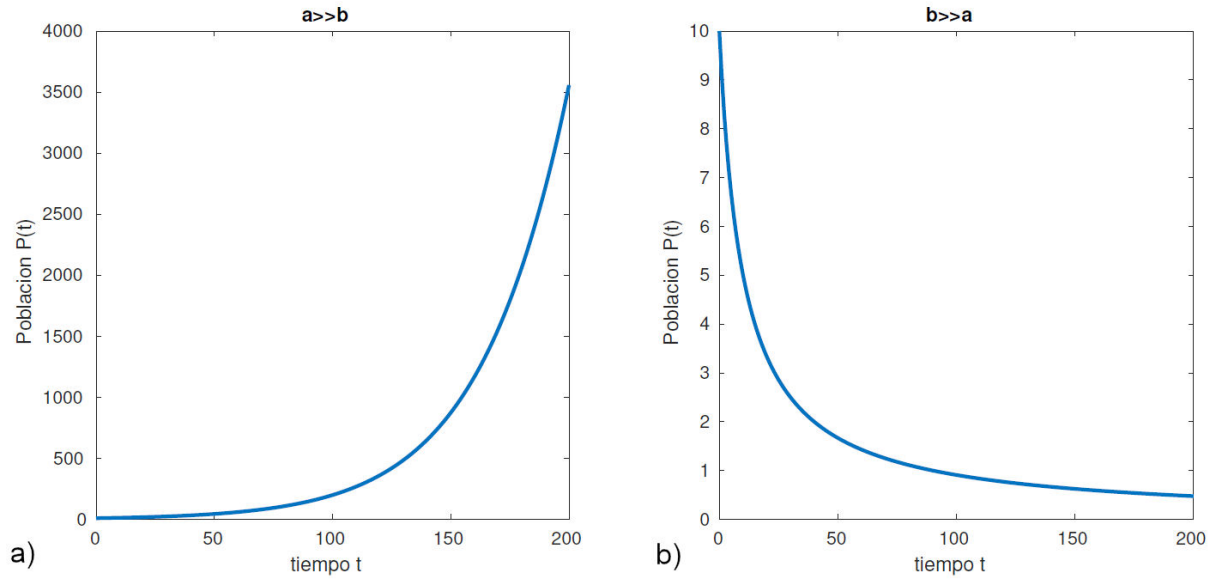

FIGURE 4. Comportamientos diferentes de las soluciones para diferentes valores de los parámetros **a** y **b**.

previamente definida: `>>> classify_ode(ode)`

```
(separable, Bernoulli, 1st_power_series, lie_group,
separable_Integral, Bernoulli_Integral)
```

Como observamos, la función anterior nos recomienda algunos de los métodos que podríamos usar como ayudas (hints) para resolver esta ecuación. Por otra parte:

```
>>> sol=dsolve(ode,p(t))
>>> sol
```

$$p(t) = \frac{ae^{a(C_1+t)}}{B(e^{a(C_1+t)} - 1)}$$

```
>>> checkodesol(ode,sol)
(True, 0)
```

Esta función nos permite verificar que la solución obtenida satisface a la ecuación diferencial (el resultado de la ejecución es True). Note que si definimos una función usando solo el denominador de nuestra solución:

```
>>>C1=sp.symbols('C1')
>>>sol1=b*(exp(a*(C1+t))-1)
```

```
>>>sol1                b*(exp(a*(C1 + t)) - 1)
>>> checkodesol(ode,sol1)
(False, b*(a + b**2*(1 - exp(a*(C1 + t)))**2))
```

Tal función no es solución pues NO SATISFACE la ecuación diferencial (el resultado de la ejecución es False).

2. Ecuación de Lagrange:

$$y = 2xy' + \log(y'). \tag{2}$$

Para resolver con Matlab usamos:

```
>>syms x y(x)
>>ode2 = y==2*x*diff(y,x)+log(diff(y,x)) ;
>>sol2(x) = dsolve(ode2)
```

y obtenemos:

```
sol2(x) =
log(((4*C1*x + 1)^(1/2) - 1)/(2*x)) + (4*C1*x + 1)^(1/2) - 1
log(-((4*C1*x + 1)^(1/2) + 1)/(2*x)) - (4*C1*x + 1)^(1/2) - 1
```

Mientras que para resolver con Python usamos:

```
>>>from sympy import *
>>>import sympy as sp
>>>x=sp.symbols('x')
>>>y=sp.Function('y')
>>>ode2=sp.Eq(log(sp.Derivative(y(x),x))+2*x*sp.Derivative(y(x),x)-y(x),0)
>>>sol2 = sp.dsolve(ode2,y(x))
```

Para obtener:

```
Eq(C1-y(x)-log(LambertW(2*x*exp(y(x))) + 2) + LambertW(2*x*exp(y(x))),0)
```

Note que la solución está dada en términos de la función de lambert W(z), la cual se define como la función inversa de wexp(w).

En este ejemplo podríamos graficar curvas soluciones para diferentes valores del parámetro C1, cuestionamientos tales como ¿hay curvas soluciones que corten el eje x? ¿hay curvas que corten el eje y? nos permiten motivar el dominio de existencia de las soluciones.

De esta forma, las siguientes instrucciones en Matlab:

```
C1=[-20,-15,-10,-5,0,5,10,15,20];
hold on
for i=1:9
y=log(((4*C1(i)*x + 1).^(1/2) - 1)./(2*x)) + (4*C1(i)*x + 1).^(1/2) - 1;
plot(x,y,'LineWidth',2)
```

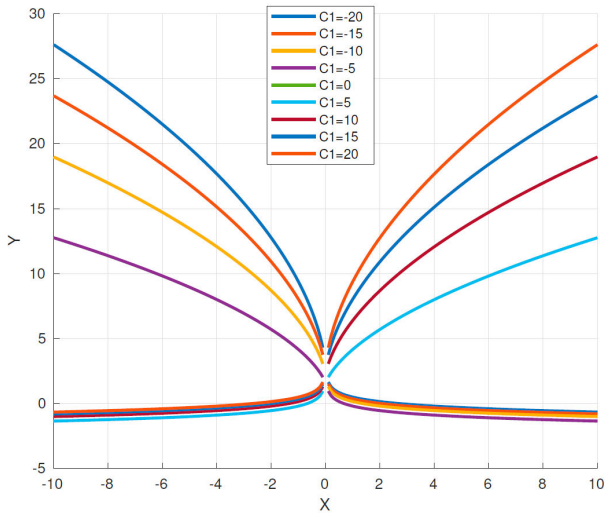

FIGURE 5. Curvas integrales para la ecuación de Lagrange $y = 2xy' + \log(y')$.

```
end
legend('C1=-20','C1=-15','C1=-10','C1=-5','C1=0','C1=5','C1=10','C1=15','C1=20')
grid on
```

permiten obtener curvas integrales para diferentes valores del parámetro C1. La Fig. 5 producida por los comandos anteriores en Matlab, muestra algunas curvas integrales de la ecuación de Lagrange. Con ayuda de la gráfica podemos contestar algunos de los cuestionamientos anterioremente expresados.

3. Resonancia:

$$y'' + 16y = 8\,\mathrm{sen}(4t), \quad y(0) = 1, \quad y'(0) = 0. \tag{3}$$

Este problema de valor inicial para una ecuación de segundo orden, no homogénea puede resolverse por el método del anulador y coeficientes indeterminados:
Resolvemos con Matlab:

```
>>syms t y(t)
>>Dy = diff(y);
>>ode = diff(y,t,2) +16*y== 8*sin(4*t);
>>cond1 = y(0) == 1;
>>cond2 = Dy(0) == 0;
>>conds = [cond1 cond2];
>>ySol(t) = dsolve(ode,conds);
```

Para obtener la solución:

```
ans =
cos(4*t)+(3*sin(4*t))/16-sin(12*t)/16- cos(4*t)*(t-sin(8*t)/8)
```

Y ahora graficamos con:

```
>>tt=0:.1:10;
>>plot(tt,subs(ySol(tt)))
>>xlabel('t');ylabel('y');grid on;
```

Para resolver con Python:

```
>>>from sympy import *
>>>import sympy as sp
>>>t=sp.symbols('t')
>>>y=sp.Function('y')
>>>ode3=sp.Eq(sp.Derivative(y(t),t,t)+16*y(t)-8*sin(4*t),0)
```

Resolvemos la ecuación diferencial:

```
>>>sol3 = sp.dsolve(ode3,y(t))
```

para obtener:

```
Eq(y(t), C2*sin(4*t) + (C1 - t)*cos(4*t))
```

Vamos ahora a definir las condiciones iniciales y substituirlas en la solución:

```
>>>const=sp.solve([sol3.rhs.subs(t,0)-1, sol3.rhs.diff(t,1).subs(t,0)-0])
>>>C1, C2 = sp.symbols('C1,C2')
>>>sol3=sol3.subs(const)
```

Definimos una función a partir de la solución simbólica obtenida:

```
>>>func1 = sp.lambdify(t,sol3.rhs,'numpy')
>>>import scipy
>>>xx = scipy.arange(0,10,.1)
>>>yy1 = func1(xx)
```

Y graficamos con los comandos:

```
>>>import matplotlib as mpl
>>>import matplotlib.pyplot as plt
>>>plt.plot(xx,yy1,'r')
>>>plt.xlabel('tiempo t')
>>>plt.ylabel('y(t)')
>>>plt.show()
```

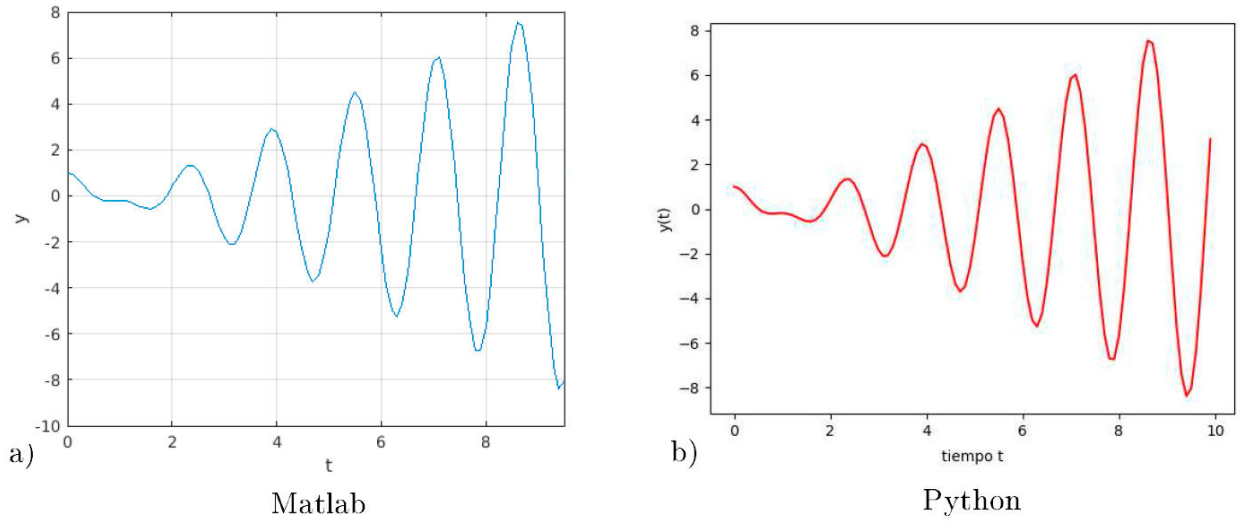

FIGURE 6. Gráficas de las soluciones de la Ec. (3) obtenidas con Matlab y Python.

La Fig. 6 muestra las soluciones obtenidas con Matlab y Python para este problema.

Con este ejemplo podemos cuestionar el comportamiento de la solución cuando t tiende a infinito. Más aún podríamos resolver:

$$y'' + 16y = 8\,\mathrm{sen}(wt), \quad y(0) = 1, \quad y'(0) = 0$$

para diferentes valores de $w = 3, 3.25, 3.75, 4.0$ y motivar el concepto de resonancia haciendo énfasis en la influencia del modelo matemático (parámetro w) sobre el modelo físico (oscilaciones crecientes). La frecuencia de la fuente o agente externo coincide con la frecuencia natural del sistema esto produce que las amplitudes de la solución crezcan en tiempo.

4. Ecuación de tercer orden:

$$y''' - y'' - y' + y = g(t). \tag{4}$$

Notamos que esta ecuación no homogénea tiene como fuente una función cualquiera g(t):
Para resolver en Matlab usamos:

```
>>syms t y(t) g(t)
>>ode4 = diff(y(t),t,3)-diff(y(t),t,2)-diff(y(t),t)+y(t)== g(t);
>>pretty(dsolve(ode4))
```

para obtener:

```
       /                      /
       |  exp(t) g(t)         |/ exp(-t) (g(t)+ 2t g(t)\
exp(-t) |  ----------- dt+exp(t)||------------------------|dt+
       /        4             / \        4              /

 C1 exp(t) + C3 exp(-t)+

          /
          |  exp(-t) g(t)
t exp(t)  |  ------------ dt + C2 t exp(t)
         /         2
```

La expresión anterior se torna un poco difícil de leer; para mejorar los formatos de salida de expresiones matemáticas, Matlab cuenta con la herramienta Live Editor [21]. La Fig. 7 nos muestra ésta herramienta en comparación con la salida tradicional en terminal.

Podemos notar que los términos que aparecen multiplicados por las constantes C1,C2,C3 corresponden a la solución general de la ecuación homogénea asociada, mientras que los términos (integrales de función resolvente multiplicada por la fuente) que contienen a $g(t)$ corresponden a una solución particular de la ecuación no homogénea.

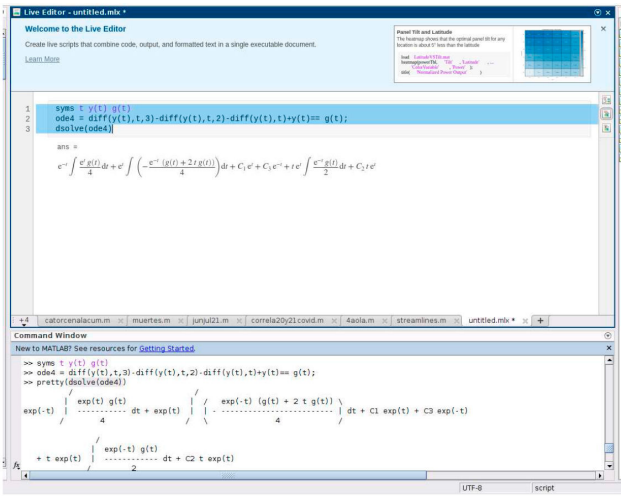

FIGURE 7. Entorno Live Editor Matlab.

Para resolver con Python usamos:

```
>>>from sympy import *
>>>import sympy as sp
>>>t=sp.symbols('t')
>>>y=sp.Function('y')
>>>g=sp.Function('g')
>>>ode4=sp.Eq(sp.Derivative(y(t),t,3)-sp.Derivative(y(t),t,t)-
sp.Derivative(y(t),t)+y(t)-g(t),0)
>>>sol4 = sp.dsolve(ode4,y(t))
```

Obteniendo así la solución

```
Eq(y(t), (C1 + Integral(g(t)*exp(t), t)/4)*exp(-t)
+ (C2 + t*(C3 + Integral(g(t)*exp(-t), t)/2) -
Integral((2*t + 1)*g(t)*exp(-t), t)/4)*exp(t)).
```

La expresión anterior, obtenida con Python en modo terminal se muestra un tanto difícil de leer. Externo a Python podemos instalar y usar Spyder, un entorno de desarrollo integrado y multiplataforma de código abierto (IDE) para programación científica en el lenguaje Python [22].

En la Fig. 8 se presenta como es el entorno spyder, y esto permite mejorar la salida de expresiones matemáticas, haciendo con esto más fácil su lectura y comprensión.

Dejando todo el cálculo engorroso al paquete simbólico podemos enfocarnos a resolver el problema con diferentes condiciones iniciales y motivar con ello la introducción de función de Green para un problema de valor inicial, más aún experimentar con diferentes fuentes externas g(t) y observar el efecto que producen sobre las soluciones.

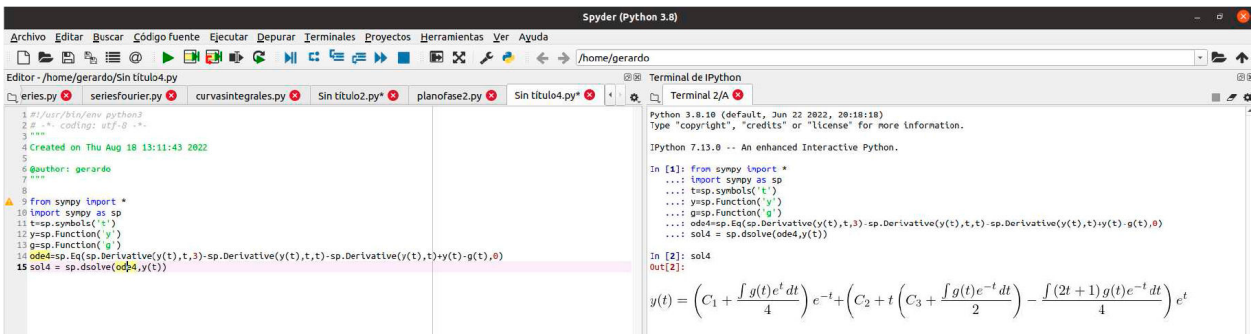

FIGURE 8. Entorno spyder donde las expresiones matemáticas lucen un formato más fácil de leer y comprender.

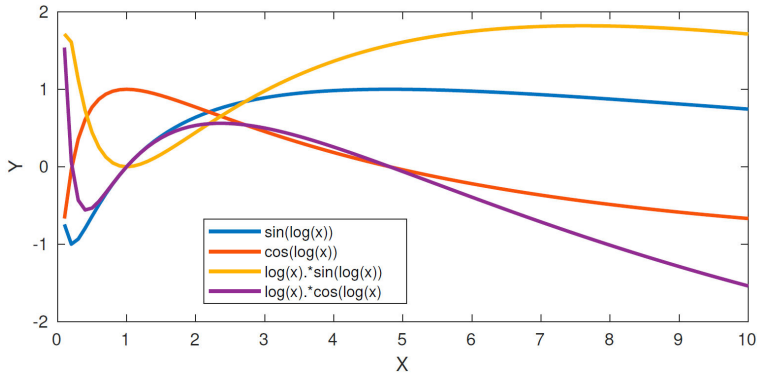

FIGURE 9. Soluciones linealmente independientes de la ecuación de Cauchy $x^4y^{(4)}(x) + 6x^3y^{(3)}(x) + 9x^2y''(x) + 3xy'(x) + y(x) = 0$.

5. Una ecuación homogénea de Cauchy-Euler de cuarto orden:

$$x^4y^{(4)}(x) + 6x^3y^{(3)}(x) + 9x^2y''(x) + 3xy'(x) + y(x) = 0. \tag{5}$$

En Matlab:

```
>>syms x y(x)
>> ode5 =x^4* diff(y(x),x,4)+6*x^3*diff(y(x),x,3)+
9*x^2*diff(y(x),2)+3*x*diff(y(x),x)+y(x)==0;
>> dsolve(ode5)
```

Para obtener:

```
C4*cos(log(x))+C2*sin(log(x))+C3*cos(log(x))*log(x)+ C1*sin(log(x))*log(x)
```

Mientras que en python usamos:

```
>>>from sympy import *
>>>import sympy as sp
>>>x=sp.symbols('x')
>>>y=sp.Function('y')
>>>ode5=sp.Eq(x**4*sp.Derivative(y(x),x,4)+6*x*x*x*x*sp.Derivative(y(x),x,3)+
9*x**2*sp.Derivative(y(x),x,2)+3*x*sp.Derivative(y(x),x)+y(x),0)
>>>sol5 = sp.dsolve(ode5,y(x))
```

Para obtener:

```
Eq(y(x),C3*sin(log(x))+C4*cos(log(x))+ (C1*sin(log(x))+C2*cos(log(x)))*log(x))
```

La literatura nos dice que un método de solución de las ecuaciones de Cauchy Euler implica la substitución $y = x^n$.
En Python podemos usar:

```
>>>n=sp.symbols('n')
>>>eq=simplify(ode5.subs(y(x),x**n))
>>>factor(eq)
```

Para obtener la ecuación auxiliar $x^n(n^2 + 1)^2$, esta expresión nos permite enfocarnos en las relación que guardan las expresiones de la solución y las raíces de la ecuación auxiliar (raíces complejas repetidas). Como una práctica de laboratorio de cómputo se puede pedir a los alumnos usar la transformación $t = \log(x)$ y mostrar como mediante cómputo simbólico se reduce la ecuación diferencial a una ecuación de coeficientes constantes.

La Fig. 9 obtenida con Matlab, muestra las cuatro soluciones linealmente independientes.

6. Un problema de valor inicial con deltas de Dirac como fuentes:

$$y''(t) - 4y'(t) + 4y(t) = 3\delta(t-1) + \delta(t-2), \quad y(0) = 1, \quad y'(0) = 1. \tag{6}$$

En Matlab usamos:

```
>>syms t y(t)
```

```
>>Dy = diff(y);
>>ode6 =diff(y(t),2)-4*diff(y(t),t)+4*y(t)==3*dirac(t-1)+dirac(t-2);
>>cond1 = y(0) == 1;
>>cond2 = Dy(0) == 1;
>>conds = [cond1 cond2];
>> ySol(t) = dsolve(ode6,conds);
```

y obtenemos:

```
exp(2*t)-t*exp(2*t)-exp(2*t)*(3*heaviside(t-1)*exp(-2)+2*heaviside(t-2)*exp(-4))+
t*exp(2*t)*(3*heaviside(t-1)*exp(-2)+heaviside(t-2)*exp(-4))
```

Por otra parte en Python resolvemos con:

```
>>>from sympy import *
>>>import sympy as sp
>>>t=sp.symbols('t')
>>>y=sp.Function('y')
>>>ode6=sp.Eq(sp.Derivative(y(t),t,2)-4*sp.Derivative(y(t),t)+
4*y(t),3*DiracDelta(t-1)+DiracDelta(t-2))
>>>sol6 = sp.dsolve(ode6)
>>>constants=solve([sol6.rhs.subs(t,0)-1,sol6.rhs.diff(t,1).subs(t,0)-1])
>>>sol6.subs(constants)
```

y obtenemos

```
Eq(y(t), (t*(exp(-4)*Heaviside(t-2)+3*exp(-2)*Heaviside(t-1)-1)-
2*exp(-4)*Heaviside(t-2) - 3*exp(-2)*Heaviside(t-1)+1)*exp(2*t)).
```

Note que este tipo de ecuación diferencial requiere para su solución, la técnica de transformada de Laplace (incluida en el comando dsolve). En este ejemplo, y a manera de práctica de laboratorio de cómputo, podriamos pedir a los alumnos experimentar moviendo las fuentes Delta($t - t_0$) para diferentes valores de t_0 y observar el efecto que causa en las soluciones.

7. Sistema masa-resorte:

$$mx''(t) + ke^{-\alpha t}x(t) = 0. \tag{7}$$

La ecuación anterior modela un resorte envejecido donde la constante decae con el tiempo. En Matlab usamos:

```
>>syms a m k t y(t)
>>ode7 =m*diff(y(t),2)+k*exp(-a*t)*y(t)==0;
>>dsolve(ode7)
```

Para obtener:

```
C1*besselj(0, (2*exp(-(a*t)/2)*(k/m)^(1/2))/a)+
C2*bessely(0, -(2*exp(-(a*t)/2)*(k/m)^(1/2))/a)
```

Mientras que en python usamos:

```
>>>from sympy import *
>>>import sympy as sp
>>>t=sp.symbols('t')
>>>y=sp.Function('y')
>>>m=sp.symbols('m')
>>>k=sp.symbols('k')
>>>a=sp.symbols('a')
>>>ode7=sp.Eq(m*sp.Derivative(y(t),t,t)+k*exp(-a*t)*y(t),0)
>>>sol7 = sympy.dsolve(ode7,y(t))
```

para obtener la ecuación:

```
Eq(y(t),C2*(k**2*t**4*exp(-2*a*t)/(24*m**2)-k*t**2*exp(-a*t)/(2*m)+1) +
C1*t*(-k*t**2*exp(-a*t)/(6*m) + 1) + O(t**6))
```

En este ejemplo hemos obtenido la solución expresada mediante funciones de Bessel (respuesta dada por Matlab) así como en serie de potencias (respuesta dada por Python). Note que mediante el uso de gráficas podríamos analizar la relación del

modelo matemático (constante del resorte) y el comportamiento de las soluciones. Para ello, basta con graficar las soluciones para diferentes valores del parámetro alfa y k fijo.

8. Problema de valor inicial para un sistema de ecuaciones lineales:

$$x'(t) = 3x(t) - 18y(t), \quad x(0) = 0, \tag{8}$$

$$y'(t) = 2x(t) + 9y(t), \quad y(0) = 1. \tag{9}$$

En Matlab resolvemos con:

```
>>syms t x(t) y(t)
>> ode1 = diff(x) == 3*x -18*y;
>>ode2 = diff(y) == 2*x + 9*y;
>>odes = [ode1; ode2]
>>cond1 = x(0) == 0;
>>cond2 = y(0) == 1;
>>conds = [cond1; cond2];
>> [xsol(t),ysol(t)] = dsolve(odes,conds)
```

y graficamos con:

```
>> t=0:0.01:4;
>> subplot(2,1,1)
>> plot(t,subs(xsol(t)),t,subs(ysol(t)))
>> xlabel('tiempo t')
>> legend('x','y')
>>subplot(2,1,2)
>>plot(subs(xsol(t)),subs(ysol(t)))
>>xlabel('x')
>>ylabel('y')
```

En Python resolvemos con:

```
>>>from sympy import *
>>>import sympy as sp
>>>t = sp.symbols('t')
>>> x, y = sp.symbols('x y', cls=Function)
>>> eqs = [Eq(x(t).diff(t), 3*x(t)-18*y(t)), Eq(y(t).diff(t),2*x(t)+9*y(t))]
>>> sol8=dsolve(eqs, [x(t), y(t)])
```

Para obtener la solución:

```
[Eq(x(t), (-18*C1*cos(3*sqrt(3)*t) - 18*C2*sin(3*sqrt(3)*t))*exp(6*t))
Eq(y(t), (C1*(-3*sqrt(3)*sin(3*sqrt(3)*t) + 3*cos(3*sqrt(3)*t)) +
C2*(3*sin(3*sqrt(3)*t) + 3*sqrt(3)*cos(3*sqrt(3)*t)))*exp(6*t))]
```

Ahora substituyendo las condiciones iniciales obtenemos la solución particular `>>>C1, C2 = sp.symbols('C1,C2')`

```
>>>constants = solve((sol8[0].subs(t,0).subs(x(0),0),
sol8[1].subs(t,0).subs(y(0),1)),{C1,C2})
>>>xsoln = expand(sol8[0].rhs.subs(constants))
>>>ysoln = expand(sol8[1].rhs.subs(constants))
```

Y graficamos con:

```
>>>import scipy
>>>tt=scipy.arange(0,4,0.01)
>>>funcxt=sp.lambdify(t,xsoln,'numpy')
>>>funcyt=sp.lambdify(t,ysoln,'numpy')
>>>xt=funcxt(tt)
>>>yt=funcyt(tt)
>>>import matplotlib as mpl
>>>import matplotlib.pyplot as plt
```

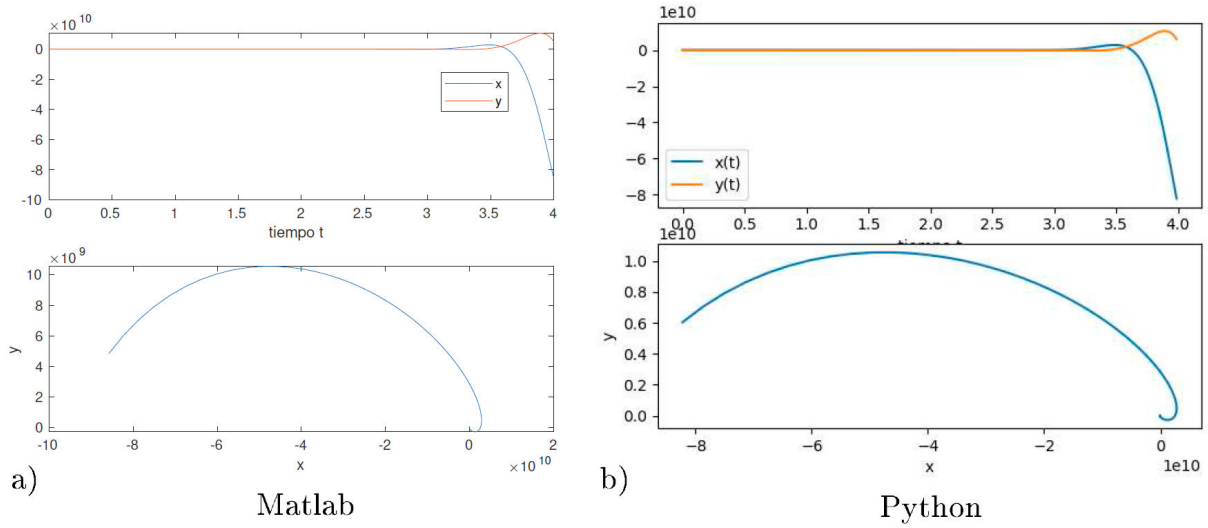

FIGURE 10. Gráficas de las soluciones del sistema lineal de ecuaciones diferenciales obtenidas con Matlab y SymPy.

```
>>>fig, (ax1, ax2) = plt.subplots(2)
>>>fig.suptitle('')
>>>ax1.plot(tt,xt,tt,yt)
>>>ax1.set(xlabel='tiempo t')
>>>ax1.legend(['x(t)', 'y(t)'])
>>>ax2.plot(xt, yt)
>>>ax2.set(xlabel='x',ylabel='y')
>>>plt.show()
```

La Fig. 10 nos muestra las gráficas de las soluciones de este sistema (x, y como funciones de t) así como la curva paramétrica $(x(t), y(t))$ definida por la solución del sistema lineal. La curva paramétrica parte del punto (0,1) girando en dirección contraria a las manecillas del reloj y alejándose del punto de partida, este modelo podría representar el movimiento de una partícula en el plano, comparando así el comportamiento anteriormente descrito con su contraparte matemática donde las soluciones involucran sinusoidales (giros) y exponenciales con exponente positivo (curva alejándose del punto de partida).

9. Ecuación de Airy:

$$y''(t) - ty(t) = 0. \tag{10}$$

Con Matlab resolvemos así:

```
>>syms t y(t)
>>ode9 = diff(y,t,2)-t*y == 0;
>>dsolve(ode9)
```

Para obtener una solución donde aparecen las funciones de Airy

```
C1*airy(0, -(t*(1 + 3^(1/2)*1i))/2) + C2*airy(2, -(t*(1 + 3^(1/2)*1i))/2)
```

podemos además hallar una solución en serie de potencias, por ejemplo tomamos condiciones iniciales en:

```
>>Dy = diff(y);
>> cond1 = y(0) == 1;
>> cond2 = Dy(0) == 1;
>>conds = [cond1 cond2];
>>sol9(t) = dsolve(ode9,conds);
>> simplify(taylor(sol9, t, 0))
```

Para obtener una solución en serie de potencias centradas en cero:

```
t^4/12 + t^3/6 + t + 1
```

O bien tomar condiciones iniciales en:

```
>>cond1 = y(1) == 1;
>>cond2 = Dy(1) == 1;
>>conds = [cond1 cond2];
>>sol9(t) = dsolve(ode9,conds,'ExpansionPoint',1);
```

Para obtener una solución en serie de potencias centradas en 1:

```
t + (t - 1)^2/2 + (t - 1)^3/3 + (t - 1)^4/8 + (t - 1)^5/24
```

Veamos ahora como se hace esto en python:

```
>>>from sympy import *
>>>import sympy as sp
>>>t = sp.symbols('t')
>>>y=sp.Function('y')
>>>ode9=sp.Eq(sp.Derivative(y(t),t,t)-t*y(t),0)
>>sol9=dsolve(ode9,y(t))
```

Obtenemos así una solución que incluye funciones de Airy

```
Eq(y(t), C1*airyai(t) + C2*airybi(t))
```

vamos ahora a hallar una solución en series de potencias centradas en cero:

```
>>>sols9 = sp.dsolve(ode9,hint='2nd_power_series_ordinary')
>>>constants=solve([sols9.rhs.subs(t,0)-1,sols9.rhs.diff(t,1).subs(t,0)-1])
>>>ysol=expand(sols9.rhs.subs(constants))
```

Y así obtenemos:

```
1 + t + t**3/6 + t**4/12 + O(t**6)
```

calculemos ahora una solución en series de potencias centradas en $t = 1$:

```
>>>s9=simplify(series(sol9.rhs,t,1))
>>>constants=solve([s9.subs(t,1)-1,s9.diff(t,1).subs(t,1)-1])
>>>ss9=s9.subs(constants)
>>>simplify(series(ss9,t,1,6))
```

Para obtener:

```
(t-1)**2/2+(t-1)**3/3+(t-1)**4/8 +(t-1)**5/24 +
t*airyaiprime(1)*airybi(1)/(airyaiprime(1)*airybi(1)-airyai(1)*airybiprime(1)) -
t*airyai(1)*airybiprime(1)/(airyaiprime(1)*airybi(1) - airyai(1)*airybiprime(1))+
O((t-1)**6,(t,1))
```

Ambos paquetes cuentan con las funciones de Airy, en este ejemplo hemos obtenido la solución en términos de estas funciones así como sus expresiones en series de potencias. En general no siempre es posible resolver una ecuación con coeficientes variables, este ejemplo permite motivar una introducción a las soluciones alrededor de puntos ordinarios, donde en algunos casos, es la única alternativa para obtener una solución.

La Fig. 11 obtenida con Matlab, nos muestra dos soluciones y_1 y y_2 de la ecuación de Airy obtenidas con `dsolve` que satisfacen $y_1(0) = 0, \quad y_1'(0) = 1, \quad y_2(0) = 1, \quad y_2'(0) = 0$. Podemos notar que para t negativo las soluciones se comportan como las soluciones oscilantes de $y'' + y = 0$, mientras que para t positivo se comportan como las soluciones exponenciales de $y'' - y = 0$.

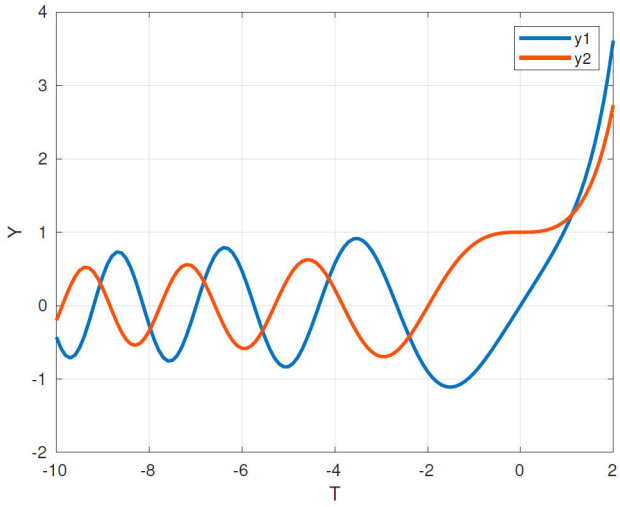

FIGURE 11. Soluciones de la ecuación de Airy.

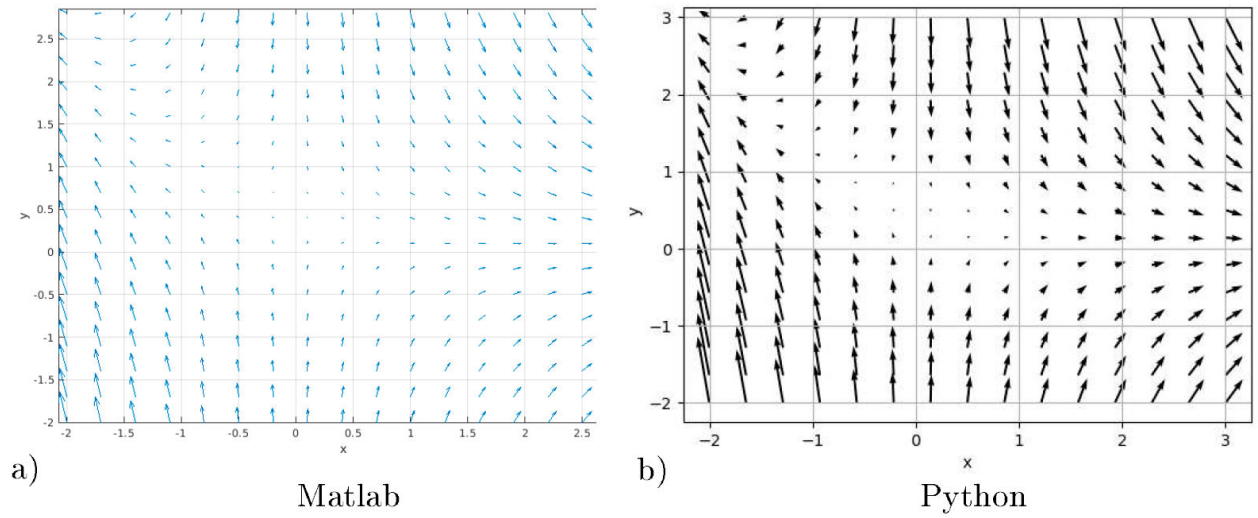

FIGURE 12. Gráficas de los campos de direcciones de una ecuación diferencial obtenidas con Matlab y Python.

10. Campo de direcciones de una ecuación diferencial:

$$x'(t) = e^{-t} - 2x(t). \tag{11}$$

Para graficar en Matlab el campo de direcciones de una ecuación diferencial, definimos primero una malla y usamos el comando quiver.

```
>>[x,y]=meshgrid(-2:0.3:3,-2:.3:3);
>>quiver(x,y,x,exp(-x)-2*y)
>>xlabel('x')
>>ylabel('y')
>>grid on
>>axis tight
```

Veamos ahora como se grafica el campo de direcciones usando python: `>>>import numpy as np`

```
>>>import matplotlib as mpl
>>>import matplotlib.pyplot as plt
>>>x,y = np.meshgrid(np.linspace(-2,3,15),np.linspace(-2,3,15))
>>>plt.quiver(x,y,x,np.exp(-x)-2*y)
>>>plt.grid('on')
>>>plt.xlabel('x')
>>>plt.ylabel('y')
>>>plt.show()
```

En la Fig. 12 mostramos los campos de direcciones obtenidos tanto con Matlab como con Python.

Ambos paquetes permiten la visualización del campo de direcciones, de esta manera podemos realizar análisis cualitativos de las soluciones. La solución por computadora nos permite responder las siguientes preguntas ¿qué trayectoria seguiría una partícula colocada en un punto (x_0, y_0) bajo la acción del campo de direcciones?

Más aún, con el apoyo de las gráficas podemos motivar el comportamiento asintótico de las soluciones con cuestionamientos tales como: ¿cuál es el comportamiento de las curvas soluciones cuando t tiende a infinito? ¿depende este comportamiento de la condición inicial elegida?

11. Un sistema no lineal (depredador -presa):

$$x'(t) = x(t)(1 - x(t) - y(t)), \tag{12}$$

$$y'(t) = y(t)(0.75 - 0.5x(t) - y(t)). \tag{13}$$

Las ecuaciones de Lotka-Volterra son un par de ecuaciones diferenciales de primer orden no lineales que se usan para describir dinámicas de sistemas biológicos en el que dos especies interactúan, una como presa y otra como depredador.

En este ejemplo además de visualizar las curvas integrales del sistema, vamos a analizar de forma analítica los puntos críticos y clasificarlos (como nodos inestables, asintóticamente estable o puntos silla).

Con Matlab:

```
>>[x,y] = meshgrid(-0.5:0.01:1.5);
>>u = x.*(1-x-y);
>>v = y.*(0.75-0.5*x-y);
>>l = streamslice(x,y,u,v);
>>X0=[0 0 1 0.5];
>>Y0=[0 0.75 0 0.5];
>>hold on
>>plot(x0,Y0,'ro','MarkerFaceColor','r')
>>xlabel('X');ylabel('Y');
>>title('Plano fase para un sistema no lineal')
>>grid on;axis tight
```

Ahora resolvemos para hallar los puntos criticos:

```
>>S=solve([x*(1-x-y)==0,y*(.75-0.5*x-y)==0],[x,y])
>> [S.x(1) S.y(1)]

ans =

[0, 0]

>> [S.x(2) S.y(2)]

ans =

[1, 0]

>> [S.x(3) S.y(3)]

ans =

[0, 3/4]

>> [S.x(4) S.y(4)]

ans =

[1/2, 1/2]
```

Calculamos el jacobiano:

```
>>ja=jacobian([x*(1-x-y),y*(.75-0.5*x-y)],[x,y])
```

Y a continuación los valores propios del jacobiano evaluado en los puntos críticos:

```
>> eig(subs(ja, [x, y], [S.x(1),S.y(1)]))

ans =

3/4
  1

>> eig(subs(ja, [x, y], [S.x(2),S.y(2)]))

ans =
```

```
 -1
1/4

>> eig(subs(ja, [x, y], [S.x(3),S.y(3)]))

ans =

-3/4
 1/4

>> eig(subs(ja, [x, y], [S.x(4),S.y(4)]))

ans =
- 2^(1/2)/4 - 1/2
  2^(1/2)/4 - 1/2
```

Veamos ahora como se realizan los mismos cálculos pero con Python: Para graficar las curvas soluciones:

```
>>>from pylab import *
>>>nodos = np.array([[0, 0, 1, 0.5], [0, 0.75,  0, 0.5]])
>>>xlabel('X')
>>>ylabel('Y')
>>>title('Plano Fase para un sistema no lineal')
>>>plot(nodos[0], nodos[1], 'ro')
>>>xvalues,yvalues=meshgrid(arange(-.5,1.5,0.001), arange(-0.5,1.5,0.001))
>>>xdot = xvalues*(1-xvalues-yvalues)
>>>ydot = yvalues*(0.75-0.5*xvalues-yvalues)
>>>streamplot(xvalues, yvalues, xdot, ydot)
>>>grid();
>>>show()
```

Hallamos ahora de forma analítica los puntos críticos:

```
>>>import sympy as sp
>>>x, y = sp.symbols('x y')
>>>pts=sp.solve([x*(1-x-y),y*(0.75-0.5*x-y)],{x,y})
>>> pts[0]
{x: 0.0, y: 0.0}
>>> pts[1]
{x: 0.0, y: 0.750000000000000}
>>> pts[2]
{x: 0.500000000000000, y: 0.500000000000000}
>>> pts[3]
{x: 1.00000000000000, y: 0.0}
```

Calculamos la matriz Jacobiana del sistema:

```
>>>X=sp.Matrix([x*(1-x-y),y*(0.75-0.5*x-y)])
>>>Y=sp.Matrix([x,y])
>>>A=X.jacobian(Y)
```

Evaluamos ahora la matriz Jacobiana en los puntos críticos y calculamos los valores propios para clasificar los puntos críticos:

```
>>> A.subs(pts[0]).eigenvects()
[(0.750000000000000, 1, [Matrix([
[  0],
[1.0]])]), (1.00000000000000, 1, [Matrix([
[1.0],
[  0]])])]
```

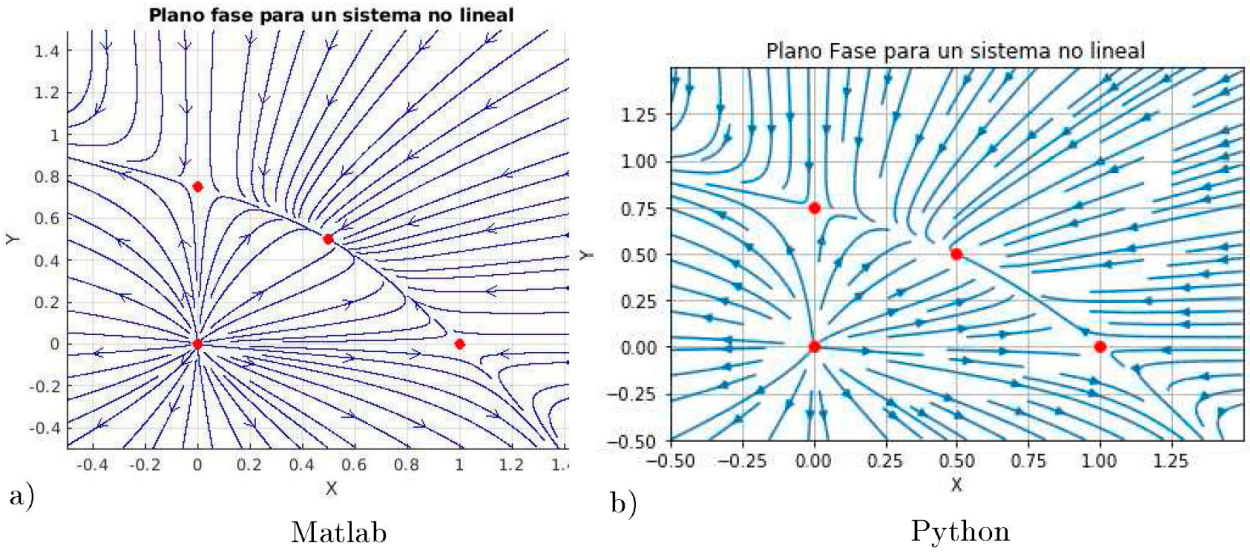

FIGURE 13. Gráficas de los curvas soluciones del sistema no lineal depredador presa obtenidas con Matlab (izquierda) y Python.

```
A.subs(pts[1]).eigenvects()
[(-0.750000000000000, 1, [Matrix([
[  0],
[1.0]])]), (0.250000000000000, 1, [Matrix([
[-2.66666666666667],
[              1.0]])])]

A.subs(pts[2]).eigenvects()
[(-0.853553390593274, 1, [Matrix([
[1.4142135623731],
[            1.0]])]), (-0.146446609406726, 1, [Matrix([
[-1.4142135623731],
[             1.0]])])]

A.subs(pts[3]).eigenvects()
[(-1.00000000000000, 1, [Matrix([
[1.0],
[  0]])]), (0.250000000000000, 1, [Matrix([
[-0.8],
[ 1.0]])])]
```

La Fig. 13 nos muestra curvas integrales (trayectorias que seguiría una partícula bajo el efecto del campo de direcciones definidas por el lado derecho del sistema diferencial), las curvas convergen en (0.5,0.5) pues se trata de un nodo asintóticamente estable, las curvas divergen de (0,0) que es un nodo inestable, mientras que los puntos (0,0.75) y (1,0) son puntos silla ya que hay curvas que se acercan y se alejan de ambos puntos. En este ejemplo los cálculos analíticos son corroborados con los gráficos obtenidos haciendo más clara la interpretación de puntos estables, inestables y puntos silla.

12. Problema de valor inicial para una ecuación no lineal (solución numérica):

$$y'(t) = \text{sen}(y^2(t)), \quad y(0) = 1. \tag{14}$$

En este caso en particular, la función dsolve en ambos paquetes no produce una solución, por ejemplo al usar Matlab obtenemos:

```
>> syms t y(t)
>>ode = diff(y,t) == sin(y^2);
>>dsolve(ode,y(0) == 1)
Warning: Unable to find symbolic solution.
```

```
ans =

[ empty sym ]
```

Como observamos no encontramos una solución analítica. Sin embargo, aún podemos usar las opciones con que cuentan ambos paquetes para obtener soluciones numéricas.
Con Matlab escribimos:

```
>>V = odeToVectorField(ode);
>>F = matlabFunction(V,'vars',{'t','Y'});
>>sol = ode45(F,[0 5],1);
>>x = linspace(0,5,100);
>>y = deval(sol,x,1);
>>plot(x,y)
```

Para resolver con Python:

```
>>> import numpy as np
>>> from scipy.integrate import odeint
>>> import matplotlib.pyplot as plt
>>> sin = np.sin
```

Definimos ahora la función que define el lado derecho de la ecuación diferencial $dy/dt = f(y,t)$

```
>>> def model(y,t):
...   dydt=sin(y*y)
...   return dydt
...
>>> y0 = 1
>>> t=np.linspace(0,20)
```

Resolvemos numéricamente con odeint:

```
>>> y=odeint(model,y0,t)
>>> plt.plot(t,y)
>>> plt.xlabel('time')
>>> plt.ylabel('y(t)')
>>> plt.show()
```

La Fig. 14 nos muestra las soluciones numéricas obtenidas con Matlab y Python para esta ecuación diferencial no lineal de primer orden.

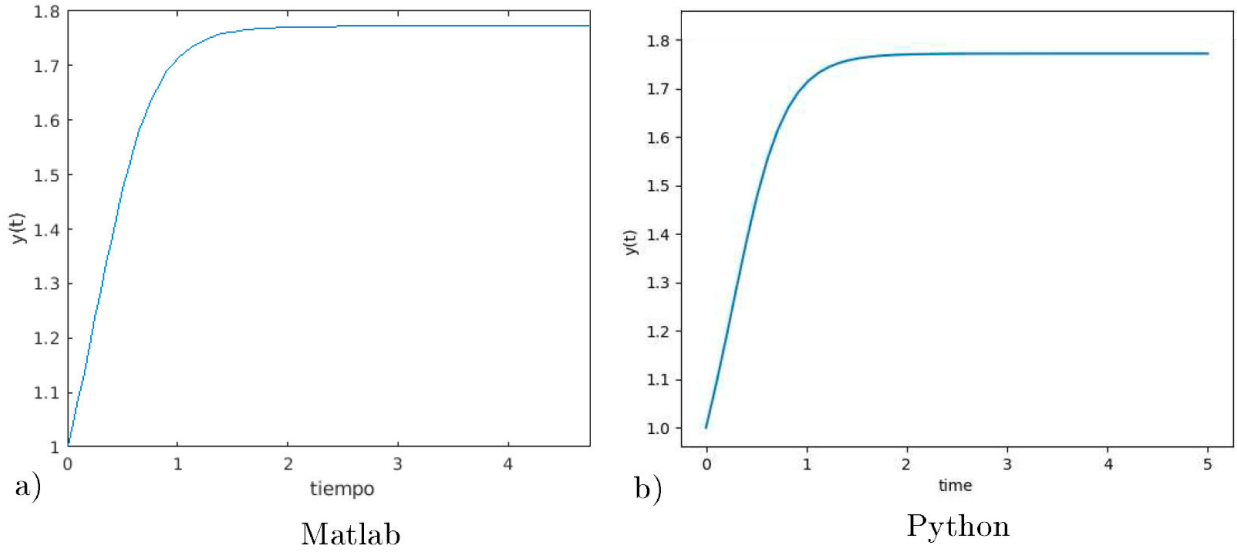

FIGURE 14. Gráficas de la soluciones numéricas de un problema de valor inicial para una ecuación diferencial no lineal obtenidas con Matlab y Python.

Este ejemplo muestra que aun cuando dsolve no pueda proporcionarnos una solución analítica, todavía en ambos programas podemos hallar una solución numérica. Cuando resolvemos de forma numérica debemos tener presente que la exactitud de nuestra aproximación puede verse afectada por las distintas fuentes de errores locales, globales y la estabilidad de la solución. En este ejemplo podemos cuestionar a los alumnos acerca del comportamiento de la solución cuando t tiende a infinito así como sí este comportamiento depende de la condición inicial elegida.

4. Conclusión

A lo largo de este trabajo hemos mostrado el uso de los comandos básicos dsolve (para hallar soluciones analíticas), ode45 y odeint (para hallar soluciones numéricas) para resolver ecuaciones diferenciales ordinarias. Nuestro objetivo principal es proporcionar al lector una guía práctica que le permita iniciar el estudio de las ecuaciones diferenciales ordinarias apoyándose en los paquetes simbólicos Symbolic Toolbox Matlab y SymPy Python. Quedan claramente expuestas las conveniencias e incomodidades de usar uno u otro paquete en función de la brevedad o extensión de los comandos utilizados para obtener o representar las soluciones, y no en función del tiempo de cómputo requerido para obtenerlas. La calidad de los gráficos y la sintáxis de la programación son muy similares en ambos paquetes. A diferencia de Matlab, en Python es necesario importar las librerías que se desean usar (por ejemplo numpy, simpy, matplotlib, etc) esto podria ser considerado como una incomodidad o desventaja por algunos usuarios. Desde el enfoque didáctico, es una ventaja de Sympy (Python) sobre Matlab el contar con las funciones `classify_ode` y `checkodesol`. Sin embargo consideramos que la mayor ventaja de Python es el hecho de ser de uso libre, ya que si bien un estudiante de licenciatura o postgrado puede aprender a usar Matlab durante su vida estudiantil bajo el permiso de la licencia institucional, a su egreso como profesionista se enfrenta con el hecho de que la gran mayoría de las empresas no compran software comercial y prefieren uso de software libre. Se han resaltado algunas de las ventajas del uso de estos paquetes de cómputo simbólico para calcular y simplificar expresiones, así como, sus opciones gráficas para fortalecer la enseñanza de conceptos importantes. El conjunto de ejemplos, cuestionamientos, así como comentarios incluidos son limitados. Sin embargo, pueden tomarse como punto de partida para que el lector interesado pueda escribir en Matlab y Python sus propios ejemplos (de acuerdo a su área de interés), así como proyectos didácticos para utilizar en clase y en tareas asignadas.

La resolución de problemas reales aplicados a la ingeniería con ecuaciones diferenciales presenta un reto tanto para los estudiantes, como para los profesores en el proceso de enseñanza-aprendizaje-evaluación.

El proceso de modelización matemática con los programas Matlab o Python fomenta el aprendizaje y comprensión de los conceptos matemáticos, permite seguir una metodología, motivar a los jóvenes, y representa una herramienta para prepararlos de manera adecuada en el ámbito profesional. Además, los proyectos de trabajo en equipo cumplen con las competencias disciplinares y las competencias genéricas al elaborar modelos, resolver problemas, generar gráficas, discutir, analizar, interpretar los resultados obtenidos, y argumentar sus conclusiones con el lenguaje matemático. En la misma línea, los estudiantes pueden responder preguntas del tipo ¿qué pasaría si . . . ? con una mínima programación.

Es de gran acierto enriquecer el contenido temático de los cursos de ecuaciones diferenciales ordinarias incluyendo el uso de paquetes computacionales para resolver ecuaciones diferenciales ordinarias [23–25]. Todo el esfuerzo que debe realizar el instructor para elegir ejemplos, ejercicios y proyectos de clase adecuados a los conceptos que se desee enseñar, se verá recompensado con la comprensión de conceptos importantes por parte de los alumnos [26–29].

Como trabajos a futuro podemos mencionar el uso de estos paquetes para crear interfaces gráficas para usuario (donde puede variar los parámetros de las ecuaciones y visualizar los cambios en las soluciones obtenidas) así como sistemas de autoevaluación de conocimientos en línea mediante preguntas de opción múltiple. El objetivo de la autoevaluación es proporcionar al estudiante una evaluación y entrenamiento para comprobar si un determinado tema del o los cursos que atiende ya lo conoce y lo comprende. De esta manera el alumno cuenta con un diagnóstico de sus fortalezas y debilidades en determinados saberes [30]. Más aún, recientemente, investigadores han demostramos que una red neuronal preentrenada en texto y afinada en código, resuelve los problemas del curso de matemáticas, explica las soluciones y genera preguntas en un nivel humano (aquí Python es usado para generar gráficas y resolver ecuaciones). Esto último abre la posibilidad de desarrollar asistentes automatizados que guien de manera individual a los alumnos en los procesos de aprendizaje [31].

Podemos concluir que Python es una excelente opción, ya que además de ser un lenguaje de programación interpretado, dinámico, multiplataforma y de libre acceso, sus librerías Sympy, Scipy y Numpy le proporcionan capacidades que le permiten estar a la par con paquetes comerciales de cómputo simbólico y numérico como Maple, Mathematica y Matlab. Esto explica el reciente surgimiento de publicaciones que abordan las ecuaciones diferenciales [32] y los métodos numéricos usando Python [33–35].

1. L. Kwan and M.A. Serdina, *Impact of Using Graphing Calculator in Problem Solving*, International Electronic Journal of Mathematics Education, (2018), 13(3).
2. F. Zeynivandnezhad, Z. Ismail Z. and Y.M. Yusof, *Teaching mathematical structures in differential equations using a computer algebra system to engineering students*, IEEE 7th International Conference on Engineering Education (ICEED), Kanazawa, Japan, 2015, pp. 10-15, https://doi.org/10.1109/ICEED.2015.7451483.
3. R. Gilbert, G. Hsiao, R. Ronkese, *Maple Projects of Differential equations*, 2nd Edition, Chapman and Hall /CRC, 2021.
4. M. Abell, J. Braselton,*Differential equations with Mathematica*, 4th edition, 2016.
5. M. Abrofarakh, T. Bux, *Solving Ordinary Differential Equations with Matlab: Learning matlab in 2 hours*, kindle edition, 2020.
6. G. Lindfield, J. Penny, *Numerical Methods using matlab*, (4th edition kindle, 2019).
7. E. Lozada, C. Guerrero, A. Coronel, R. Medina, *Classroom Methodologies for teaching and learning differential equations: A systematic literature review and biblioimetric analysis*, Mathematics **9** (2021) 745.
8. B.K. Amangeldievna, *Formation of sustainable motivation to study the subject Differential Equations*, Eurasian journal of learning and academic teaching, **l7** (2022).
9. A. C. Brandi and R.E. García, *Motivating engineering students to math classes: practical experience teaching ordinary differential equations*, IEEE Frontiers in Education Conference 2017.
10. C. Coelho, R. Marreiros and A. C. Conceicao, *Interactive learning of modeling with ordinary differential equations*, SYMCOMP, Portugal 2015.
11. O. N. Kwon, *Conceptualizing the realistic mathematics education approach in the teaching and learning of ordinary differential equations*, International conference on the teaching of mathematics (at the undergrade level, 2002).
12. C. Guerrero, M. Camacho, and H.R. Mejía, *Dificultades de los estudiantes en la interpretación de las soluciones de ecuaciones diferenciales ordinarias que modelan un problema*, *Enseñanza de la ciencia*, **28** (2010) 341-352.
13. O. N. Kwon, *Differential Equations Teaching and Learning*, Encyclopedia of Mathematics Education pp 220-223, Springer, 2020.
14. S. Habre, *Improving understanding in ordinary differential equations through writing in a dynamical environment*, Teaching Mathematics and its Applications: An International Journal of the IMA, **31** (2012) 153.
15. A. P. C. Lopes, and F. da Silva Reis, *Contributions of mathematical modelling for learning differential equations in the remote teaching context* Acta Scientiae, **24** (2022) 184.
16. B.H. West, *Teaching Differential Equations without Computer Graphics Solutions is a Crime*, CODEE Journal **11** (2018).
17. G. Ortigoza, *Resolviendo ecuaciones diferenciales ordinarias con Maple y Mathematica*, Rev. Mex. Fis. E., **53** (2007) 155.
18. G. Ortigoza, *Ecuaciones diferenciales Ordinarias con Maxima*, Rev. Edu. Mat., **21** (2009) 143.
19. website Matlab help center symbolic toolbox solving odes, https://www.mathworks.com/help/symbolic/solve-a-single-differential-equation.html.
20. Sympy 1.8 documentation, Sympy Modules Reference, ODE, disponible en https://docs.sympy.org/latest/modules/solvers/ode.html|.
21. Matlab Live Editor, disponible en https://la.mathworks.com/products/matlab/live-editor.html.
22. The scientific Python development environment, https://www.spyder-ide.org|
23. R. L. Lipsman, J. E. Osborn, and J. M. Rosenberg, *The SCHOL Project at the University of Maryland: Using Mathematical Software in the Teaching of Sophomore Differential Equations*, J. Num. Analysis, Industrial and Applied Mathematics, **3** (2008) 81.
24. S. M. Maat, and E. Zakaria, *Use of computer algebra systems in teaching and learning of ordinary differential equations among engineering technology students*, In book: Outcome-Based Science, Technology, Engineering, and Mathematics Education, 2012.
25. N. Lohgheswary, Z.M. Nopiah, E. Zakaria, A.A. Aziz and S. Salmaliza, *Incorporating Computer Algebra System in Differential Equations Syllabus*, Journal of Engineering and Applied Sciences, **14** (2019) 7475-7480.
26. S. M. Maat, E. Zakaria, *Exploring students' understanding of ordinary differential equations using computer algebraic system (Cas)*, the Turkish journal of educational technology, **10** (2011) I3.
27. F. Zeynivandnezhad, R. Bates, *Explicating mathematical thinking in differential equations using a computer algebra system*, International journal of mathematical education in science and technology, 2017.
28. N. Eyrikh, R. Bazhenov, T. Gorbunova, N. Markova and A. Zhunusakunova, *The Advantages of Using Computer Algebra System Maple in Learning Differential Equation*, V International Conference on Information Technologies in Engineering Education (Inforino), (2020) 1-5, https://doi.org/10.1109/Inforino48376.2020.9111726..
29. B.I. Zaleha, F. Zeynivandnezhad, B.M.Y. Yudariah, S. Bambang,*Integrating a computer algebra system as the pedagogical tool for enhancing mathematical thinking in learning differential equations*, Proceedings of the IETEC'13 Conference, Ho Chi Minh City, Vietnam, 2013.
30. B. F. Azevedo, A.I. Pereira, F. P. Fernandes, M. F. Pacheco, *Mathematics learning and assessment using MathE platform: A case study*, Educational and Information Technologies **27** (2022) 1747-1769.
31. I. Drori *et al*., *A neural network solves, explains, and generates university math problems by program synthesis and few-shot learning at human level*, Proc Natl Acad Sci U S A. **119** (2022) e2123433119.
32. M. Kanda, *Ordinary Differential equations and physical phenomena a short introduction with Python*, Asakura Publishing, 2020.

33. S. Chapra and D. Clough, *Applied Numerical Methods with Python for engineers and scientists*, 1st Ed, McGraw Hill, 2021.

34. R. Johansson, *Numerical Python: Scientific Computing and Data Science Applications with Numpy, SciPy and Matplotlib*, second edition Apress, 2019.

35. S. Linge, H. P. Langtangen, *Programming for Computations Python: A Gentle Introduction to Numerical Simulations with Python 3.6*, Second Edition, springer Open, 2018.