Revista Difusiones, ISSN 2314-1662, Num. 20, 2(1) enero-julio 2021, pp.88-104
Fecha de recepción: 18-04-2021. Fecha de aceptación: 09-06-2021

Análisis de entornos matemáticos numéricos para simulación

Analysis of numerical mathematical environments for simulation

Enrique Eduardo Tarifa[1]
eetarifa@fi.unju.edu.ar
Universidad Nacional de Jujuy, Facultad de Ingeniería, San Salvador de Jujuy, Jujuy, Argentina
Consejo Nacional de Investigaciones Científicas y Técnicas de Argentina (CONICET)
Código Orcid: https://orcid.org/0000-0003-3220-1440

Sergio Luis Martínez[2]
smartinez@fi.unju.edu.ar
Universidad Nacional de Jujuy, Facultad de Ingeniería, San Salvador de Jujuy, Jujuy, Argentina
Universidad Católica de Santiago del Estero, Departamento Académico San Salvador,
San Salvador de Jujuy, Jujuy, Argentina

Samuel Franco Domínguez[3]
sfrancodominguez@fi.unju.edu.ar
Universidad Nacional de Jujuy, Facultad de Ingeniería, San Salvador de Jujuy, Jujuy, Argentina

Álvaro Fabián Núñez[4]
alfanunez@fi.unju.edu.ar
Universidad Nacional de Jujuy, Facultad de Ingeniería, San Salvador de Jujuy, Jujuy, Argentina

[1] Dr. en Ingeniería Química, Ing. Químico, investigador de CONICET, docente-investigador de la UNJu.
[2] Mg. en Ingeniería Electrónica, Ing. Electrónico, docente-investigador de la UNJu, docente de DASS-UCSE.
[3] Ing. Químico, docente-investigador de la UNJu.
[4] Ing. Químico, docente-investigador de la UNJu.

Resumen

En investigación científica y, en particular, en simulación de procesos, MATLAB es el software estándar cuando se requiere un entorno matemático numérico. Sin embargo, el elevado precio de su licencia, el código propietario y la lentitud de cálculo son sus principales debilidades. En este trabajo, se analizan entornos matemáticos numéricos de código abierto y que no requieren el pago de licencias, con el fin de determinar si pueden sustituir a MATLAB en el área de simulación de procesos. Para ello, se recurre a evaluaciones realizadas en trabajos previos y se realiza una evaluación propia usando el modelo de espacio de estados de un caso de estudio. Del análisis realizado, se recomienda a GNU Octave como el mejor reemplazo de MATLAB por su alta compatibilidad y rendimiento. No obstante, también se recomienda el entorno Juno del lenguaje Julia, a pesar de no ser compatible con MATLAB, por presentar excelentes características y alcanzar velocidades comparables a C++.

Palabras clave

simulación, entorno numérico, GNU Octave, Julia

Abstract

In scientific research and, in particular, in process simulation, MATLAB is the standard software when a numerical mathematical environment is required. However, the high price of its license, the proprietary code and the slow calculation are its main weaknesses. In this work, open source numerical mathematical environments that do not require the payment of licenses are analyzed in order to determine if they can replace MATLAB in the area of process simulation. To do this, evaluations made in previous works are used and an own evaluation is carried out using the state space model of a case study. From the analysis performed, GNU Octave is recommended as the best MATLAB replacement for its high compatibility and performance. Nevertheless, the Juno environment of the Julia language is also recommended, despite not being compatible with MATLAB, for presenting excellent features and reaching speeds comparable to C ++.

Key Words

simulation, numerical environment, GNU Octave, Julia

Introducción

En investigación científica, el entorno MATLAB es la aplicación más utilizada cuando se requiere realizar cálculos numéricos. En particular, en simulación de procesos, dicho software se encuentra entre los más comúnmente usados. Sin embargo, el alto costo de la licencia, la relativa baja velocidad de cálculo y el carácter propietario del código son sus principales desventajas. Por ese motivo, es de interés analizar software específico, sin costo de licencia y de código abierto, que pueda sustituir a MATLAB en el campo de la simulación de procesos. Además, es deseable que el nuevo software sea compatible con MATLAB para poder aprovechar los códigos que ya se desarrollaron usando ese entorno. Varios fueron los programas que se desarrollaron tratando de cumplir con los requerimientos planteados, pero únicamente son dos los que actualmente continúan en vigencia: GNU Octave y Scilab (Almeida et al., 2012; Wouwer et al., 2014).

Recientemente, un nuevo tipo de entornos surgió para sustituir a MATLAB. Estos entornos están basados en verdaderos lenguajes de programación. En particular, dos entornos de este tipo son especialmente prometedores: Spyder para el lenguaje Python y Juno para el lenguaje Julia. Estos entornos cumplen con los requerimientos planteados, excepto con el referido a la compatibilidad con MATLAB. En cambio, ofrecen nuevas características que son también deseables: flexibilidad y velocidad.

Estos nuevos entornos fueron diseñados para resolver “el problema de los dos lenguajes” y “el problema de las dos culturas”. El primer problema es el que enfrentan los programadores en todos los proyectos: para el desarrollo se requiere un lenguaje de alto nivel, mientras que para la producción se necesita un lenguaje de bajo nivel. Los lenguajes de alto nivel permiten escribir códigos entendibles y ofrecen amplias facilidades para la depuración de los mismos. En cambio, los lenguajes de bajo nivel son menos flexibles, pero ofrecen velocidad y capacidad para manejar grandes volúmenes de datos (Shamshoian, 2018). Este problema obliga a los programadores a trabajar con dos lenguajes, uno para el desarrollo y otro para la producción, invirtiendo tiempo extra en la traducción. Cuando ello no es posible, eligen un único lenguaje, aceptando sus debilidades en el desarrollo o en la producción.

Los lenguajes de programación empleados para el desarrollo son generalmente interpretados; esto es, que el código se transforma en lenguaje de máquina a medida que se ejecuta. Esta característica brinda a esos lenguajes gran flexibilidad; pero les quita eficiencia, volviéndolos lentos e incapaces de manejar grandes volúmenes de datos. A pesar de estos inconvenientes, últimamente se han popularizado varios de estos lenguajes, siendo los más difundidos: Visual Basic, Java y Python. En cuanto a los lenguajes de programación empleados para la producción se destaca C++, que es un lenguaje compilable; esto es, primero todo el código se traduce al lenguaje máquina y recién entonces se ejecuta. Esto le quita flexibilidad; pero lo vuelve eficiente, lo que le permite

manejar grandes volúmenes de datos a gran velocidad. A esta categoría pertenece Fortran, el preferido en el área de la Física (Phillips, 2014).

Los nuevos entornos tratan de resolver este problema usando lenguajes interpretados para brindar la flexibilidad requerida. Sin embargo, para cumplir con el requisito de la velocidad, ofrecen también la posibilidad de compilar el código, aunque con distintos grados de dificultad y de rendimiento.

A "el problema de los dos lenguajes", se agrega "el problema de las dos culturas": la cultura de los científicos y la cultura de los programadores. Los primeros ven al software como una herramienta para obtener los resultados que les interesan. Ellos son desarrolladores y usuarios de sus propios códigos. En cambio, el centro de atención de los programadores es el software en sí mismo, el cual está pensado para satisfacer las necesidades de un determinado usuario (Ballanco, 2018).

Los nuevos entornos tratan de adaptar, para los científicos, lenguajes que fueron diseñados para programadores. Con ese fin, imitan lo mejor posible la interfaz y la forma de trabajo de MATLAB. No obstante, debido a que no emplean el lenguaje de MATLAB, no son compatibles con este software.

Por lo anterior, el entorno numérico a ser usado en simulación de procesos debe tener características específicas. Primero, debe se interactivo; esto es, debe permitir la ejecución de una instrucción sin necesidad de correr un programa completo. La velocidad de ejecución del código debe ser lo más elevada posible. El código debe ser conciso, intuitivo y fácil de leer y entender. El código se debe poder compartir, esto implica que debe ser portable a otras plataformas (Datseris, 2018).

En este trabajo, se analizan los entornos numéricos disponibles para simulación de procesos con el fin de seleccionar el más adecuado. La selección será de acuerdo a todas las características deseables citadas previamente. Por ese motivo, el análisis se centra en las alternativas no comerciales compatibles con MATLAB: GNU Octave y Scilab. También, se incluye en el análisis al entorno Spyder para el lenguaje Python y al entorno Juno para el lenguaje Julia. Para el análisis se recurre a evaluaciones realizadas en trabajos previos y se realiza una evaluación propia usando el modelo de espacio de estados de un caso de estudio.

Sistema en estudio

Para evaluar el desempeño de los entornos numéricos seleccionados en aplicaciones de simulación de procesos, se simulará el reactor CSTR (Continuous Stirred Tank Reactor, reactor tanque agitado continuo) que se presenta en la Figura 1. Este caso fue propuesto por Rawlings et al. (2009). En este equipo, tiene lugar una reacción de esterificación en fase líquida orgánica. Los reactivos A —el ácido orgánico— y B —la base orgánica— ingresan por corrientes separadas, en las que están disueltos por un solvente orgánico.

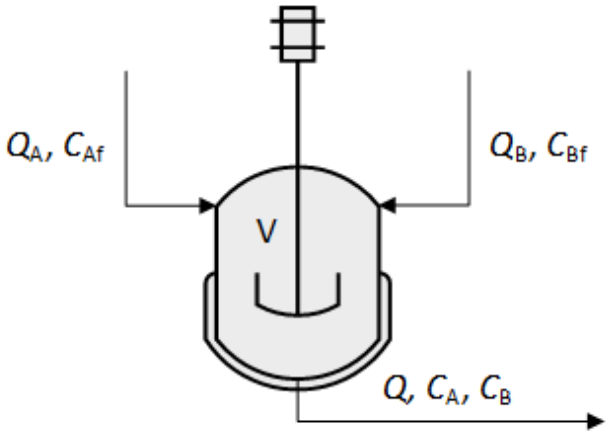

Figura 1. Reactor CSTR de esterificación (elaboración propia).

Los reactivos reaccionan de la siguiente manera:

$$A+B \rightarrow ester+H_2O \quad (1)$$

La velocidad de reacción *r* está dada por:

$$r = kC_A C_B \quad (2)$$

donde *k* es la constante cinética y C_A y C_B son las concentraciones molares en el reactor del ácido y de la base, respectivamente.

Se supone que la densidad y el volumen líquido en el reactor son constantes; entonces, se tiene que el caudal de salida *Q* es igual a la suma de los caudales de entrada Q_A y Q_B:

$$Q = Q_A + Q_B \quad (3)$$

Bajo estas condiciones, los balances de componentes son los siguientes:

$$\frac{dC_A}{dt} = \frac{1}{\tau}\left(\frac{Q_A}{Q}C_{Af} - C_A\right) - kC_A C_B \quad (4)$$

$$\frac{dC_B}{dt} = \frac{1}{\tau}\left(\frac{Q_B}{Q}C_{Bf} - C_B\right) - kC_A C_B \quad (5)$$

donde C_{Af} es la concentración en la alimentación del ácido orgánico, C_{Bf} es la concentración en la alimentación de la base orgánica y es el tiempo de residencia del reactor.

Las concentraciones de A y B son los dos primeros elementos del vector de variables de estado del modelo:

$$x_1 = C_A, x_2 = C_B \quad (6)$$

El vector de variables de estado se completa con las variables auxiliares necesarias para

calcular las conversiones de A y B. En un estado dinámico, la conversión de un reactivo varía con el tiempo. Para un reactivo dado, en un momento dado, la conversión será igual al cociente entre la cantidad de moles consumidos por la reacción y la cantidad total alimentada al reactor. Entonces, las variables auxiliares que se necesitan para calcular las conversiones son la cantidad de A que reaccionó, la cantidad de B que reaccionó, la cantidad de A que ingresó en el reactor y la cantidad de B que ingresó en el reactor, todo por unidad de volumen. Por estequiometría, las dos primeras cantidades son iguales; por ello, se asignan al tercer elemento del vector de estado:

$$\frac{dx_3}{dt} = k\,C_A\,C_B \tag{7}$$

En cambio, las cantidades de A y de B que ingresan al reactor por unidad de volumen, se asignan al cuarto y quinto elemento, respectivamente:

$$\frac{dx_4}{dt} = \frac{Q_A}{Q\tau} C_{Af} \tag{8}$$

$$\frac{dx_5}{dt} = \frac{Q_B}{Q\tau} C_{Bf} \tag{9}$$

Luego, las conversiones X_A y X_B son:

$$X_A = \frac{x_3}{x_4} \tag{10}$$

$$X_B = \frac{x_3}{x_5} \tag{11}$$

El modelo formado por las ecs. (4)-(11) fue complementado con las siguientes condiciones iniciales:

$$x_0 = \begin{pmatrix} C_{A0} & C_{B0} & 0 & C_{A0} & C_{B0} \end{pmatrix}^T \tag{12}$$

El modelo descripto es un modelo de espacio de estados (Ogata, 2010). En la implementación de este modelo en los entornos evaluados, se empleó el siguiente parámetro para fijar la relación de caudales de los reactivos:

$$\alpha = \frac{Q_A}{Q_B} \tag{13}$$

Dicha implementación se realizó con el fin de evaluar, para los entornos seleccionados, la riqueza expresiva del lenguaje, la velocidad de cálculo, las facilidades gráficas y la compatibilidad con MATLAB.

Características deseables

Como se adelantó, el entorno a elegir debe reunir una serie de características deseables para la simulación de procesos. A continuación, se enuncian las características que son deseables en base a la propia experiencia y a la de otros autores (Dateris, 2018; Almeida et al., 2012; Wouwer et al., 2014; Shamshoian, 2018):

Compatibilidad con MATLAB

Debido a la inserción que tiene MATLAB en el campo de la simulación de procesos, existe una variedad de códigos disponibles, así como también una amplia documentación acerca del lenguaje que emplea. La compatibilidad con MATLAB permitiría reusar el código existente propio o de otros autores. También, facilitaría la transición al nuevo entorno porque seguiría siendo útil la experiencia que ya se tiene con el lenguaje de MATLAB. Por último, permitiría el trabajo colaborativo con otros grupos de investigación que no deseen cambiar de entorno.

Consumo de recursos

Es deseable que el entorno se instale fácilmente y consuma poco espacio en el disco. También, se requiere que no consuma demasiada memoria y potencia del procesador cuando se ejecute.

Interactividad

La interactividad es la propiedad que tiene el lenguaje de poder ejecutar comandos o porciones del código. Esto agiliza en gran medida el desarrollo del código al permitir realizar pruebas mientras se escribe en lugar de tener que esperar hasta finalizarlo. Para permitir la interactividad, el lenguaje debe ser interpretado. MATLAB, al igual que todos los lenguajes analizados en este trabajo son interactivos y, por lo tanto, son interpretados. La Figura 2 presenta la ventana de comandos de MATLAB, donde se realiza el producto de una matriz y un vector.

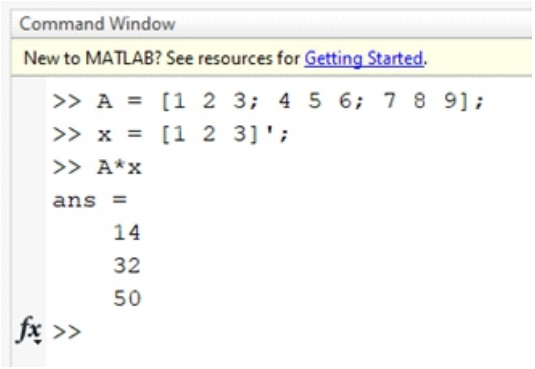

Figura 2. Ventana de comando de MATLAB (elaboración propia)

Tipado dinámico

Un lenguaje de programación es dinámicamente tipado si una variable puede tomar valores

de distintos tipos —enteros, reales, cadenas— durante la ejecución del código. MATLAB tiene tipado dinámico al igual que todos los entornos analizados.

Con el tipado dinámico, no es necesario declarar el tipo de cada variable a emplear. Esto agiliza el desarrollo de códigos hasta cierto punto; pero puede ser contraproducente en códigos extensos: al no ser necesaria la declaración de variables, pueden introducirse involuntariamente nuevas variables por error de tipeo. La Figura 3 muestra un ejemplo de tipado dinámico en MATLAB; la variable a es primero un vector y luego es un escalar.

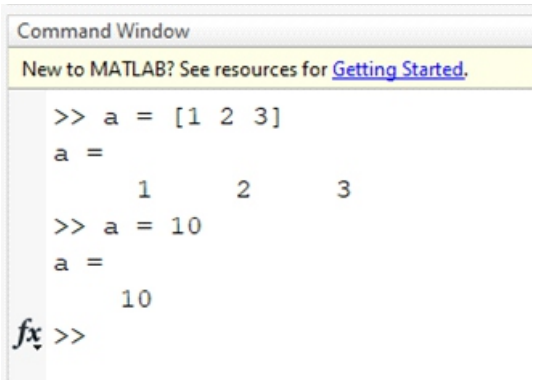

Figura 3. Ejemplo de tipado dinámico en MATLAB (elaboración propia).

Velocidad

La velocidad de ejecución del código debe ser lo más alta posible. Los lenguajes compilables son los más veloces. Generalmente, se acepta que la implementación en C++ tiene la máxima velocidad, y por eso es la referencia para evaluar la velocidad de otros lenguajes.

Expresividad

La sintaxis del lenguaje de programación debe ser lo suficientemente expresiva como para permitir la escritura de códigos concisos, intuitivos y entendibles. Esta característica del lenguaje facilita la escritura del código y su posterior mantenimiento.

Portabilidad

En la investigación científica es fundamental la cooperación, para lo cual el código debe poder compartirse fácilmente. Eso implica que debe poder correr en los sistemas operativos más difundidos —Windows, GNU Linux y Mac OS— sin modificaciones. Todos los entornos analizados en este trabajo cumplen con este requisito.

Costo

El entorno debe ser no comercial para evitar el impacto sobre el financiamiento de los proyectos de investigación del mantenimiento de licencias de software.

Accesibilidad del código

Debe ser de código abierto para conocer exactamente lo que hace cada sentencia y, si es necesario, personalizar sus funcionalidades. Todas las alternativas a MATLAB que se

analizan en el presente trabajo cumplen con este requisito.

Disponibilidad de bibliotecas

El lenguaje debe tener disponibles las bibliotecas necesarias para simulación de procesos: resolución de sistemas de ecuaciones algebraicas lineales y no lineales, resolución de sistema de ecuaciones diferenciales ordinarias y parciales, rutinas gráficas, manejo de archivos, vinculación con planillas de cálculo. De no tener bibliotecas propias, debe permitir la vinculación con bibliotecas existentes para otros lenguajes.

Resultados

MATLAB

La primera versión de MATLAB es de 1984. El entorno contiene las siguientes ventanas en su interfaz: explorador de archivos, espacio de trabajo (muestras las variables en memoria), historial de comandos, consola (donde se introducen las instrucciones a ejecutar en forma inmediata), editor (donde se escribe el código). El editor dispone de varias facilidades para la depuración del código. MATLAB posee una amplia biblioteca de funciones, organizadas en grupos orientados a diferentes disciplinas, denominados toolboxes. Además, cuenta con Simulink, un simulador gráfico de diagramas de bloques. Los diagramas de bloques ofrecen una forma alternativa para implementar modelos de espacio de estados (Ogata, 2010). La Tabla 1 muestra el código de MATLAB para el caso de estudio.

```
function dummy()
% Listado en MATLAB
global tau k caf cbf qaq qbq
tau = 10; k = 0.1; caf = 8; cbf = 4;
ca0 = caf; cb0 = cbf;
alpha= 1;
qaq = alpha/(1+alpha);
qbq = 1/(1+alpha);
tfin = 5*tau;
nts = 100;
tpts = linspace(0,tfin,nts)';
x0 = [ca0;cb0;0;ca0;cb0];

[tpts,x] = ode15s(@cstr,tpts,x0);

xa = x(:,3)./x(:,4);
xb = x(:,3)./x(:,5);
figure(1);
plot(tpts,[x(:,1),x(:,2)])
figure(2);
plot(tpts,[xa, xb])

function rhs = cstr(t,x)
  global tau k caf cbf qaq qbq
  ca = x(1);
  cb = x(2);
  rhs = [(qaq*caf-ca)/tau-k*ca*cb;
         (qbq*cbf-cb)/tau-k*ca*cb;
         k*ca*cb;
         qaq/tau*caf;
         qbq/tau*cbf];
```

Tabla 1. Listado en MATLAB (elaboración propia).

GNU Octave

La primera versión de GNU Octave es de 1993. La interfaz del entorno tiene las mismas ventanas que la interfaz de MATLAB. El lenguaje que emplea es el más compatible con MATLAB. Sin embargo, esta compatibilidad deja de existir cuando se requiere algún toolbox o rutina propia de MATLAB. La Tabla 2 contiene el código implementado en GNU Octave para el caso de estudio. Las diferencias que se observan con respecto al código de MATLAB (Tabla 1) se deben a que se tuvo que sustituir la función ode15s por la función lsode.

```
% Listado de GNU Octave
global tau k caf cbf qaq qbq
tau = 10; k = 0.1; caf = 8; cbf = 4;
ca0 = caf; cb0 = cbf;
alpha= 1;
qaq = alpha/(1+alpha);
qbq = 1/(1+alpha);

function rhs = cstr(x,t)
  global tau k caf cbf qaq qbq
  ca = x(1);
  cb = x(2);
  rhs = [(qaq*caf-ca)/tau-k*ca*cb;
         (qbq*cbf-cb)/tau-k*ca*cb;
         k*ca*cb;
         qaq/tau*caf;
         qbq/tau*cbf];
endfunction

tfin= 5*tau;
nts = 100;
tpts= linspace(0,tfin,nts)';
x0 = [ca0;cb0;0;ca0;cb0];

x = lsode('cstr',x0,tpts);

xa = x(:,3)./x(:,4);
xb = x(:,3)./x(:,5);
figure(1);
plot(tpts,[x(:,1),x(:,2)])
figure(2);
plot(tpts,[xa,xb])
```

Tabla 2. Listado en GNU Octave (elaboración propia).

Scilab

La primera versión de Scilab es de 1994. La interfaz es similar a la de MATLAB. El lenguaje es compatible con MATLAB, pero en menor medida que GNU Octave. Cuenta con un entorno para simulación de diagramas de bloques, el Xcos. Este entorno es el equivalente a Simulink de MATLAB, pero no es compatible con él. La contiene el código correspondiente al caso de estudio. Las diferencias con el código de MATLAB (Tabla 1) se deben al reemplazo de la función ode15s por la función ode.

```
// Listado en Scilab
tau = 10; k = 0.1; caf = 8; cbf = 4;
ca0 = caf; cb0 = cbf;
alpha= 1;
qaq = alpha/(1+alpha);
qbq = 1/(1+alpha);

function rhs=cstr(t,x)
   ca = x(1);
   cb = x(2);
   rhs = [(qaq*caf-ca)/tau-k*ca*cb;
          (qbq*cbf-cb)/tau-k*ca*cb;
          k*ca*cb;
          qaq/tau*caf;
          qbq/tau*cbf];
endfunction

tfin= 5*tau;
nts = 100;
tpts= linspace(0,tfin,nts)';
x0 = [ca0;cb0;0;ca0;cb0];
t0 = 0;
x = ode(x0,t0,tpts,cstr)';
xa = x(:,3)./x(:,4);
xb = x(:,3)./x(:,5);
figure(1);
plot(tpts,[x(:,1),x(:,2)])
figure(2);
plot(tpts,[xa,xb])
```

Tabla 3. Listado en Scilab (elaboración propia).

Spyder

Spyder es el entono desarrollado para el lenguaje de propósito general Python. La interfaz es similar a la de MATLAB. El paquete científico SciPy para Python surgió en 2001. El lenguaje no es compatible con MATLAB. El código se puede compilar con el compilador Numba, pero requiere modificaciones para adaptarlo al compilador. Numba solo optimiza funciones matemáticas, no optimiza todo el código. La Tabla 4 muestra el código para el caso de estudio.

```
# Listado en Python
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt

tau = 10
k = 0.1
caf = 8
cbf = 4
ca0 = caf
cb0 = cbf
alpha = 1
qaq = alpha/(1+alpha)
qbq = 1/(1+alpha)
tfin = 5*tau
nts = 100

def model(x,t):
  ca = x[0]
  cb = x[1]
  rhs = [(qaq*caf-ca)/tau-k*ca*cb,
         (qbq*cbf-cb)/tau-k*ca*cb,
         k*ca*cb,
         qaq/tau*caf,
         qbq/tau*cbf]
  return rhs

x0 = [ca0,cb0,0,ca0,cb0]
t = np.linspace(0,tfin,nts)

x = odeint(model,x0,t)

xa = x[:,2]/x[:,3]
xb = x[:,2]/x[:,4]

plt.plot(t,x[:,0])
plt.plot(t,x[:,1])
plt.show()

plt.plot(t,xa)
plt.plot(t,xb)
plt.show()
```

Tabla 4. Listado en Python (elaboración propia)

Juno

La interfaz del entorno Juno para el lenguaje Julia es similar al de MATLAB, pero los lenguajes que emplean no son compatibles. Julia es un lenguaje moderno, la primera versión es de 2012. Julia brinda soporte para Unicode, incluyendo UTF-8, lo que permite la utilización de símbolos, como las letras griegas y símbolos matemáticos, en el código. La sintaxis es clara y permite un manejo de vectores similar al de MATLAB. Cuenta con una amplia biblioteca, y puede emplear las desarrolladas para Python, R, C/Fortran, C++ y Java. Una característica distintiva de Julia es la compilación just in time. Gracias a ella, Julia puede ejecutar comandos como si fuera un lenguaje interpretado; pero los programas se ejecutan con una velocidad comparable a los compilados. La Tabla 5 contiene la implementación del caso de estudio usando Julia. En las primeras líneas, se importan los paquetes necesarios; pero esto debe realizarse una única vez.

```
# Listado en Julia
importPkg; Pkg.add("ParameterizedFunctions")
importPkg; Pkg.add("Plots")
importPkg; Pkg.add("OrdinaryDiffEq")

usingParameterizedFunctions, Plots,
OrdinaryDiffEq

# definición del sistema
reactor= @ode_def
begin
 dca = (qaq*caf-ca)/τ-k*ca*cb
 dcb = (qbq*cbf-cb)/τ-k*ca*cb
 dx3 = k*ca*cb
 dx4 = qaq/τ*caf
 dx5 = qbq/τ*cbf
end τ k caf cbf ca0 cb0 qaq qbq

# condiciones iniciales
x0 = [8.0,4.0,0.0,8.0,4.0]
# rango de tiempo
tspan= (0.0,50.0)
# parámetros
p = [10.0,0.1,8.1,4.0,8.0,4.0,1.0,0.5,0.5]

prob = ODEProblem(reactor,x0,tspan,p)
sol = solve(prob)
xa = sol[3,:]./sol[4,:]
xb = sol[3,:]./sol[5,:]

plot(sol,vars=(1))
plot!(sol,vars=(2))
plot(sol.t,xa)
plot!(sol.t,xb)
```

Tabla 5. Listado en Julia (elaboración propia)

Discusión

GNU Octave y Scilab son los entornos más utilizados como alternativa directa de MATLAB por su compatibilidad. En las pruebas realizadas por Coman et al. (2012), MATLAB y Octave fueron capaces de resolver con la misma velocidad problemas del mismo tamaño, mientras que Scilab mostró limitaciones en ambos aspectos. Almeida et al. (2012) llegaron a conclusiones similares luego de realizar pruebas en Windows, GNU Linux y Mac OS. Lo mismo fue observado en el presente trabajo cuando se implementó el modelo del caso de estudio en cada entorno y en cada sistema operativo de interés. Además, si bien Scilab posee un conversor para importar archivos de MATLAB, la importación no funcionó en las pruebas realizadas para el caso de estudio.

La instalación de los entornos analizados presentó distintos grados de dificultad. Las instalaciones de los entornos Spyder y Juno fueron las más complejas debido a que requieren las instalaciones conjuntas tanto de las interfaces gráficas como de sus respectivos lenguajes. En ambos casos se requirieron la instalación de paquetes adicionales provocando un mayor uso de espacio de almacenamiento en disco. En cambio, GNU Octave y Scilab fueron de fácil instalación y consumieron menos espacio de disco.

En cuanto a la velocidad de cálculo, Kamiński (2017) resolvió el modelo de la Opción Asiática

usando las siguientes alternativas: Julia, Julia con la opción de vectores, Python interpretado y Python compilado con Numba (Tabla 6). El parámetro m es una medida del tamaño del problema. Del análisis surge que la implementación con Julia es la más veloz en todos los casos. Partiendo de ese trabajo, se analizó la sensibilidad al escalado. Para ello, se consideró que, entre el primer caso y el último, el tamaño del problema creció en un factor de 210/25 = 32. Para este factor de crecimiento, la última fila de la Tabla 6 muestra los correspondientes factores f de aumento de tiempo de procesamiento para las distintas implementaciones. De acuerdo a los factores f, Numba presenta la menor sensibilidad al escalado.

m	Julia	J. vector	Python	Numba
2^5	0.06	0.11	7.53	0.73
2^6	0.10	0.17	14.22	0.89
2^7	0.19	0.49	27.28	1.12
2^8	0.39	0.92	54.97	1.86
2^9	0.77	1.73	110.42	3.17
2^{10}	1.53	2.88	212.25	5.77
f= 32	25.50	26.18	28.19	7.90

Tabla 6. Tiempo de cálculo en segundos para el modelo de opción asiática (elaboración propia).

El modelo del reactor de esterificación planteado como caso de estudio pudo ser resuelto por todos los entornos analizados (Figura 4 y Figura 5). Durante el estudio, se observó que dichos entornos cumplen con casi todas las características deseables. Sin embargo, algunos destacan notablemente en algunos aspectos: mientras GNU Octave sobresale en la compatibilidad con MATLAB, el entorno Juno del lenguaje Julia lo hace en todas las otras características.

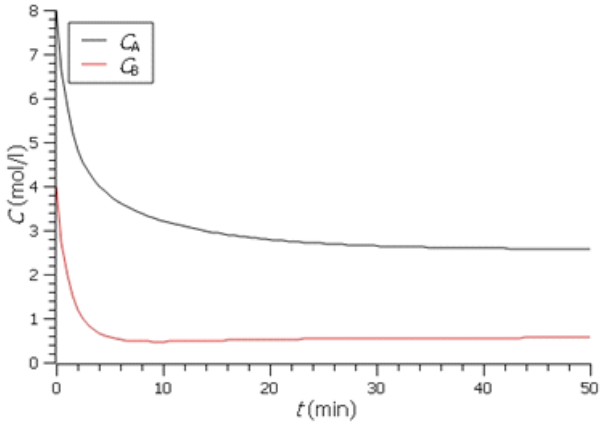

Figura 4. Evolución de las concentraciones (elaboración propia).

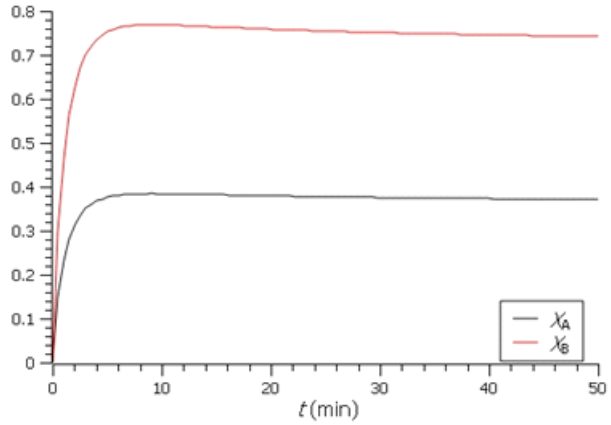

Figura 5. Evolución de las conversiones (elaboración propia).

Julia y Python tienen otros entornos —similares al de MATLAB— implementados en VS Code de Microsoft; lo cual puede ser importante en el futuro debido a que dicha empresa adquirió Atom en 2018, el editor donde funciona el entorno Juno. Además, ambos lenguajes funcionan en Jupyter, una interfaz moderna, basada en celdas, que permite documentar el código. Esta interfaz admite la ejecución de celdas individuales o grupos de celdas. A su vez, cada celda puede contener una única sentencia o un programa completo. De esta manera, la interpretación o la compilación del código es un proceso transparente para el usuario.

En la Tabla 7 se resumen los resultados de las evaluaciones realizadas. Estos resultados coinciden con los de trabajos previos al señalar a GNU Octave como el entorno más compatible con MATLAB. Sin embargo, los resultados alcanzados con Julia en el entorno Juno son ampliamente mejores si se deja de lado la compatibilidad con MATLAB. Esta ventaja se incrementa si se considera la disponibilidad de nuevos y potentes entornos como los son VS Code y Jupyter.

Característica s	MATLAB	GNU Octave	Scilab	Spyder	Juno
Compatibilidad con MATLAB	Perfecta	Alta	Media	Ninguna	Ninguna
Consumo de recursos	Alto	Medio	Medio	Bajo	Bajo
Interactividad	Alto	Alto	Alto	Alto	Alto
Tipeado dinámico	Sí	Sí	Sí	Sí	Sí
Velocidad	Baja	Baja	Baja	Media	Alta
Expresividad	Media	Media	Media	Alta	Alta
Portabilidad	Sí	Sí	Sí	Sí	Sí
Costo	Alto	Nulo	Nulo	Nulo	Nulo
Accesibilidad del código	No	Sí	Sí	Sí	Sí
Disponibilidad de bibliotecas	Alta	Baja	Baja	Alta	Alta

Tabla 7. Resumen de resultados (elaboración propia)

Conclusiones

En este trabajo, se analizaron entornos matemáticos numéricos no comerciales que pueden sustituir a MATLAB cuando se requiere realizar cálculos numéricos en la investigación científica en general y en el campo de la simulación de procesos en particular.
En el análisis realizado, se distinguieron dos grupos de entornos. En el primer grupo, están los entornos exclusivamente originados en el campo de la matemática: MATLAB, GNU Octave y Scilab. En el segundo grupo, están los entornos desarrollados para los lenguajes de programación de propósito general Python y Julia. Ambos tipos de entornos son adecuados para desarrollos rápidos de prototipos orientados a la investigación.
En base a investigaciones realizadas por otros autores y a las propias conclusiones obtenidas luego del análisis, se concluye que GNU Octave es el entorno más apropiado para reemplazar MATLAB si el propósito es trabajar con modelos de espacio de estados. En cambio, si se va a trabajar con diagramas de bloques, la única alternativa no comercial a Simulink de MATLAB es Xcos de Scilab, que es una aplicación que no fue analizada en el presente trabajo.
En cuanto a los entornos basados en Python y Julia, ninguno es compatible con MATLAB. Python es un lenguaje maduro y ampliamente difundido. Julia es un lenguaje que se sigue desarrollando; no obstante, ya presenta excelentes características como para ser considerado el más adecuado para simulación de procesos.

Referencias

Almeida, E. S., Medeiros, A. C., & Frery, A. C. How good are MatLab, Octave and Scilab for Computational Modelling? Computational and Applied Mathematics, 31(3), 523-538. https://doi.org/10.1590/S1807-03022012000300005, 2012.

Ballanco, J. JuliaCon 2016 | The Two Cultures of Programming [Archivo de video]. [The Julia Programming Language]. https://www.youtube.com/watch?v=C3iR_PknlFc, 2018.

Coman, E., Brewster, M. W., Popuri, S. K., Raim, A. M., & Gobbert, M. K. A Comparative Evaluation of Matlab, Octave, FreeMat, Scilab, R, and IDL on Tara (Technical Report HPCF–2012–15; p. 46). http://hpcf-files.umbc.edu/research/papers/ComanHPCF2012.pdf, 2012.

Datseris, G. JuliaCon 2018 | Why Julia is the most suitable language for science [Archivo de video]. [The Julia Programming Language]. https://www.youtube.com/watch?v=7y-ahkUsIrY&t=532s, 2018.

Kamiński, B. First steps with Julia for numerical computing [Archivo de video]. [PyData]. https://www.youtube.com/watch?v=gaJorAU644o&t=889s, 2017.

Ogata, K. Ingeniería de control moderna (5°). PEARSON EDUCACIÓN S.A., 2010.

Phillips, L. Scientific computing's future: Can any coding language top a 1950s behemoth? Ars Technica. https://arstechnica.com/science/2014/05/scientific-computings-future-can-any-coding-language-top-a-1950s-behemoth/, 2014.

Rawlings, J. B., Eaton, J. W., & Ekedt, J. G. A User's Guide for Translating Octave to Matlab. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.489.2734&rep=rep1&type=pdf, 2009.

Shamshoian, M. Julia: A Solution to the Two-Language Programming Problem. The Bottom Line. https://thebottomline.as.ucsb.edu/2018/10/julia-a-solution-to-the-two-language-programming-problem, 2018.

Wouwer, A. V., Saucez, P., & Vilas, C. Simulation of ODE/PDE Models with MATLAB, OCTAVE and SCILAB. Scientific and Engineering Applications. Springer, 2014.