Elementos básicos de programación en MATLAB® y su aplicación al graficar funciones tratadas en cálculo diferencial

Basic Programming Elements in MATLAB® and their Application in Differential Calculus Function Graphics

Recibido: 17 de febrero de 2017- Aceptado: 18 de mayo de 2017

Para citar este artículo: W. Castro y H. Moreno, «Elementos básicos de programación en MATLAB® y su aplicación al graficar funciones tratadas en cálculo diferencial», *Ingenium*, vol. 19, no. 37, pp. 38-61, ene. - jun. 2018.»

Wilson Castro Zapata*
Henry Moreno Cañadas**

Resumen

Este artículo aplica elementos de programación básicos en MATLAB® para abordar conceptos de funciones matemáticas que hacen parte del currículo de cálculo diferencial, mediante el manejo de variables y estructuras tales como ciclos y condicionales, scripts y uso de function. Muestra y explica los scripts de aplicación desarrollados en temas de funciones matemáticas como desplazamientos verticales y horizontales de la gráfica de una función, estiramiento y reflexión vertical y horizontal y la continuidad de una función. Se tratan también las funciones polinomiales, trigonométricas, definidas por partes y curvas paramétricas. Se ilustran algunos programas pequeños para el manejo de funciones, y termina con el desarrollo de una aplicación con interfaz gráfica para el

* Especialista en Actuaría, Universidad Antonio Nariño. Profesor, Universidad de San Buenaventura, sede Bogotá. E-mail: wcastro@usbbog.edu.co

** M. Sc. Ciencias de la Educación, Profesor, Universidad de San Buenaventura, sede Bogotá. E-mail: hmoreno@usbbog.edu.co

estudio del movimiento parabólico. MATLAB® es un software comercial y para su manejo se recomienda revisar los tipos de licencia existentes.

Palabras clave

MATLAB, programación en MATLAB, variables, funciones, curvas paramétricas, continuidad, funciones polinómicas, funciones trigonométricas, transformación de funciones, creación de interfaces en MATLAB, aplicaciones computacionales, gráficas, graficación en MATLAB.

Abstract

This article applies basic programming elements in MATLAB® to address concepts of mathematical functions according to the curriculum of differential calculus, by handling variables and structures such as cycles and conditionals, scripts, and use of *function*. It shows and explains the developed application scripts in themes of mathematical functions as vertical and horizontal shifts of the graph of a function, vertical and horizontal stretching and reflecting and the continuity of a function. Polynomial, trigonometric and piecewise defined functions are treated. Some short programs are illustrated for the handling of functions and it ends with the development of an application with graphic interface for the study of the parabolic movement. MATLAB® is a commercial software and it is recommended that you review the existing license types for its using.

Keywords

MATLAB®, programming in MATLAB, variables, functions, parametric curves, continuity, polynomial functions, trigonometric functions, transformation of functions, creating interfaces in MATLAB, computational applications, graphs, plotting in MATLAB.

1. Introducción

La Facultad de Ingeniería de la Universidad de San Buenaventura tiene al software MATLAB como recurso de apoyo y complemento de los planes educativos en las asignaturas de matemáticas, así como en otras específicas para cada programa de ingeniería. El aprendizaje y manejo de herramientas computacionales en donde se involucre la resolución de problemas, permite la construcción de esquemas conceptuales que favorecen la construcción y exploración de temáticas propias de la formación del ingeniero. Asimismo, es importante para el ingeniero aprender a programar con lenguajes y herramientas computacionales para la simulación. Gloria Contreras y Patricia Carreño confirman esta afirmación al realizar una revisión sobre simuladores y los beneficios de la simulación en la enseñanza [10], y autores como Héctor Vega han desarrollado algoritmos para soluciones matemáticas [12]. Afortunadamente, se ha llegado a un alto grado de similitud en los lenguajes de programación, de tal manera que si el estudiante de ingeniería aprende la programación en Java, C++ o Python, le será relativamente fácil adaptarse a la programación en MATLAB, o viceversa. El autor en su experiencia docente ha implementado el material que se muestra en este documento en los cursos de matemáticas básicas y cálculo diferencial con estudiantes de

ingeniería, dada la importancia que tiene la programación en la formación de los ingenieros. De la misma forma, la aplicación y el aprendizaje tempranos en las áreas académicas propicia una verdadera construcción del conocimiento, superando entre otras la sola memorización de procedimientos y comandos; también es útil para abordar varios temas de funciones matemáticas de una manera didáctica como estrategia pedagógica para el aprendizaje de los estudiantes, según lo plantea Mauricio Reyes y Vianney Díaz [9]. Se espera aportar con este trabajo un conjunto de recursos útiles, que puedan implementar aquellos docentes que cuentan con MATLAB, para la enseñanza de las matemáticas y el manejo de la programación, que redundan en un aprendizaje más profundo, tanto de MATLAB como de los temas de funciones matemáticas. Se aclara que por ser este un trabajo introductorio en la parte de programación en MATLAB, en la mayoría del código no se realiza validación de datos ni manejo de errores.

2. Elementos de programación en MATLAB®

Modelo de flujo de control *if-else-end*

Con la instrucción *if-else-end* es posible cambiar la secuencia en que se ejecutan otras instrucciones, dependiendo de que se cumplan algunas condiciones. La forma general de esta instrucción [1] es:

```
if condición
 instrucciones
elseif condición
 instrucciones
else
 instrucciones
end
```

A continuación se muestran varios scripts para ilustrar diferentes usos y simplificaciones de este condicional.

Script 1. Se simula un bono que para los menores de edad tiene un valor de $200.000 y si es mayor de edad, de solo $50.000. Observe los mensajes correspondientes según el flujo que presenta el programa. Observe el siguiente script guardado con el nombre *edad_y_bono.*

```
% Ingresar en una variable edad, la edad de una persona:
edad=input('Digite la edad: ')
if edad<18
 bono=200000;
 fprintf('La persona es menor de edad \n')
else
 bono=50000;
 fprintf('La persona es mayor de edad \n')
end
fprintf('el bono es de %g \n',bono)
```

Al ejecutar e ingresar valores de edad de 12 y luego 37, se tiene:

```
>> edad_y_bono
Digite la edad: 12
edad = 12
La persona es menor de edad
el bono es de 200000
>> edad_y_bono
Digite la edad: 37
edad = 37
La persona es mayor de edad
el bono es de 50000
```

Operadores lógicos y su uso en la creación de gráficas a trozos

Para representar gráficas de funciones definidas a trozos, se utilizan los índices o variables lógicas.

Script 3. Para graficar la función:

$$f(x)=\begin{cases} x^2+1 & \text{si } x<0 \\ 1 & \text{si } 0\le x<1 \\ -x+2 & \text{si } x\ge 1 \end{cases}$$

En MATLAB se modela la función mediante un Script que se muestra a continuación, así como la figura generada al ejecutarlo:

```
%Funcion a trozos
x=-4:0.05:4; %Introduce un vector en el intervalo
%Se define la función mediante:
y=(x.^2+1).*(x<0)+1.*((0<=x)&(x<1))+(-x+2).*(x>=1);
plot(x,y,'r') %dibuja la función en color rojo -'r'
axis([-3 3 -2 5]) %Se ajustan los ejes
title('Función definida a trozos') %Inserta título
xlabel('x') %Inserta rótula para eje x
ylabel('y') %Inserta rótula para eje y
grid on %Inserta una cuadrícula
```

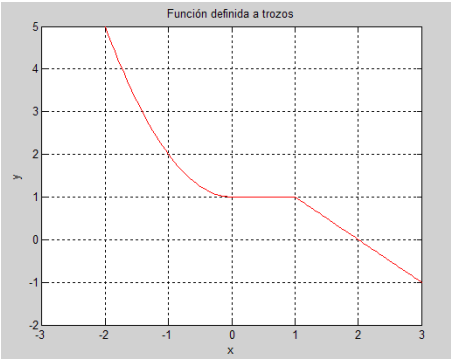

Fig. 1. Función definida a trozos.

MATLAB evalúa la expresión lógica como 1 o 0 para Verdadero o Falso respectivamente, de allí que el dominio de la función para cada parte se pueda tomar en una expresión lógica (que da 1 o 0) y luego se multiplique este 1 o 0 vectorialmente (número a número de dicho vector) según la expresión de la función.

Explorando la continuidad de una función

De la definición de continuidad [4] de una función en x=a:

i. f(a)=b (la función está definida en x=a)

iii. $\lim_{x\to a} f(x) = c$ (el límite existe)

iii. b=c, es decir, correspondan los criterios i y ii.

El estudio de conceptos matemáticos se puede reforzar mediante el planteamiento y resolución de problemas [7]. Así, para la continuidad de una función, se puede profundizar con el resultado gráfico en un ejercicio planteado. Por ejemplo, dada la función definida a trozos:

$$f(x) = \begin{cases} x, & x < -2 \\ (x+a)^2 + c, & -2 \le x < 1 \\ -x + b, & x \ge 1 \end{cases}$$

Se deben encontrar algunas soluciones para los valores de **a, b y c** para que la función sea continua en todos los números reales y mostrar sus gráficas.

Se espera que el estudiante pruebe teniendo en cuenta las tres condiciones para que una función sea continua, así al evaluar la continuidad en los extremos de las funciones que definen la función principal, por ejemplo en x=-2, se debe cumplir:

i. f(-2) esté definida

ii. $\lim_{x\to -2} f(x)$ exista (es decir, por la izquierda y por la derecha de -2, el límite sea el mismo)

iii. $\lim_{x\to -2} f(x) = f(-2)$

Así, teniendo en cuenta estos criterios y luego del análisis y desarrollo (se llega a infinitas soluciones), se pueden obtener soluciones particulares como a=1, c=-3, b=2, al graficar:

```
x=linspace(-4,4,300);
y=x.*(x<-2)+((x+1).^2-3).*((x>=-2)&(x<1))+(-x+2).*(x>=1);
plot(x,y)
title('función a trozos')
axis([-4 4 -4 2])
grid on
```

Cuya gráfica es:

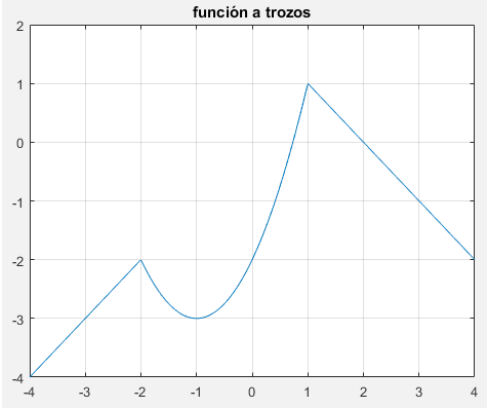

Fig. 2. Función a trozos para los valores a=1, c=-3, b=2.

Al visualizar la gráfica se puede constatar que esta es continua, con lo cual se tiene la certeza de haber encontrado una solución correcta al problema planteado. Esta es una de las posibles soluciones. Se puede plantear al estudiante que encuentre al menos otra solución.

3. Ciclos en MATLAB

La instrucción *for* permite formar ciclos de instrucciones que se repiten un número definido de veces. La sintaxis [1] es:

```
for variable=expresión
  Intrucciones
End
```

En esta instrucción, si se cumple que *variable = expresión*, entonces se ejecutan todas las instrucciones dentro del ciclo. Si no se cumple, el flujo del programa sigue a la siguiente línea de instrucciones después de *end*.

En los siguientes scripts se grafican en una misma figura varias funciones para observar transformaciones específicas [5], [4]. Cada curva o función tiene su propio color.

Script 3. Transformaciones de funciones matemáticas. En este script se generan 7 parábolas con base en y=x^2, la cual se grafica en color rojo, se utiliza *hold on* para que todas aparezcan en la misma gráfica y al final se utiliza *grid on* para producir una cuadrícula.

```
%Tema: Transformación de funciones
 % Desplazar vertical y horizontalmente una parábola.
% a) Desplazamientos verticales
x=linspace(-5,6,50); % se crea un vector
    hold on % todas las gráficas en la misma figura
for c=-3:3 % Ciclo for con el vector c=-3 -2 -1 0 1 2 3
    y=x.^2+c; % parábola desplazada en el valor c, inicia con -3.
if (c==0) % evalúa condición para dar color rojo a y=x^2
   plot(x,y,'r')
elseif (c<0) % evalúa condición para dar color verde a
% las parabolas por debajo de y=x^2
   plot(x,y,'g')
else
  % evalúa condición para dar color azul a
% las parabolas por encima de y=x^2
   plot(x,y,'b')
end
grid on
end
title('Desplazamientos verticales de la parabola y=x^2')
```

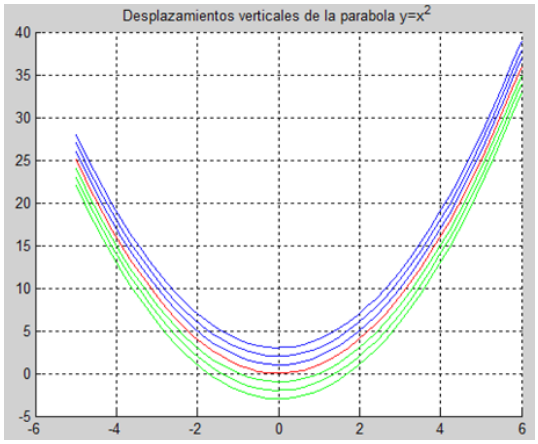

Fig. 3. Traslaciones verticales de la parábola y=x2.

Script 4. Desplazamientos horizontal y vertical:

```
x=linspace(-5,6,50);
for b=-2:2 %El término b para desplazamiento vertical
 for c=-3:3 % El término c para desplazamiento horizontal
   y=(x+c).^2+b;
 if (c==0)
   plot(x,y,'r')
 elseif (c<0)
   plot(x,y,'g')
 else
   plot(x,y,'b')
  end
 end
hold on
grid on
end
title('Desplazamientos horizontales y verticales de la parabola y=x^2')
```

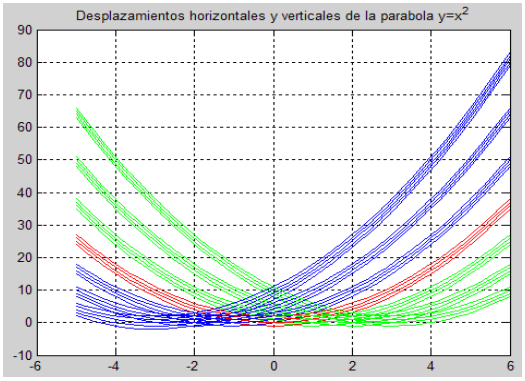

Fig. 4. Traslaciones vertical y horizontal de la parábola y=x2.

Ahora se presentan los siguientes scripts para realizar otras transformaciones de las funciones.

Script 5. Para ampliar y estrechar una función:

```
% Ampliando y estrechando una parabola
x=-3:0.1:3; %Vector para definir dominio de la función: [-3,3]
  %con incrementos de 0.1. También puede usarse la función
  % linspace así: x=linspace(-3,3,100) donde el número 100
  %indica el número de valores intermedios entre -3 y 3.
% Ampliando -o Estrechando - verticalmente: y=cf(x)
hold on % Comando para hacer otra gráfica en la misma figure
            % si desea una nueva gráfica digite figure.
for c=1:4 %Para crear 4 curvas
  y=c.*x. ^2; %Forma de la parábola con un factor c
  if (c==1)
     plot(x,y,'r') %En rojo grafica la parábola y=x^2.
     title('y=cx^2');
 else
     plot(x,y,'b') %En azul grafica las otras parábolas.
 end
end
grid on %Comando para generar una cuadrícula.
title('Transformaciones de la parabola y=cx^2')
```

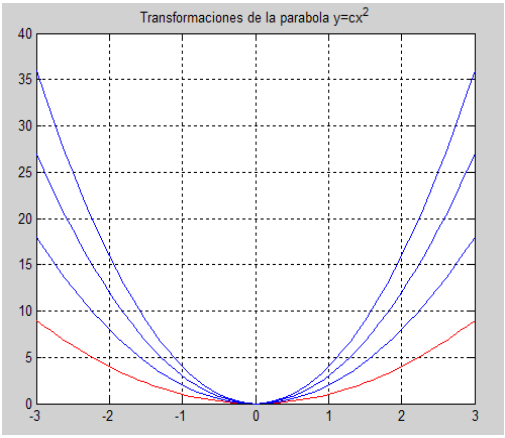

Fig. 5. Transformaciones de la parábola y=cx2 por el factor c para c con valores 1, 2, 3 y 4.

Script 6. Para transformación vertical en un factor 1/c:

```
% Disminuyendo verticalmente: (1/c)f(x)
x=-3:0.1:3;
hold on
for c=1:4
   y=(1/c).*x.^2;
   if (c==1)
       plot(x,y,'r')
   else
       plot(x,y,'b')
    end
end
grid on %Comando para generar una cuadrícula.
title('Transformaciones de la parabola y=(1/c)x^2')
```

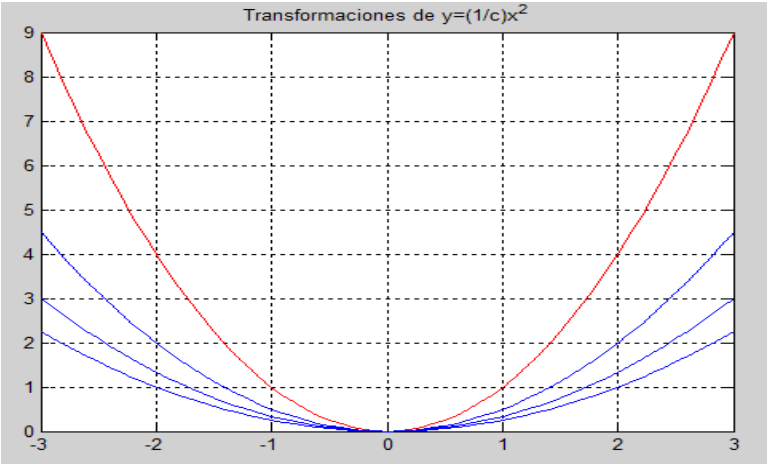

Fig. 6. Transformaciones de la parábola y= (1/c)*x2 por el factor c para c con valores 1, 2, 3 y 4.

Script 7:

```
x=-3:0.1:3;
for c=-3:3
y=c.*x.^2;
if (c==1)
plot(x,y,'r')
else
hold on
plot(x,y,'b')
end
grid on
end
```

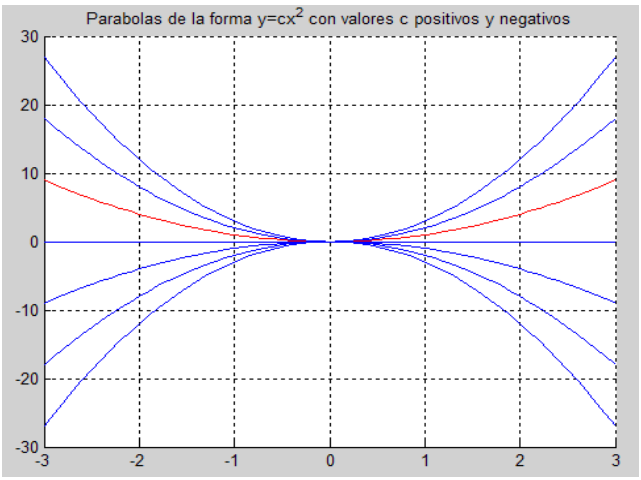

Fig. 7.Parábolas de la forma y=cx2 con el factor c con valores -3, -2,-1, 0, 1, 2, y 3.

Script 8. Para graficar polinomios de varios grados:

```
% Trazando polinomios
x=-3:0.1:3;
 for a=2:4
    y=x.^a;
    if a==2
    plot(x,y,'r')
    elseif (a==3)
      plot(x,y,'b')
    else
      plot(x,y,'y')
    end
    hold on
    title('Polinomios de la forma y=x^a para a entre 2 y 4')
end
 grid on
```

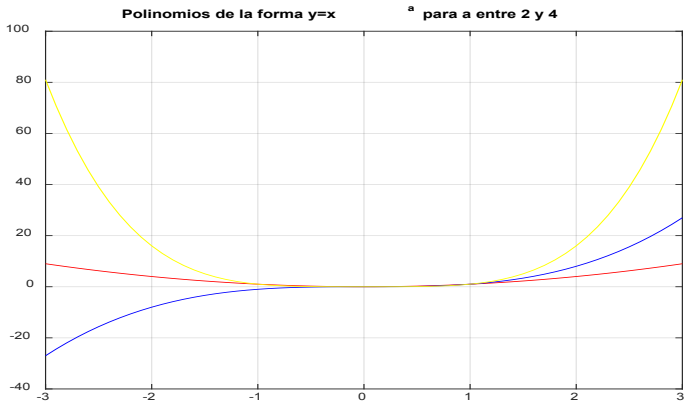

Fig. 8. Polinomios de la forma y=xa para valores de a de 2, 3 y 4.

Funciones trigonométricas

Las funciones seno y coseno tienen la forma general [5]:

$$y = A\cos\left(\frac{2\pi}{T}(x - C)\right) + D$$

Donde A es la Amplitud, T es el período, C es la traslación (o corrimiento horizontal) y D es la traslación (o corrimiento vertical). Así, cuando el período es 2π, la amplitud es 1 y no hay traslaciones se tiene y=cos(x).

En el script 9 se grafican varias curvas de la función coseno con amplitudes A=1, 2, 3 y 4:

```
%Varias amplitudes al tiempo:
x=0:0.1:2*pi; %Se define el vector para el dominio
for A=1:4 % Se definen las Amplitudes: 1,2,3 y 4 en un ciclo
y=A*cos(x); %Se genera el vector y para cada amplitud
if A==1
hold on
   plot(x,y,'r') %Gráfica roja para A=1
else
   hold on
   plot(x,y,'b') %Gráficas azules para otras amplitudes
end
end
title('y=Acos(x) con A=1,2,3,4')
grid on
xlabel('eje X')
ylabel('eje Y')
```

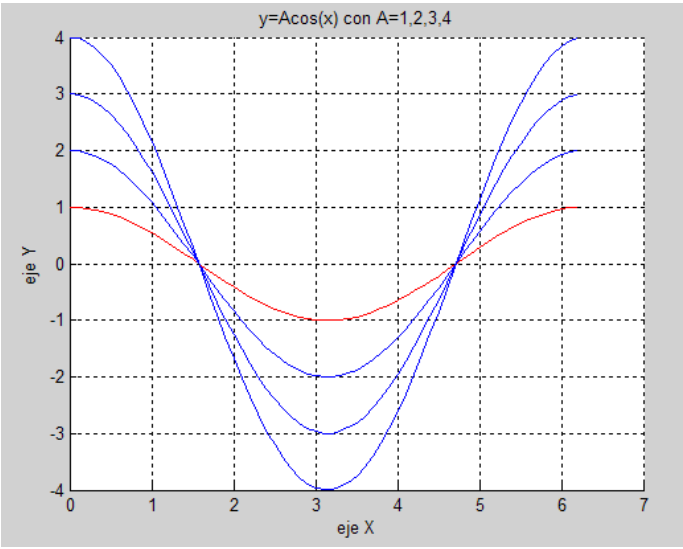

Fig. 9. Varias funciones coseno con amplitudes A=1, 2, 3 y 4.

Script 10 con varias transformaciones de la función coseno y graficadas en la misma figura.

```
%Gráficas del coseno con múltiples periodos
 x=0:0.1:2*pi; %Se define el vector para el dominio
% Se define el factor F=(2*pi/T), así tenemos y=cos(Fx). Se observa el factor según el período:
% T=pi/2-> F=4; T=pi ->F=2, T=2pi ->F=1, T=4pi->F=0.5
% Periodo T
  for F=[0.5 1 2 4] % Se definen los T=pi/2,pi,2pi,4pi
y=cos(F.*x); %Se genera el vector y para cada Periodo T
if F==1
    hold on
    plot(x,y,'r') %En rojo la gráfica con T=2pi
elseif F==4
    hold on
    plot(x,y,'b') %En azul la gráfica con T=pi/2
elseif F==2
    hold on
    plot(x,y,'y') %En azul la gráfica con T=pi
else % en este caso para los otros periodos
    hold on
    plot(x,y,'g') %En verde gráficas con T= 4pi
end
end
title('y=cos((2pi/T)x) con períodos T=pi/2,pi,2pi y 4pi')
grid on
xlabel('eje X')
ylabel('eje Y')
%El siguiente código es para ubicar etiquetas explicativas para cada curva
x4pi = 2;
y4pi = 0.58;
txt1 = '\leftarrow T = 4pi';
text(x4pi,y4pi,txt1)
xpim = 1.7;
ypim = 0.83;
txt2 = '\leftarrow T = pi/2';
text(xpim,ypim,txt2)
xpi = 3.5;
ypi = 0.82;
txt3 = '\leftarrow T = pi';
text(xpi,ypi,txt3)
x2pi = 4.9;
y2pi = 0.92;
txt4 = 'T = 2pi \rightarrow ';
text(x2pi,y2pi,txt4)
```

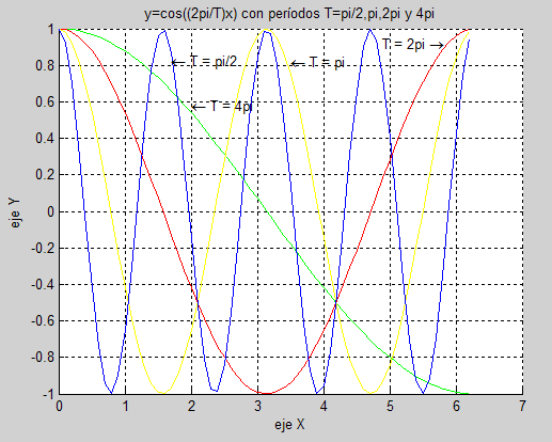

Fig. 10. Varias funciones f(x)=cos((2π/T)x) con períodos T de π/2, π, 2π y 4π.

Script sobre curvas paramétricas

En el script 11 se grafican múltiples curvas de la bruja de Agnesi, estudiada en cálculo diferencial [4], con lo cual se logra una silueta que despierta el interés de los estudiantes. Se aplica un ciclo *for* para variar el factor a entre -10 y 10.

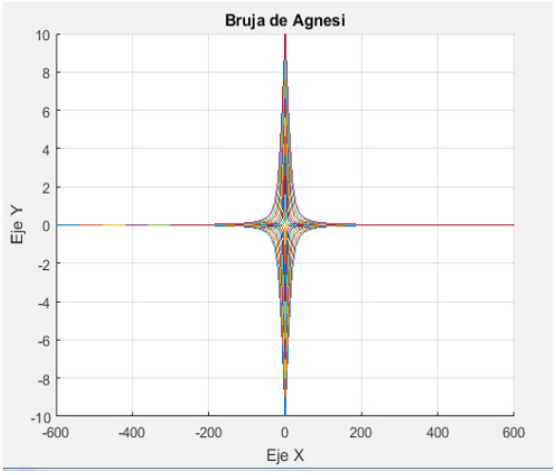

Fig. 11. Varias curvas paramétricas de la bruja de Agnesi, con el factor a con valores enteros de -10 a 10.

4. Funciones en MATLAB

Se tratan a continuación diferentes formas de trabajar con funciones en MATLAB.

Directamente desde el Command Window

La primera forma de crear una función es desde el Command Window, con el comando *inline*. Así, el código en MATLAB para establecer la función f(t) = t ^ 2, g(t) = 2t-5 y hacer h(x) = f(g(x)), que es la función compuesta [4], es:

```
syms x
f=inline('t^2')
g=inline('2*t-5')
h=f(g(x))
```

Y al ejecutar, se observa el siguiente resultado:

```
f =Inline function:
    f(t) = t^2
g =Inline function:
    g(t) = 2*t-5
h =(2*x - 5)^2
```

Crear funciones como archivos

En la definición de una función se requiere uno o varios argumentos o parámetros (x_1, x_2, …, x_M), y puede arrojar uno o varios resultados (y_1, y_2,…, y_M). De esta manera, se define de forma general una función en MATLAB con la sintaxis: function [y1, y2,…, yM] = mifuncion (x1, x2, …, xM), en la que se declara una función llamada *mifuncion*.

Para Guardar (Save) el código de una función, este queda en un archivo de texto con una extensión *.m*. El nombre del archivo debe coincidir con el nombre de la primera función en el archivo. Los nombres de función válidos comienzan con un carácter alfabético, y pueden contener letras, números o guiones bajos.

Es útil pensar en una función como una máquina [4]. Si x está en el dominio de la función f, entonces, x entra a la máquina, es aceptada como una entrada y la máquina produce una salida f(x) según el tipo de función.

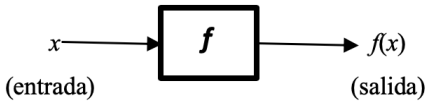

Fig. 12. Diagrama de bloque de una función.

Se ilustra el concepto con el siguiente código, en el que se crea una función en MATLAB con File/New Function M-file, con el nombre cambiov (cambio de variable):

```
function y=cambiov(x)
fprintf('El valor de la variable x al ingresar a cambiov es x=%g \n',x)
x=20; %Redefine el valor de la variable al valor 20
fprintf('El valor de la variable x modificado dentro de cambiov es x=%g \n',x)
end
```

Luego se invoca la función desde el Command Window, con el valor 50:

```
>> cambiov(50)
El valor de la variable x al ingresar a cambiov es x=50
El valor de la variable x modificado dentro de cambiov es x=20
```

Función con una salida (un valor calculado).

Se crea la función promedio3 que acepta tres valores y calcula el promedio de estos como único resultado (el promedio aritmético).

```
function y = promedio3(x1,x2,x3)
%Función para calcular el promedio aritmético de 3 valores
y=(x1+x2+x3)/3;
end
```

La llamada a la función se puede hacer desde la ventana de comandos (Command window); por ejemplo, si un estudiante tiene tres notas y quiere saber el promedio:

```
>>promedio3(4,3,2)
ans = 3
```

También se puede hacer desde un script:

```
a=4;
b=3;
c=2;
res=promedio3(4,3,2);
disp(res)
```

Al ejecutar este script se muestra el resultado que se ha guardado en la variable res.

En la llamada a la función *promedio3* su parámetro ***x*** adquiere el valor del primer argumento ***a***, el segundo parámetro ***y*** toma el valor del argumento ***b***, ***z*** toma el valor de ***c***; se efectúa el promedio aritmético en el cuerpo de la función, se guarda el resultado en la variable ***prom*** que devuelve la función. Al ejecutar el script, el valor que guarda ***prom*** se copia en la variable ***res***. Las variables ***x***, ***y*** y ***z*** son locales a la función y, por lo tanto, no aparecen en la ventana ***Workspace***, no se puede acceder a ellas desde la ventana de comandos.

Se observa que esta función solo calcula el promedio aritmético. Para un promedio ponderado en que no todas las cantidades tienen el mismo valor, se requiere hacer otra operación. Por ejemplo, si se requiere saber el promedio de una materia en la que se tienen tres notas con porcentajes de 30%, 30% y 40%, respectivamente, definimos una función llamada *promediop* de la siguiente manera:

```
function y = promediop(x1,x2,x3)
%Función para calcular el promedio ponderado de 3 valores en que los
% porcentajes son 30%, 30% y 40% respectivamente
y=0.3*x1+0.3*x2+0.4*x3;
end
```

Y al llamarla (invocarla) desde el Command Window para 4, 3 y 2 respectivamente:

```
>>promediop(4,3,2)
ans = 2.9000
```

Observe cómo la nota ahora no es aprobatoria con un mínimo de 3.0. Observe que el orden como se pasan los argumentos es fundamental aquí, dada la forma como se lleva a cabo el cálculo.

Función para calcular el promedio ponderado con vectores como argumentos de entrada. Dado que MATLAB es muy poderoso para operaciones con vectores y matrices, se puede realizar el mismo promedio de forma más general con vectores; en este caso deben ser del mismo tamaño. Por razones del proceso de aprendizaje se muestra una sencilla función para este propósito, pero sin validar los datos de entrada.

```
function y = promediop2(x1,x2)
% Función para calcular el promedio ponderado de los valores ingresados en un vector x1 según
la ponderación dada en el vector %%x2. Se utiliza el producto escalar entre los dos vectores
para calcular este promedio ponderado (el producto escalar multiplica
% los primeros elementos de cada vector y los suma con los correspondientes productos de los
segundos, terceros… n-esimos
% elementos de cada vector):
y=dot(x1,x2);
end
```

Luego se prueba con los mismos datos de la función anterior desde el Command Window:

```
>> x1=[4,3,2];
>> x2=[0.3,0.3,0.4];
>> promediop2(x1,x2)
ans = 2.9000
```

Función Pitágoras. Función para calcular la hipotenusa o un cateto cuando se le envían dos lados de un triángulo rectángulo. Se recomienda al lector consultar la estructura de control ***switch-case***. En este ejemplo se valida el número de parámetros de entrada a la función con el comando ***nargin.***

```
function tpitagoras2(l1,l2,tipo)
%Función que aplica el Teorema de Pitágoras, los argumentos de entrada son dos lados y con
el argumento tipo que es %char, se indica con h: si se desea calcular la hipotenusa, los lados
son los catetos. Aquí toma los argumentos como %los catetos y calcula la hipotenusa con la
formula h=sqrt(c1 ^ 2+c2 ^ 2)
% c: si se desea calcular un cateto, los lados son la hipotenusa y el otro cateto. Se prueba
cuál lado es el %mayor y se aplica el Teorema de Pitágoras para hallar el cateto que falta.
%Paso 1. Validar que se han ingresado los argumentos requeridos: solo dos argumentos, para
esto uso de la función %nargin -number of function input arguments
switch nargin
  case 3 % Cantidad apropiada de argumentos
     %Cálculo del cateto o la hipotenusa según 'tipo'
    switch tipo
        case 'h' %caso para calcular la hipotenusa
           l=sqrt(l1 ^ 2+l2 ^ 2);
           fprintf('La hipotenusa es h=%g \n',l)
        case 'c' % Caso para calcular un cateto
           %se ingresaron la hipotenusa y un cateto, el mayor es la
           %hipotenusa:
          if(l1 >l2)
           l=sqrt(l1 ^ 2-l2 ^ 2);
           fprintf('El cateto es c=%g \n',l);
          else
           l=sqrt(l2 ^ 2-l1 ^ 2);
           fprintf('El cateto es c=%g \n',l);
          end
      end
  otherwise %Cantidad no apropiada de argumentos
      fprintf('Debe ingresar dos argumentos');
  end
end
```

Se ilustran varias corridas a continuación:

```
>> tpitagoras2(10,8,'c')
El cateto es c=6
>> tpitagoras2(6,10,'c')
El cateto es c=8
>> tpitagoras2(6,8,'h')
La hipotenusa es h=10
```

Funciones con argumentos otras funciones

Se ilustra una función que recibe como argumento otra función, así como los argumentos del dominio para graficar. Los parámetros son (no se realiza en estos scripts la validación de entrada de datos ni el manejo de errores):

func: La función que recibe como parámetro.

xmin: Valor mínimo para el dominio.

xmax: Valor máximo para el dominio.

ns: Cantidad de puntos entre *xmin* y *xmax*.

Es decir, la función utiliza el comando linspace para establecer el dominio de la ecuación que va a graficar:

```
function g=graphfunc(func,xmin,xmax,ns)
x=linspace(xmin,xmax,ns);
plot(x,func(x))
disp('Va la gráfica')
g='O.K';
end
```

Para establecer los parámetros es importante seguir la notación:

f=@(x) ecuación

Por ejemplo, invoque la función para que grafique y=cos(x)-x, definiendo f así:

f1=@(x) cos(x)-x

Invoque (llame) la función graphfunc para que realice la gráfica de y=cos(x)-x con dominio D=[0,2pi] y con 50 puntos, de la siguiente manera:

```
>>graphfunc(f1,0,2*pi,50)
```

5. Desarrollo de una aplicación con guide para interfaz gráfica: movimiento parabólico

Se ilustra en este programa el uso de funciones como la cuadrática y el manejo desde MATLAB de funciones en un entorno gráfico. Se ilustra el desarrollo de una aplicación que realiza las gráficas de Posición vs. Tiempo y Velocidad vs. Tiempo en un movimiento parabólico, programa que realiza los cálculos con los datos de entrada que suministre un usuario. Los datos de entrada son velocidad inicial, ángulo de lanzamiento, gravedad y posiciones iniciales sobre cada eje *X* y *Y*. En [3] se modela este movimiento y se genera la gráfica de Altura vs. Tiempo. En [2] se modela el movimiento parabólico en forma vectorial y se generan las gráficas de Altura alcanzada vs. Tiempo, Distancia alcanzada vs. Tiempo y trayectoria del proyectil.

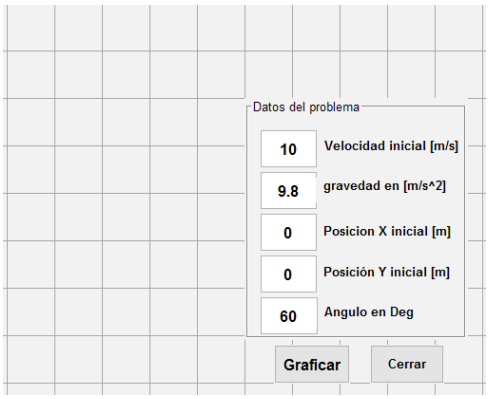

Fig. 13. Interfaz gráfica de la aplicación del movimiento parabólico.

En [11] se trabaja también sobre las ecuaciones de la trayectoria de un misil. El desarrollo de este programa se basa en estos trabajos, y se va más allá graficando también la velocidad horizontal y la vertical y la del cambio de la magnitud de la velocidad, con lo cual se realiza el análisis del movimiento parabólico desde sus fundamentos físicos.

Se desarrolla para esta modelación una interfaz gráfica con GUIDE, como se ilustra en Fig. 13.

En los espacios vacíos se van a generar las gráficas con tres gráficas en la parte superior para la trayectoria (*y* vs. *x*) y las componentes del vector de posición contra el tiempo (*y* vs. *t* y *x* vs. *t*), y tres gráficas de velocidad contra tiempo en la parte inferior (componente vertical, componente horizontal y magnitud de la velocidad vs. tiempo).

Se configuran los diferentes objetos de la interfaz y se procede con la programación de cada cuadro de tipo *Edit Text* que se utilizan para ingreso de datos, y de los botones para proceder con los cálculos. En general, para los cuadros *Edit text* se muestra la programación del primero que es *v0*; sobre este objeto, dé click derecho y seleccione *View Callbacks/callback*, allí agregue las líneas de código que se muestra a continuación:

```
function v0_Callback(hObject, eventdata, handles)
global datos
global vinicial
vinicial=str2double(get(hObject,'String'));
datos(1)=1;
```

De esta forma el programa captura el valor que se ingrese en este objeto y lo guarda en una variable global *vinicial*. De forma análoga se procede con los demás *Edit Text,* guardando en cada caso los valores dados desde la interfaz en las variables globales *gravedad, xinicial, yinicial* y *tetadeg*.

Para el manejo de los cálculos requeridos, se crea una función *mparabolico* para el manejo de los vectores de tiempo (*t*) y de posición *x,y* y la *velocidad*; estos tres últimos vectores se crean en un vector *x* con seis componentes, así la primera es para la posición horizontal *x*, la segunda para la posición vertical, la tercera para el vector magnitud de velocidad total, la cuarta para la velocidad vertical, la quinta para la velocidad horizontal y la sexta para la derivada de la magnitud de la velocidad total. La función es:

```
function [t,x]=mparabolico1(v0,tetad,g,x0,y0)
%Función que calcula los vectores de posición y velocidad con el tiempo dados unos
parámetros como son:
%v0: Velocidad inicial en [m/s]
%tetad: Ángulo de lanzamiento en sistema sexagesimal
%x0: Posición inicial sobre eje horizontal en [m].
%y0: Posición inicial sobre eje vertical en [m].
% g: gravedad, por defecto g= 9.8 m/s^2.
% 1. Ajuste de los grados de sexagesimales a radianes:
```

```
teta=tetad*pi/180;
% 2. Cálculo del vector de tiempos desde 0 hasta el tiempo de vuelo:
%Paso 2.1: Cálculo del tiempo de vuelo tv:
if y0==0
tv=2*v0*sin(teta)/g;
else
   ts=v0*sin(teta)/g;
   ymax=y0+v0*sin(teta)*ts-0.5*g*ts^2;
   vfy=sqrt(2*g*ymax);
   tb=vfy/g;
   tv=ts+tb;
end
%Paso 2.2: Creación del vector con 100 puntos:
t=linspace(0,tv,100);
%Si este comando se vuelve obsoleto, utilice t=[0:tv/100:tv]
%3. generar los vectores x, y, v
for i=1:100
    x(i,1)=x0+v0.*cos(teta).*t(i); %Para vector x
x(i,2)=y0+v0.*sin(teta).*t(i)-0.5*g.*t(i)^2 ;%Para vector y
%genera vector de Magnitud velocidad total:
x(i,3)=sqrt((v0.*cos(teta))^2+(v0.*sin(teta)-g.*t(i))^2);
x(i,4)=v0.*sin(teta)-g.*t(i); %genera vector velocidad vertical Vy
x(i,5)=v0*cos(teta) %genera vector de velocidad horizontal Vx
% Genera vector de la derivada de la magnitud de la velocidad
x(i,6)=(-g.*(v0*sin(teta)-g.*t(i)))/sqrt((v0*cos(teta))^2+(v0*sin(teta)-g.*t(i)).^2);
end
```

Para el botón graficar el código es:

```
function graficar_Callback(hObject, eventdata, handles)
% Graficas del MRUV, dados los parámetros V0, Teta, g, x0, y0
%Ingrese los parámetros:
global datos
global vinicial
global gravedad
global xinicial
global yinicial
global tetadeg
if size(datos)>0
else
   datos=[0 0 0 0 0]
end
% Si no ha ingresado valores, tome por default los datos de la interfaz:
if datos(1)==0
   vinicial=10;
end
if datos(2)==0
   gravedad=9.8;
end
if datos(3)==0
   xinicial=0;
end
if datos(4)==0
   yinicial=0;
end
if datos(5)==0
   tetadeg=60;
end
%Invoque la función de cálculo y generacion de vectores para construcción de %las gráficas
mparabolico1
```

```
[t,x]=mparabolico1(vinicial,tetadeg,gravedad,xinicial,yinicial);
%Construcción de las gráficas
%Gráfica de la trayectoria
subplot(2,4,1)
plot(x(:,1),x(:,2));
grid on
xlabel('Distancia horizontal [m]')
ylabel('Altura [m]')
title('Trayectoria del proyectil')
%Gráfica de la altura vs. tiempo
subplot(2,4,2)
plot(t,x(:,2));
grid on
xlabel('tiempo [s]')
ylabel('Altura [m]')
title('Altura alcanzada vs. tiempo')
%Gráfica de la distancia vs. tiempo
subplot(2,4,3)
plot(t,x(:,1));
grid on
xlabel('tiempo [s]')
ylabel('Distancia x [m]')
title('Distancia alcanzada vs. tiempo')
%Gráfica de la velocidad vs. tiempo
subplot(2,4,7)
plot(t,x(:,3));
grid on
xlabel('tiempo [s]')
ylabel('Velocidad v [m/s]')
title('Magnitud Velocidad vs. tiempo')
%Gráfica de la velocidad vertical vs. tiempo
subplot(2,4,5)
plot(t,x(:,4));
grid on
xlabel('tiempo [s]')
ylabel('Velocidad Vy [m/s]')
title('Velocidad vertical Vy vs. tiempo')
%Gráfica de la velocidad horizontal Vx vs. tiempo
subplot(2,4,6)
plot(t,x(:,5));
grid on
xlabel('tiempo [s]')
ylabel('Velocidad Vx [m/s]')
title('Velocidad horizontal Vx vs. tiempo')
%Gráfica en una nueva figura de la derivada de la magnitud de velocidad V vs. tiempo
hold on
figure
plot(t,x(:,6))
grid on
xlabel('tiempo [s]')
ylabel('d|V|/dt [m/s^2]')
title('Derivada de Magnitud Velocidad vs. tiempo')
```

Al ejecutar y presionar el botón Graficar se tiene la Fig. 14:

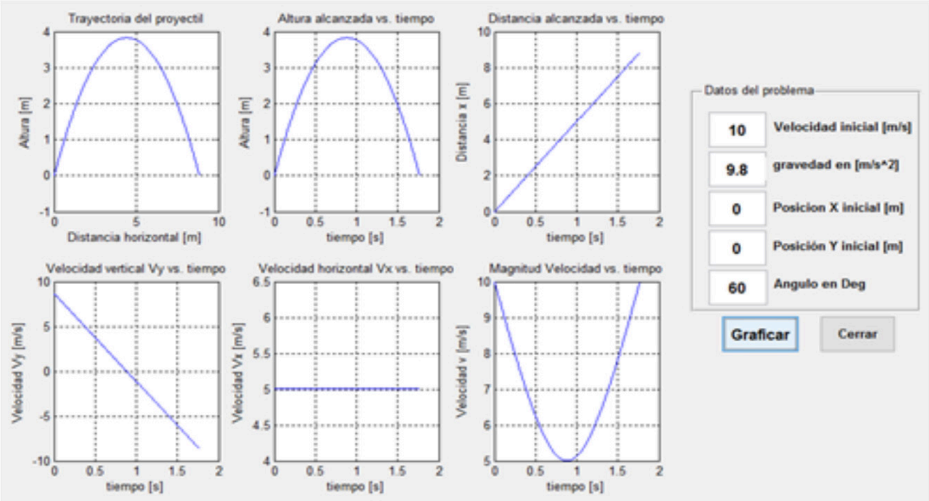

Fig. 14. Salida del programa con los datos indicados.

La gráfica de la magnitud de la Velocidad vs. Tiempo, con la magnitud de velocidad definida como:

$$|V(t)| = \sqrt{Vx^2 + Vy^2} = \sqrt{(v_0\cos\theta)^2 + (v_0\text{sen}\theta - gt)^2}$$

Esta se calcula para cada elemento del vector de tiempo, como se observa en el código. Esta gráfica no es usual que se presente en los textos. Se generó una gráfica adicional en una figura independiente, precisamente de la tasa de cambio (o derivada) de la magnitud de la velocidad total con el tiempo (Fig. 14), y esto que d(|V|)/dt es una aceleración dada por:

$$\frac{d(|V(t)|)}{dt} = \frac{-g(v_0\text{sen}\theta - gt)}{\sqrt{(v_0\cos\theta)^2 + (v_0\text{sen}\theta - gt)^2}}$$

En este artículo, a este cambio en la magnitud de la velocidad total de un cuerpo sometido al movimiento parabólico se le denominará *aceleración de vuelo*.

Se observa, como es bien conocido, que mientras un cuerpo permanece en movimiento parabólico está sometido a la gravedad, por lo que la derivada de Velocidad vertical *Vy* vs. Tiempo es una constante igual a la gravedad (Fig. 14). Para el caso de la magnitud de la velocidad con el tiempo, se observa que en el punto de máxima altura, la magnitud de la velocidad no cambia porque en este preciso instante solo está presente la velocidad horizontal, que es constante. En este punto se tiene un mínimo, y en la gráfica de la Fig. 15 para este valor puntual del tiempo no hay cambio en la magnitud de la velocidad (la aceleración de vuelo es cero).

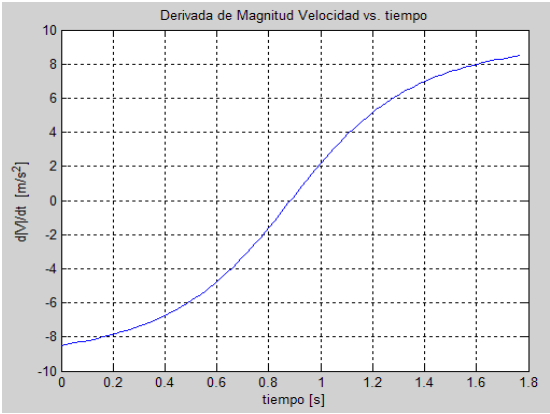

Fig. 15. Gráfica de la derivada de la magnitud de la Velocidad vs. Tiempo (aceleración de vuelo) para los datos indicados en la Fig. 14.

Se observa en la Fig. 15 que mientras el cuerpo sube, el cambio en la magnitud de la velocidad es negativo (aceleración sobre el cuerpo negativa o desaceleración); es cero en el instante de alcanzar la máxima altura y es positivo mientras el cuerpo va bajando, con lo cual representa fielmente el fenómeno físico. Asimismo, este cambio en la magnitud de la velocidad (aceleración del cuerpo) nunca es superior al valor absoluto de la gravedad ($g=9.8\ m/s^2$), que también es obvio por ser el movimiento parabólico una combinación de un movimiento rectilíneo uniforme (sin aceleración) para la componente horizontal y del movimiento uniformemente acelerado por la gravedad para la componente vertical. Por consiguiente, la tasa de cambio máxima posible de la magnitud de la velocidad total de un cuerpo en el movimiento parabólico es la misma gravedad. Entonces, se puede decir que en el límite cuando la velocidad horizontal tiende a cero (se tiene solo velocidad vertical o caída libre), la tasa de cambio de la magnitud de la velocidad total es igual a la gravedad. Según esto, en la Fig. 15, que muestra el cambio de la aceleración de vuelo, se tienen asíntotas horizontales en $a=9.8m/s^2$ y en $a= -9.8m/s^2$. Por supuesto, se piensa más bien en estos valores como los límites de la aceleración de vuelo, ya que el tiempo de vuelo restringe el dominio de la gráfica en el movimiento parabólico. Es importante anotar que el movimiento parabólico aquí analizado y las gráficas mostradas son válidos para velocidades inferiores a la velocidad de escape.

6. Conclusiones

En este artículo se dio la introducción de elementos básicos de programación en MATLAB y, mediante varios ejemplos codificados en scripts o en funciones programadas, se ilustró su aplicación en el estudio de funciones matemáticas.

Se encuentra que en la formación del ingeniero es fundamental el aprendizaje de la programación con lenguajes y herramientas computacionales para la simulación.

Se realizó una simulación con interfaz gráfica para el movimiento parabólico y se halló y graficó, entre otras, la aceleración de vuelo, definida en este artículo como la tasa de cambio de la magnitud de la velocidad total en un movimiento parabólico.

Se comprueba que la simulación de situaciones matemáticas como fenómenos físicos permite profundizar en las características de los mismos.

Referencias

[1] [En línea]. Disponible en: www.mathworks.com

[2] D. Báez López, *MATLAB con Aplicaciones a la Ingeniería, Física y Finanzas*, México: Alfaomega grupo editor, 2006.

[3] H. Vega, *Lógica y algoritmos de programación en Matlab aplicada a la ingeniería*, Colombia: Editorial Bonaventuriana, 2006, pp. 286-292.

[4] J. Stewart, *Cálculo. Conceptos y contextos, 4a edición*, México: Engage Learning Editores, 2010.

[5] T. Jr. George, *Cálculo. Una variable. Decimosegunda edición*, México: Addison - Wesley (Pearson), 2010

[6] R. Serway, y J. Jewett, *Física 1, 3a. edición*, Thomson, 2003.

[7] G. Polya, *How to solve it, 2da. Edición*, Ed., Doubleday Anchor Books, 1957.

[8] M. Salett Bienbengut, «Modelaje matemático en la enseñanza de matemática en la ingeniería: posibilidades y dificultades», *Ingenium, revista de la facultad de Ingeniería*, año 16, no. 31, ene. - jun. 2015.

[9] J. M. Reyes Vergara y V. R. Díaz Pérez, «Didáctica de las ciencias en el contexto de la sociedad del conocimiento», *Ingenium, revista de la facultad de Ingeniería,* año 12, no. 23, ene. - jun. 2011.

[10] G. A. Contreras y P. Carreño, «Simuladores en el ámbito educativo: un recurso didáctico para la enseñanza», *Ingenium, revista de la facultad de Ingeniería*, año 16, no. 31, ene. - jun. 2015.

[11] W. Pinzón Velasco, «Trayectoria de misil balístico intercontinental como trayectoria de vuelo propuesta para una aeronave de turismo espacial tipo SPACE-SHIP-ONE», *Ingenium, revista de la facultad de Ingeniería*, año 1, no. 27, ene. - jul. 2013.

[12] H. M. Vega, «Algoritmo para la solución de determinantes y sistemas lineales por reducción matricial», *Ingenium*, año 7, no. 14, jul. - dic. 2006.