Tema 5

Cálculo Simbólico con MATLAB

El cálculo simbólico utiliza símbolos para representar datos y manipula estos símbolos para encontrar soluciones exactas a los problemas. Esto contrasta con el cálculo numérico, que hemos visto en temas anteriores, que normalmente utiliza números para representar datos y aproximar soluciones a problemas. El cálculo simbólico permite, por ejemplo, simplificar expresiones complejas y proporcionar leyes interpretables que serían difíciles de derivar usando únicamente datos numéricos.

En MATLAB, el cálculo simbólico se realiza utilizando el paquete Symbolic Math Toolbox, que proporciona una serie de funciones específicas para trabajar con variables simbólicas. Este paquete, está basado en el software MAPLE, que fue desarrollado originalmente en 1981 por el Grupo de Cálculo Simbólico en la Universidad de Waterloo, Ontario, y que se basa en un pequeño núcleo escrito en C, que proporciona el lenguaje Maple ("Mathematics Pleasure").

Mediante el uso del cálculo simbólico en MATLAB los usuarios pueden declarar variables simbólicas, crear funciones simbólicas, y realizar operaciones matemáticas simbólicas como suma, resta, multiplicación, división, potenciación, logaritmos, exponenciales y trigonométricas. Además, se pueden calcular derivadas e integrales simbólicas de funciones, resolver ecuaciones algebraicas y diferenciales, y visualizar funciones y gráficos en dos y tres dimensiones. El cálculo simbólico en MATLAB también permite la manipulación de expresiones simbólicas, como factorización, simplificación y expansión.

5.1. Declaración y manipulación de objetos simbólicos en MATLAB

Para trabajar en MATLAB usando el cálculo simbólico, el primer paso es declarar las variables que se van a usar como objetos simbólicos usando las funciones `sym` y `syms`. Por ejemplo, ejecutar en la línea de comandos $\gg$ `syms a b` declara `a` y `b` como variables simbólicas, como se puede comprobar haciendo $\gg$ `whos a b`.

Esta asignación simbólica permite luego utilizar las variables para la tarea que queramos, pudiendo usarlas como constantes, variables o para designar expresiones. En el siguiente código se muestra un ejemplo de operaciones básicas simbólicas:

```
>> % Definimos variables simbolicas:
>>  syms x
>> % Creamos una función basada en x :
>>  f= x^3 - 6*x + 10
```

Ahora, ya podemos realizar operaciones algebraicas sencillas con la función `f`, como suma o multiplicación de variables, suma de constantes o incluso derivarlas e integrarlas, como veremos más adelante. Como ejercicio inicial, pruebe los resultados de hacer:

- `>> f+5`
- `>> f+x`
- `>> f+10*x`
- `>> f/x`
- `>> f^2`

En una expresión simbólica podemos sustituir una variable o una constante para calcular su valor, utilizando la función `subs()`. Por ejemplo, si queremos obtener el valor de f en $x = 2$, simplemente tendremos que escribir `subs(f,x,2)`, es decir sustituimos en la función simbólica f, la variable x por el número 2, cuyo resultado es 19. Hay que tener en cuenta que, al trabajar en simbólico, el resultado sigue siendo simbólico. Si se quisiera obtener el resultado en precisión numérica, se debería usar la función `double`. Como ejemplo:

- `>> a= subs(f,x,2.1)` asigna el valor simbólico `ans= 6661/1000` a la variable `a`.
- `>> a= double(subs(f,x,2.1))` asigna el valor numérico `ans= 6.6610` a la variable `a`.

MATLAB puede trabajar de manera simbólica con funciones de varias variables, mientras estas sean declaradas por anticipado. Por ejemplo, el código

```
>> % Definimos variables simbólicas:
>>  syms x y
>> % Creamos una función basada en x e y:
>>  f= x^3 - 6*y + 10
```

crea la función $f(x, y) = x^3 - 6y + 10$, que se puede manipular de manera similar a cuando f solo depende de x. En este caso, para obtener el valor de la función f en un punto dado por las coordenadas (x, y), por ejemplo $(1, 3)$, hay que modificar la función `subs(variable,entrada,salida)` como `subs(f,[x, y],[1,3])` que nos devuelve `ans= -7`.

Como curiosidad, a veces podemos alternar nuestros programas entre variables numéricas y simbólicas. Por ejemplo, se puede escribir: `A=[1 1/4;1 2*pi]`, que crea una matriz numérica:

$A =$

```
1.0000 0.2500
1.0000 6.2832
```

Si ahora escribimos la expresión `sym(A)`, veremos que no solo recuperamos la variable simbólica de la matriz, sino que MATLAB recupera el valor de variables pre-asignadas, como π:

```
sym(A)
ans=
 [1 1/4]
[1 2*pi]
```

5.2. Operaciones algebraicas con funciones simbólicas en MATLAB

MATLAB permite la simplificación de las expresiones algebraicas que le proporcionemos. Por ejemplo, consideremos la función $f(x) = \frac{1-x^2}{1-x}$. Para obtener su versión simplificada solamente tendríamos que hacer:

```
syms x
simplify((1 - x^2)/(1 - x))
ans=
x+1
```

o, alternativamente, `f=(1 - x ^ 2)/(1 - x); simplify(f).`

MATLAB también comprende las funciones no algebraicas, así,

`≫ syms x y ; ≫ simplify(exp(x)*exp(y))`

devuelve la combinación de ambas exponenciales, `ans= exp(x + y).`
Como ejercicio se deja al estudiante comprobar la relación $\cos(x)^2 + \sin(x)^2 = 1$ usando `≫ syms x y ; ≫ simplify(sin(x)^2 + cos(y)^2)`

Una función de MATLAB, importante cuando se trabaja con calculo simbólico, es la función `assume()` que establece suposiciones o condiciones matemáticas sobre variables simbólicas. La llamada a `assume()` elimina automáticamente todas las suposiciones anteriores sobre las variables indicadas en la condición. Por ejemplo, si quisiéramos trabajar con sumas de logaritmos, por ejemplo para simplificar la expresión $\log(x) + \log(x^2) = \log(x^3)$, podríamos escribir:

`≫ syms x ; ≫ simplify(log(x)+log(x^2)).`

En este caso, como no hemos especificado el valor posible de `x`, MATLAB nos devuelve `log(x^2) + log(x)` en vez de `log(x^3)`. Para obtener la simplificación correcta, hay que "explicar" a MATLAB que solo queremos la solución cuando $x > 0$, para ello, bastaría con escribir `assume(x, 'real')` antes de simplificar.

MATLAB también es capaz de representar las funciones polinomiales simbólicas como el producto de sus factores y desarrollar aquellas expresiones que se encuentran agrupadas. Para ello, se usa la función `expand()`. Consideremos, como ejemplo, la función que hemos agrupado antes, $f(x, y) = e^{x+y}$. Podemos escribir:

`≫ syms x y; ≫ expand(exp(x+y)).`

y obtendremos como resultado: `ans= exp(x)*exp(y)`. Como ejercicio, compruebe que el resultado de expandir $f(x) = (x-1)^3 = x^3 - 3x^2 + 3x - 1$. De manera similar funciona la función `factor()`, que factoriza la expresión (numérica o simbólica) que le pidamos.

Una ventaja que tiene el cálculo simbólico en MATLAB es que permite trabajar con expresiones tal y como se escribirían "con papel y bolígrafo" al estudiar matemáticas. Para ello, se dispone de la función `pretty()` que, aunque no sirve para trabajar después con su resultado, proporciona los resultados de forma más elaborada que como se presentan normalmente, como se puede ver en la Figura 5.1.

```
>> f= (6*x^2 + 5*x +12) / (5*x^4 +2*x + cos(x));
>> pretty(f)
      2
  6 x  + 5 x + 12
--------------------
                4
2 x + cos(x) + 5 x
```

Figura 5.1: Uso de la función `pretty`.

Una función que seguramente utilicemos cuando trabajemos con MATLAB es `taylor()`, que calcula el desarrollo en serie de Taylor alrededor de un punto dado. Por defecto, está programada para devolver el polinomio de grado 5 alrededor del cero, pero es fácilmente editable. Así:

- $\gg$ `taylor(exp(-x))` tiene como resultado `ans = - x^5/120 + x^4/24 - x^3/6 + x^2/2 - x + 1`. El uso de `sympref('PolynomialDisplayStyle','ascend')` cambia el sentido ascendente o descendente de la serie.
- $\gg$ `taylor(log(x),x,1,'Order',3))` calcula el polinomio de Taylor de grado 3 alrededor del punto $x = 1$ del logaritmo de x, resultando en `ans= x - (- 1 + x)^2/2 - 1`.
- $\gg$ `taylor(sin(x),x,0, 'Order',4)))` calcula el polinomio de Taylor de grado 4 alrededor del punto $x = 0$ del seno de x, resultando en `ans= x - x^3/6`.

Es importante tener en cuenta que, para estas operaciones y, en general, si no se especifica lo contrario, MATLAB utiliza como variable preferente la letra `x`. Si la expresión a considerar no contiene la letra `x`, MATLAB, utiliza como variable preferente la letra minúscula más próxima a ella según el orden alfabético (sin considerar la `i` y la `j`, que las reserva para representar la unidad imaginaria).

5.2.1. Álgebra simbólica en MATLAB

Como no puede ser de otra manera, MATLAB dispone de herramientas muy potentes para trabajar con vectores y matrices de forma simbólica. Para ilustrar las capacidades de MATLAB vamos a trabajar directamente sobre un ejemplo.

Supongamos que queremos construir una matriz de Vandermonde, que son muy útiles en la interpolación de polinomios. Estas matrices se caracterizan por presentar una progresión

geométrica en cada fila, de tal manera que $V = (V_{ij})$ si $V_{ij} = (\alpha)_i^{j-1}$. Es decir:

$$V = \begin{pmatrix} 1 & \alpha_1 & \alpha_1^2 & ... & \alpha_1^{n-1} \\ 1 & \alpha_2 & \alpha_2^2 & ... & \alpha_2^{n-1} \\ & & & ... & \\ 1 & \alpha_n & \alpha_n^2 & ... & \alpha_n^{n-1} \end{pmatrix}$$

Una de las características principales de estas matrices es que su determinante se puede expresar con la siguiente fórmula general:

$$|V| = \prod_{1 \le i \le j \le \hat{} n} (\alpha_j - \alpha_i)$$

Para crear una matriz, por ejemplo de 3×3 de Vandermonde canónica en MATLAB podemos hacer simplemente:

```
% declaro variables
syms A
N=4;
v = sym('v', [N, 1]);
A = v.^(0:1:N-1)
```

que nos devuelve la matriz deseada:

```
[1, v1, v1^2, v1^3]
[1, v2, v2^2, v2^3]
[1, v3, v3^2, v3^3]
[1, v4, v4^2, v4^3]
```

Supongamos ahora que queremos construir una matriz de Vandermonde con $\alpha = [1, 2, -1, -2]$. Para ello podemos escribir:

```
≫ V0='[1 1 1 1; 1 2 4 8; 1 -1 1 -1;1 -2 4 -8]';
≫ V=str2sym(V0);
```

que resulta en:

```
[1, 1, 1, 1]
[1, 2, 4, 8]
[1,-1, 1,-1]
[1,-2, 4,-8]
```

Como se puede observar, la primera columna es siempre 1, al estar todos los α_i elevados a 0. **NOTA**: para practicar otros modos de definir variables simbólicas, en este caso, hemos definido la matriz V convirtiendo una cadena de caracteres a variable simbólica.

Se puede calcular el determinante de esta matriz, usando `det()`, al igual que si se tratase de una matriz numérica. Se deja al estudiante comprobar que su valor coincide con la fórmula antes presentada.

De manera equivalente, se puede calcular la matriz inversa:

```
≫ X =inv(V)

X =

 [ 2/3,-1/6, 2/3, -1/6]
 [2/3,-1/12,-2/3, 1/12]
 [-1/6, 1/6, -1/6, 1/6]
 [-1/6, 1/12, 1/6,-1/12]
```

y comprobar que es correcta:

```
≫ V*X

ans=

 [1,0,0,0]
 [0,1,0,0]
 [0,0,1,0]
 [0,0,0,1]
```

Los autovalores simbólicos se pueden calcular usando la función `eig()`:

```
≫ eig(V)

ans=

 [ -6]
 [ 2]
 [-6^(1/2)*1i]
 [6^(1/2)*1i]
```

Recuerde que se puede escribir `[autovec, autoval]= eig(V)` y para obtener los autovalores y los autovectores de manera simultánea. Compruebe, usando las herramientas de cálculo simbólico que esta expresión cumple la formula $|A - \lambda I| = 0$.

Otra característica del álgebra simbólica en MATLAB es que permite obtener el polinomio característico de una matriz, aplicando `charpoli`. Para el caso que estamos usando como ejemplo, bastaría con escribir:

```
syms x; p = charpoly(V,x); p = x^4 + 4*x^3 - 6*x^2 + 24*x - 72,
```

que se puede factorizar (función `factor()`), expandir (función `expand()`) o incluso pedir que MATLAB nos lo devuelva en LaTeX para incorporarlo a un documento (función `latex()`).

5.3. Diferenciación e integración de funciones simbólicas en MATLAB

MATLAB permite la derivación e integración de expresiones simbólicas mediante las funcio-

nes `diff()` e `int()`. Su uso es sencillo, `diff(f,x, n)`: calcula la n-ésima derivada respecto a `x` de una expresión simbólica. Cuando se escribe solamente `diff(f)`, MATLAB deriva respecto de la variable simbólica preferente.

Por ejemplo, podemos calcular las primeras tres derivadas sucesivas de $f = \log(x)$ mediante:

```
% declaro variables
clear
syms x
for n=1:3
    deriv= diff(log(x), 'x', n)
end
```

que nos imprime,`deriv= 1/x, -1/x^2` y `2/x^3`.

Siempre hay que tener en cuenta la variable sobre la que queremos derivar. Por ejemplo, si queremos hacer $\frac{d^2f}{dy^2}$ con $f = x\cos(xy)$ deberemos escribir $\gg$ `diff(x*cos(x*y),y,2)` (lógicamente tras $\gg$ `syms x y`) lo que nos devuelve: $-x^3\cos(xy)$. Si, por el contrario, queremos hacer $\frac{d^2f}{dx^2}$, deberemos escribir $\gg$ `diff(x*cos(x*y),x,2)`, cuyo resultado es $-2y\sin(xy) - xy^2\cos(xy)$. Al igual que lo visto anteriormente para la manipulación de expresiones algebraicas, escribir $\gg$ `f = x*cos(x*y);` $\gg$ `diff(f,y,2);` $\gg$ `diff(f,x,2)` es equivalente a lo anterior.

Existen otras funciones en MATLAB que permiten realizar derivadas de manera rápida, como `gradient()` que calcula el gradiente de una expresión, $\nabla f(x,y) = \left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right)$. Por ejemplo, si queremos calcular el gradiente de $f(x,y) = x^2 + x * y - y - 1$ solo deberemos escribir:

```
clear
% Defino variables
syms x y
% Defino una función f
f= x^2 + x*y - y - 1;
% Calculo su gradiente
Fgrad = gradient(f,[x,y])
```

De manera análoga, la integración se lleva a cabo con el comando `int`, que permite realizar integrales definidas o indefinidas. Así la función `int(f,s)` calcula una primitiva de f respecto de la variable s, $\int f ds$, mientras que `int(f,a,b)` proporciona la integral definida de f respecto de la variable simbólica preferente, $\int_a^b f dx$, y `int(f,s,a,b)` respecto de la variable s, $\int_a^b f ds$.

Como ejemplo, usando $f = ae^{ax}\cos(y)$ vamos a calcular $\int f dx$, $\int f dy$, $\int_0^1 f dx$ y $\int_0^1 f dy$. Para ello tendremos que escribir:

1. $\gg$ `syms x y a b;`
2. Declaro la función: $\gg$ `f = a*exp(a*x)*cos(b*y);`
3. Calculo $\int f dx = e^{ax}\cos(by)$: $\gg$ `int(f); ans= exp(a*x)*cos(b*y)`
4. Calculo $\int f dy = ae^{ax}\left(\frac{\sin(by)}{b}\right)$: $\gg$ `int(f,y); ans= (a*exp(a*x)*sin(b*y))/b`

5. Calculo $\int_0^1 f dx = (e^{ax} - 1)\cos(by)$: $\gg$ `int(f,0,1); ans= cos(b*y)*(exp(a) - 1)`

6. Calculo $\int_0^1 f dy = ae^{ax}\left(\frac{\sin(by)}{b}\right)$: $\gg$ `int(f,y,0,1); ans= (a*exp(a*x)*sin(b))/b`

5.4. Límites de funciones simbólicas en MATLAB

Usando MATLAB podemos calcular el límite hacia un determinado valor de una expresión simbólica. Para ello podemos usar la función `limit()`.

Por ejemplo, podemos calcular $\lim_{x\to 0} \frac{\sin(x)}{x} = 1$ mediante:

```
syms x; limit(sin(x)/x,0), ans = 1
```

o bien $\lim_{x\to \infty} \frac{\sin(x)}{x} = 0$, mediante:

```
syms x; limit(sin(x)/x,inf), ans = 0
```

El limite de MATLAB también es capaz de devolver expresiones, por ejemplo podemos calcular la expresión $\lim_{n\to\infty}\left(1+\frac{x}{n}\right)^n = e^x$;

```
syms x; limit((1 + x/n)^n, n, inf), ans =exp(x)
```

Obsérvese que, en este último caso, hemos calculado el límite cuando $n \to \infty$, especificando que no use la variable preferente `x` sino `n`.

Como curiosidad MATLAB permite el cálculo de límites laterales alrededor de un punto a, $\lim_{x\to a^-}$ y $\lim_{x\to a^+}$ mediante `limit(funcion, a, 'left')` y `limit(funcion, a, 'right')`. Compruebe la validez de estas funciones para el cálculo de $\lim_{x\to 0^-} \frac{x}{|x|} = -1$ y $\lim_{x\to 0^+} \frac{x}{|x|} = 1$.

5.5. Solución simbólica de expresiones algebraicas y de ecuaciones diferenciales ordinarias en MATLAB

MATLAB permite resolver expresiones algebraicas, ecuaciones diferenciales e incluso sistemas de ecuaciones simbólicamente usando el *Symbolic Math Toolbox* mediante las funciones `solve()` y `dsolve()`.

La función `solve` resuelve por defecto el problema homogéneo, asumiendo $f = 0$. Así, si queremos encontrar el valor de x que hace que $\sin(x) = 0$ simplemente deberemos escribir:

- `syms x; f=sin(x); solve(f)`, que nos devuelve el valor 0.

Si, por el contrario, quisiéramos resolver el problema $\sin(x) = 1$, deberíamos escribir:

- `syms x; f=sin(x); solve(f==1)` que nos devuelve el valor $\pi/2$.

Obsérvese que en este último caso hemos realizado una comparación lógica (==) para resolver la ecuación, aunque alternativamente hubiéramos podido usar: `syms x; f=sin(x)-1; solve(f)`.

La función `solve` permite igualmente resolver sistemas de ecuaciones simbólicas. Veamos un ejemplo: queremos resolver el sistema de ecuaciones dado por:

$$x^2 + xy - y = 1$$
$$-3x + 3y = 0.$$

Para ello tenemos dos opciones. La primera pasa por escribir todo de manera agrupada, como por ejemplo: `syms x y ;[x0,y0] = solve(x^2 + x*y - y == 1, - 3*x + 3*y == 0)` que nos resuelve el sistema y nos asigna las soluciones $x0 = (1, -1/2)$ e $y0 = (1, -1/2)$ de manera inmediata. Sin embargo, muchas veces nos encontramos que los problemas no son tan sencillos, por lo que se puede resolver el mismo sistema paso a paso haciendo:

```
clear
% Defino variables
syms x y
% Defino una función f
f= x^2 + x*y - y - 1;
% Defino una función y
g= - 3*x + 3*y;
% Resuelvo para f y g
[x0,y0] = solve(f,g)
```

Para obtener la solución simbólica de ecuaciones diferenciales ordinarias (EDOs), en vez de `solve()`, se debe usar la función `dsolve()` y, evidentemente, especificar las condiciones iniciales o de contorno correspondientes. Vamos a ver su uso mediante varios ejemplos.

Imaginemos que queremos resolver el problema de valor inicial $y' = y^2$. En este caso, lo primero que hay que especificar a MATLAB es la dependencia de la variable y escribiendo: $\gg$ `syms y(t)`. MATLAB es capaz de resolver la ecuación simplemente escribiendo $\gg$ `dsolve(diff(y)==y^2)`, que nos devuelve los posibles resultados de la EDO: `ans= -1 / (C1 + t), 0`. Como se puede observar, al no haber especificado la condición inicial, MATLAB nos devuelve el coeficiente `C1`.

Sin embargo, como norma general, es más conveniente resolver los problemas con EDOs especificando previamente la ecuación y condiciones iniciales o de contorno, `dsolve(edo, condiciones)`, siguiendo el siguiente esquema (aplicado a la EDO anterior):

1. Definir variables simbólicas: $\gg$ `syms y(t)`
2. Escribir ecuación diferencial: $\gg$ `ode= diff(y)==y^2`
3. Escribir condición de contorno/inicial, por ejemplo :$\gg$ `cond = y(0) == 1;`
4. Resolver: $\gg$ `dsolve(ode, cond)`

Como podemos ver, si realizamos los pasos anteriores, la solución es `-1/(- 1 + t)` y ya no depende de `C1`.

Cuando la ecuación que queremos resolver es de orden superior, por ejemplo $y'' - y' = y$ (recordemos: $\frac{d^2y}{dt^2} - \frac{dy}{dt} = y$), se deben especificar las condiciones de contorno/iniciales de manera combinada. Por ejemplo, para resolver $y''-y' = y$ con $y(0) = 0$ y $y'(0) = 1$ el procedimiento sería:

1. Definir variables simbólicas: $\gg$ `syms y(t)`
2. Definir la derivada primera : $\gg$ `Dy=diff(y);`
3. Escribir ecuación diferencial: $\gg$ `ode= diff(y,2)- diff(y)==y;`
4. Escribir la primera condición de contorno/inicil: $\gg$ `cond = y(0) == 0;`
5. Escribir la segunda condición de contorno/inicial: $\gg$ `cond2 = Dy(0) == 1;`
6. Agrupar las condiciones de contorno/iniciales:$\gg$ `conds=[cond cond2];`
7. Resolver: $\gg$ `dsolve(ode, conds)`

Es importante destacar que, al igual que hemos agrupado las condiciones de contorno/iniciales, se pueden igualmente **agrupar varias ecuaciones diferenciales**, lo que es especialmente útil para trabajar con sistemas de ecuaciones.

NOTA: Evidentemente no todas las EDO tienen solución analítica. Si no es posible resolverla analíticamente, MATLAB nos devolverá el mensaje: `Warning: Unable to find symbolic solution` y deberemos intentar resolver la ecuación numéricamente, por ejemplo usando `ode45()`.

5.6. Problema Propuesto

Un oscilador amortiguado forzado es un sistema físico que oscila alrededor de una posición de equilibrio estable, sujeto a la ley de Hooke y amortiguado por un rozamiento en un fluido, además de estar sometido a una fuerza externa restauradora que varía armónicamente con el tiempo.

Para mantener este sistema oscilando, es necesario suministrar energía al mismo, lo que lo convierte en un oscilador “forzado”. La amplitud de las oscilaciones de este sistema puede aumentar si la energía suministrada por la fuerza aplicada es mayor que la disipada por el rozamiento, y permanece constante cuando ambas energías son iguales.

Cuando la frecuencia de la fuerza aplicada se aproxima a la frecuencia natural del sistema, se produce una resonancia en amplitud, lo que lleva a una amplitud máxima en las oscilaciones. Este fenómeno es común en sistemas naturales y se observa en diversos contextos, desde la mecánica o la acústica, hasta la electrónica. En este ejercicio vamos a resolver el problema del oscilador amortiguado forzado y a comprobar cuándo se produce la resonancia.

Consideremos el oscilador amortiguado forzado de la Figura 5.2, sujeto a una fuerza oscilatoria $F(t) = A_0 * \sin(\omega t)$.

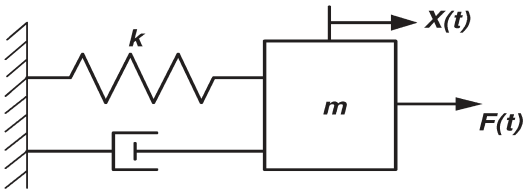

Figura 5.2: Esquema de un oscilador amortiguado forzado.

Si hacemos el equilibrio de fuerzas sobre el sistema podemos escribir:

$$\sum F = ma \rightarrow F(t) - kx - c\frac{dx}{dt} = m\frac{d^2x}{dt^2},$$

donde k y c son las constantes del muelle y del amortiguador, respectivamente, y hemos escrito la velocidad como $\frac{dx}{dt}$ y la aceleración como $\frac{d^2x}{dt^2}$. Si dividimos la ecuación anterior por la masa, podemos escribir:

$$x'' = -\omega_0^2 x - bx' + A * \sin(\omega t)$$

donde $\omega_0^2 = k/m$ es la frecuencia natural del sistema elevada al cuadrado, $b = c/m$ y $A = A_0/m$. Esta ecuación debe complementarse con dos condiciones iniciales, que pueden ser la posición y la velocidad iniciales, $x(t = 0) = x_0$, $x'(t = 0) = v_0$.

Para resolver este problema usando el paquete simbólico de MATLAB, deberemos seguir el mismo esquema visto en el apartado anterior.

```
%
% Declaro variables simbólicas
syms x(t) w02 b A w x0 v0

% Declaro la derivada de x
Dx=diff(x);

% Condiciones iniciales
cond = x(0) == x0;
cond2 = Dx(0) == v0;
conds=[cond cond2];

% Ecuación del Amortiguador Forzado
ode= diff(x,2) == -w02*x - b*diff(x) + A*sin(w*t);

%% Solución Analítica
sol=dsolve(ode,conds);
```

Como podemos observar, hemos resuelto analíticamente la ecuación, pero no podemos visualizar su resultado, aunque sabemos que el sistema estará amortiguado si $\omega \gg \omega_0$ y entrará en resonancia cuando $\omega \sim \omega_0$. Para comprobarlo vamos a asumir que, inicialmente, el oscilador se encuentra en la posición $x(0) = 0{,}2$ m y que tiene una velocidad de $x'(0) = 0{,}8$ m/s. Usemos para visualizar el ejemplo los valores de $\omega_0 = 1$ s^{-2}, $A = 0{,}14$N/kg y $b = 0{,}2$ s^{-1}.

Para ver los dos casos, $\omega \gg \omega_0$ y $\omega \sim \omega_0$, vamos a crear dos conjuntos de condiciones, uno con $\omega = 50$ s^{-1} y otro con $\omega = 1{,}1$ s^{-1}:

```
% Parametros a sustituir
param=[x0 v0 w02 w A b];

% Valores amortiguados, de los parametros [x0 v0 w02 w A b];
% Observe que w es mucho mayor que w0
value_am=[0.2 0.8 1 50 0.1 0.2];

% Valores de resonancia, de los parametros [x0 v0 w02 w A b];
% Observe que w se aproxima a w0
value_res=[0.2 0.8 1 1.1 0.1 0.2];
```

Una vez hecho esto, solo nos queda sustituir los valores en la solución calculada y representar cada caso usando la función `fplot()` presentada en el tema anterior:

```
%% Sustiyo los valores en la solución analítica
sol_am=subs(sol, param,value_am); % solución amortiguada
sol_res=subs(sol, param,value_res); % solución cerca de la resonancia
```

A continuación se puede observar como quedaría el código completo, cuyo resultado se muestra en la Figura 5.3.

```
% Limpio entorno, cierro figuras
clear; close all;

% Declaro variables simbólicas.
syms x(t) w02 b A w x0 v0

% Declaro Diferencial
Dx=diff(x);

% Condiciones Iniciales
cond = x(0) == x0;
cond2 = Dx(0) == v0;
conds=[cond cond2];

% Ecuación del Amortiguador Forzado
ode= diff(x,2) == -w02*x - b*diff(x) + A*sin(w*t);

% Solución Analítica
sol=dsolve(ode,conds);

% Parámetros a sustituir
param=[x0 v0 w02 w A b];


% Valores amortiguados, de los parámetros [x0 v0 w02 w A b]
% Observe que w es mucho mayor que w0
value_am=[0.2 0.8 1 50 0.1 0.2];

% Valores de resonancia, de los parametros [x0 v0 w02 w A b]
% Observe que w se aproxima a w0
```

```
value_res=[0.2 0.8 1 1.1 0.1 0.2];

% Sustiyo los valores en la solución analítica
sol_am=subs(sol, param,value_am);
sol_res=subs(sol, param,value_res);

% Represento las soluciones
figure;
subplot(2,1,1);
fplot(sol_am,[0 60],'LineWidth',2) % Evalúo de t=0s a t=60 s
grid on;
set(gca, 'FontSize',14);
xlabel('tiempo␣(s)', 'FontSize',14);
ylabel('Posicion␣(m)', 'FontSize',14);
title('Oscilador␣amortiguado', 'FontSize',14);

subplot(2,1,2);
fplot(sol_res,[0 60],'LineWidth',2) % Evalúo de t=0s a t=60 s
grid on;
set(gca, 'FontSize',14);
xlabel('tiempo␣(s)', 'FontSize',14);
ylabel('Posicion␣(m)', 'FontSize',14);
title('Oscilador␣en␣Resonancia', 'FontSize',14);
```

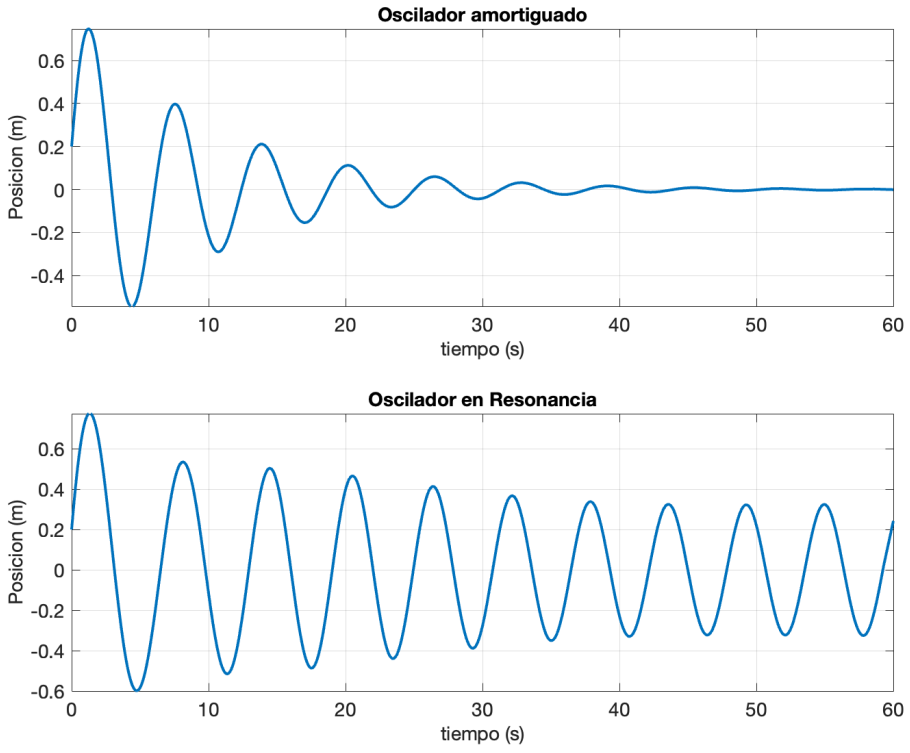

Figura 5.3: Solución amortiguada y en resonancia del oscilador armónico amortiguado forzado.