Laboratorio 5: Manejo de polinomios en MATLAB.

Sebastián Valencia Calderón
201111578

1 Introducción

Los polinomios son estructuras algebraicas que relacionan números complejos a través de la suma de términos no lineales sobre la variable perteneciente al dominio de la función. La estructura de los polinomios es tan versátil que además de ser una función continua y diferenciable, el conjunto de los polinomios con las operaciones fundamentales sobre ellas, constituye un espacio vectorial. Por esta misma naturaleza, los polinomios pueden ser tratados como estructuras discretas, lo que resulta ideal y pertinente para un óptimo tratamiento de ellos usando un computador digital. Existe una gran variedad de algoritmos fundamentales que hacen parte de un conjunto de algoritmos útiles para el tratamiento de polinomios y otras estructuras en áreas como computación científica, comunicaciones, procesamiento de señales, estadística, etc.

Una notación conveniente para el tratamiento de polinomios es la siguiente:

$$P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i$$

La última parte de la igualdad, sugiere la siguiente representación:

$$\begin{bmatrix} a_0 & a_1 & \dots & a_{n-1} & a_n \end{bmatrix} \begin{bmatrix} 1 \\ x \\ \vdots \\ x^{n-1} \\ x^n \end{bmatrix} = \sum_{i=0}^{n} a_i x^i$$

El vector fila de la anterior relación, sugiere una representación vectorial de los coeficientes del polinomio, este vector contiene toda la información del polinomio, el grado y los coeficientes. Con esta representación vectorial, es fácil tratar y almacenar polinomios. De hecho, esta es la representación de polinomios en MATLAB. Dado un polinomios, conviene realizar las operaciones básicas sobre el (suma, resta, producto, división, evaluación, derivación y hallar las raíces de un polinomio). El desarrollo de este laboratorio, pretende explorar la representación y manipulación de polinomios en MATLAB. El desarrollo de estos ejercicios, pretende comprometerse con los siguientes objetivos:

1. Introducir de forma experimental algunos de los conceptos relacionados con el cálculo de raíces de un polinomio de grado n.

2. Identificar la representación de polinomios de cual hace uso MATLAB y su relación con el álgebra matricial.

3. Reconocer algunas de las funciones incluidas en MATLAB para trabajar con polinomios.

2 Procedimiento

Para cumplir los objetivos enumerados anteriormente, se desarrollan ejercicios propuestos sobre ciertos algoritmos, hallando las relaciones obtenidas a partir de ellos mediante el uso de suma, productor, división y diferenciación. Para cada uno de estos polinomios, se evalúa el mismo en un rango pertinente y se gráfica cada uno de ellos. Además, las raíces de los mismos se hallan y se grafican en el plano complejo.

El desarrollo de cada uno de estos ejercicios, requiere comprender el funcionamiento y concepto de cada una de las funciones dispuestas en MATLAB para el manejo de polinomios. El desarrollo del laboratorio, comienza con la comprensión de cada uno de estos métodos o funciones.

3 Resultados

A continuación, se exponen los resultados, las metodologías propuestas para los análisis y las herramientas de ejecución para cada uno de los problemas propuestos.

3.1 Manejo de polinomios en MATLAB

Se estudia la documentación sobre el uso y los conceptos involucrados en las funciones más relevantes para el manejo de polinomios en MATLAB. La referencia principal para entender cada uno de estos procedimientos, es [3], y [2]. Los ejemplos, fueron obtenidos en la documentación de MATLAB.

- `roots(p)`: retorna a manera de vector, las raíces de un polinomio cuya representación está dada por el vector columna p. El vector de entrada, consiste en el vector en $\begin{bmatrix} a_0 & a_1 & \dots & a_{n-1} & a_n \end{bmatrix} \in \mathbb{R}^{n+1}$, el cual a su vez, representa un polinomio $P_n(x) = a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0$. La función `roots(p)`, resuelve la ecuación $a_n x^n + a_{n-1}x^{n-1} + \cdots + a_1 x + a_0 = 0$, donde la variable x, no posee exponentes negativos. Un ejemplo de uso es:

```
p = [1 0 0 0 -1];   % vector representing the polynomial x^4 - 1
r = roots(p);       % a vector of roots or zeros of the polynomal
    represented by p
```

  Ahora la variable r, contienen los números complejos que satisfacen la ecuación $x^4 - 1 = 0$. Para calcular las raices, del polinomio, MATLAB calcula los valores propios de una matriz cuya diagonal contiene los coeficientes del vector p.

- `conv(p, q)`: retorna la convolución (en el contexto de procesamiento de imagenes, la convolución representa el área en la que dos vectores u y v se traslapan. Algebraicamente, es una operación equivalente a la multiplicación de polinomios) de dos

vectores. El parámetro opcional `shape`, especifica si el vector resultante debe tener el tamño de alguno de sus dos parámetros. Si éste último es `same`, retorna la misma parte que p, si es `valid`, retorna la parte de la convolución sin ceros en los extremos. Si se quiere multiplicar los polinomios $x^2 + 1$ y $2x + 7$ en MATLAB, se debe ejecutar las siguientes instrucciones.

```
u = [1 0 1];        % defines the polynomial x^2 + 1 pointed by u
v = [2 7];          % defines the polynomial 2*x + 7 pointed by v
w = conv(u, v);     % polynomial product between u and v
```

Ahora la variable w, contiene una representación del polinomio $2\,x^3 + 7\,x^2 + 2\,x + 7$.

- `deconv(p, q)`: aplica el proceso inverso a la deconvolución entre los vectores p y q, haciendo uso de división larga. El vector cociente, es devuelto en el primer valor de retorno, y el residuo en el segundo, de tal manera, $p =$ `conv(q, quatient)` + `remainder`. El siguiente script, se ejecuta para demostrar el funcionamiento de `deconv(p, q)`, y para verificar este hecho:

```
u = [ 1 2 3 4]; % defines the polynomial x^3 + 2*x^2 + 3*x + 4 pointed
    by u
v = [10 20 30]; % defines the polynomial 10*x^2 + 20*x + 30 pointed by
    v

c = conv(u, v); % polynomial product between u and v
[q, r] = deconv(c, u);  % quotient and remainder of long division
    between c and u
```

La variable c, toma un valor para representar el polinomio $10\,x^5 + 40\,x^4 + 100\,x^3 + 160\,x^2 + 170\,x + 120$, la variable q, representa el polinomio $10\,x^2 + 20\,x + 30$, y $r = 0$. Es fácil verificar que $c =$ `conv(u, q)` + `r`.

- `polyder(p)`: retorna la derivada del polinomio p, determinada por el orden y los coeficientes del mismo. Un segundo parámetro asignado a únicamente una variable (`k = polyder(a, b)`), representa la derivada del producto de dos polinomios, mientras que la función aplicada a dos argumentos y asignada a un vector de dos elementos (`[q, d] = polyder(a, b)`), retorna la derivada del cociente entre dos polinomios. Dado que el primer uso puede ser usado para calcular los otros dos, se incluye un script ejemplificando este primer caso.

```
p = [3 0 -2 0 1 5]; % defines the polynomial 3*x^5 - 2*x^3 + x + 5
    pointed by p
q = polyder(p); % the derivative of p, is now pointed by q
```

En este caso, p, representa al polinomio $3\,x^5 - 2\,x^3 + x + 5$, del cual su derivada es $15\,x^4 - 6\,x^2 + 1$, representada en el vector q.

- `polyval(p, x)`: retorna el valor de evaluación del polinomio p, de grado n, en el valor x, es decir, reemplaza la variable del polinomio por el valor de x y calcula la suma de términos. A continuación, se ejemplifica su uso:

```
p = [3 2 1];         % polynomial whose value is 3*x^2 + 2*x + 1 is now
    pointed by p
v = polyval(p, 2);   % v holds the value of p(2)
```

Después de la ejecución de las instrucciones anteriores, p, representa el polinomio $3\,x^2 + 2\,x + 1$, cuyo valor en $x = 2$, es 17, y es el valor de la variable x.

3.2 Aplicación de conceptos

Sean:

$$f(x) = x^2 + 2x + 3 \qquad g(x) = 9x^4 + 4x^3 + 3x^2 + 2x + 1$$

1. La generación de polinomios, se logra a través de:

```
f = [1 2 3];        % Vector representation of x^2 + 2x + 3
g = [9 4 3 2 1];    % Vector representation of 9x^4 + 4x^3 + 3x^2 + 2x +
     1
```

2. Dados los vectores fila usados en el literal anterior para la generación y representación de los polinomios en MATLAB, cada uno de estos representan los siguientes polinomios obtenidos con el siguiente script:

```
symf = poly2sym(f); % Symbolic representation of f(x)
symg = poly2sym(g); % Symbolic representation of g(x)
```

El anterior script, da como resultado: `symf` $= x^2 + 2x + 3$, y `symg` $= 9x^4 + 4x^3 + 3x^2 + 2x + 1$

3. Los polinomios $f(x)$ y $g(x)$, pueden ser usados para obtener los siguientes polinomios:

   - $h(x) = f(x) \times g(x)$
   - $k(x) = f(x) \ / \ g(x)$
   - $v(x) = \partial g(x) \ / \ \partial x$

   En MATLAB, estas operaciones se materializan en el siguiente script.

```
h = conv(f, g);              % Product between f(x) and g(x)
[kq kr] = deconv(f, g);      % Division between f(x) and g(x)
v = polyder(g);              % Derivative of g(x)
```

   La representación simbólica de los polinomios obtenidos es:

   $\texttt{h(x)} = 9\,x^6 + 22\,x^5 + 38\,x^4 + 20\,x^3 + 14\,x^2 + 8\,x + 3.$

   $\texttt{kq(x)} = 0$, $\texttt{kr(x)} = x^2 + 2\,x + 3.$

   $\texttt{v(x)} = 36\,x^3 + 12\,x^2 + 6\,x + 2.$

4. Por medio de la función `root(p)`, se hallan las raíces de cada uno de los polinomios dados o encontrados a partir de ellos:

   - **Raíces de $f(x)$:**
     $$\texttt{roots(f)} = [-1.0000 \pm 1.4142i]$$
   - **Raíces de $g(x)$:**
     $$\texttt{roots(g)} = [0.2021 \pm 0.5956i \qquad -0.4244 \pm 0.3174i]$$
   - **Raíces de $h(x)$:**
     $$\texttt{roots(h)} = [-1.0000 \pm 1.4142i \qquad 0.2021 \pm 0.5956i \qquad -0.4244 - \pm 0.3174i]$$
   - **Raíces de $kr(x)$:**
     $$\texttt{roots(kr)} = [-1.0000 \pm 1.4142i]$$
   - **Raíces de $v(x)$:**
     $$\texttt{roots(v)} = [-0.0000 \pm 0.4082i \qquad -0.3333]$$

5. A continuación, se gráfica cada uno de los polinomios, y sus raíces en el plano complejo. Para esto, se hace uso de la siguiente función definida en MATLAB para este propósito (es necesario incluir la función, ya que es quien usa directamente las funciones `roots`, y `polyval`). Para el posicionamiento y despliegue de cada una de las graficas, se uso la referencia [1], y la referencia [4].

```
function [] = plotpolroots(pol, name, color, max, min)
% pol is a polynomial represented as in Matlab, name is the name of the
% polynomial to be displayed. color is the color desired for the plot
    of
% the same polynomial. [min, max], defines the range of evaluation

    % Plotting the polynomial
    x = linspace(max, min);         % x-axis values
    vval = polyval(pol, x);         % y-axis values
    subplot(2, 1, 1);
    plot(x, vval, sprintf('%s', color), 'LineWidth',2);
    title('Graph')
    xlabel('x')
    ylabel('Evaluation of polynomial')
    legend(sprintf('y = %s(x)', name))
    grid on

    % Plotting the roots
    subplot(2, 1, 2);
    ff = plot(roots(pol), 'k*');
    set(ff, 'Markersize',10);
    title('Roots')
    xlabel('Complex component')
    ylabel('Real component')
    grid on

    % General title, and saving procedures
    ha = axes('Position',[0 0 1 1],'Xlim',[0 1],'Ylim',[0 1],'Box','off
        ','Visible','off','Units','normalized', 'clipping' , 'off');
    text(0.5, 1,sprintf('Graph and roots of polynomial %s(x) = %s',name
        , char(poly2sym(pol))),'HorizontalAlignment','center','
        VerticalAlignment', 'top');
    set(gcf, 'Position', [400 400 1000 400]);
    saveas(gcf, sprintf('../img/%splot.png', name));

end
```

- **Gráfica y raíces de** $f(x)$ A través de la invocación: `plotpolroots(f, 'f', 'y', -100, 100)`

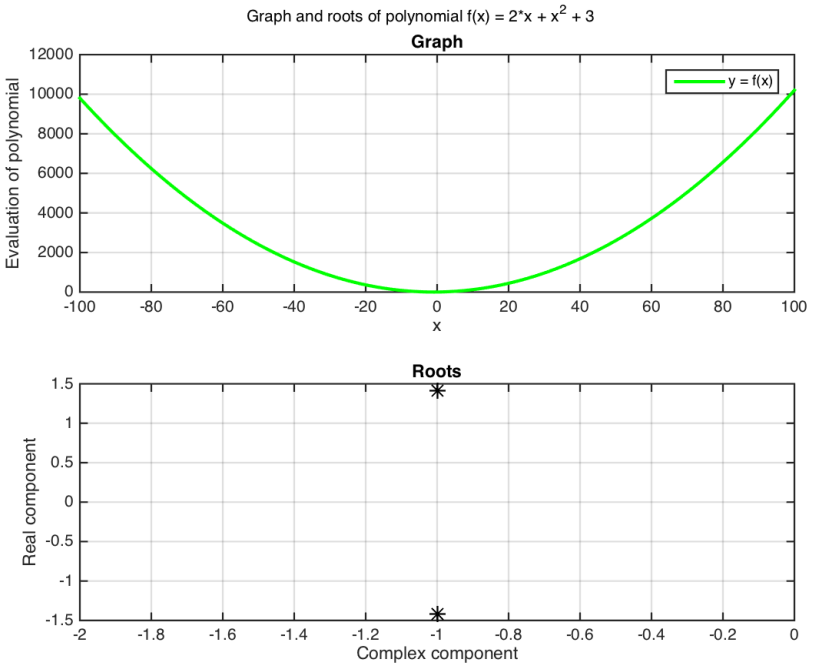

Figura 1

- **Gráfica y raíces de** $g(x)$ A través de la invocación: `plotpolroots(g, 'g', 'm', -1000, 1000)`

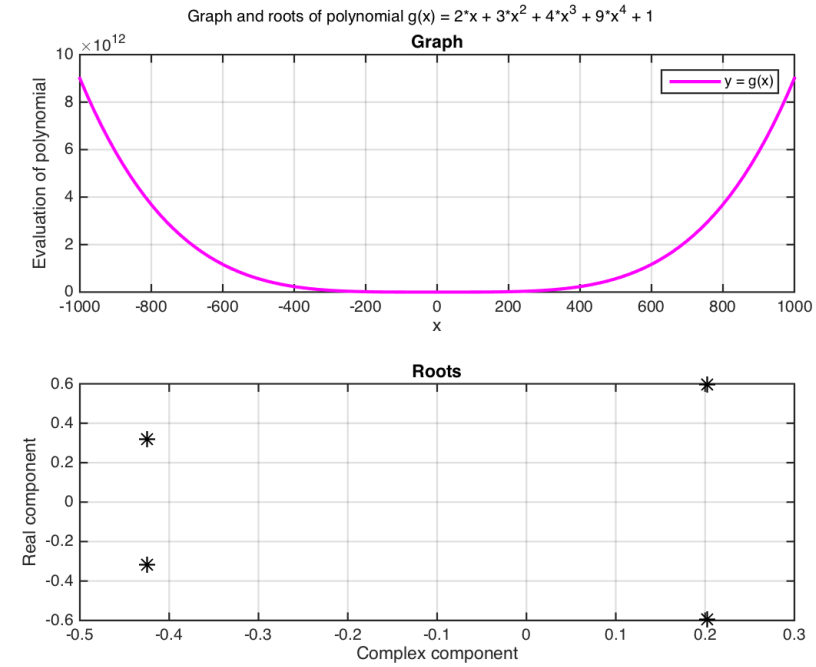

Figura 2

- **Gráfica y raíces de** $h(x)$ A través de la invocación: `plotpolroots(h, 'h', 'c', -1000, 1000)`

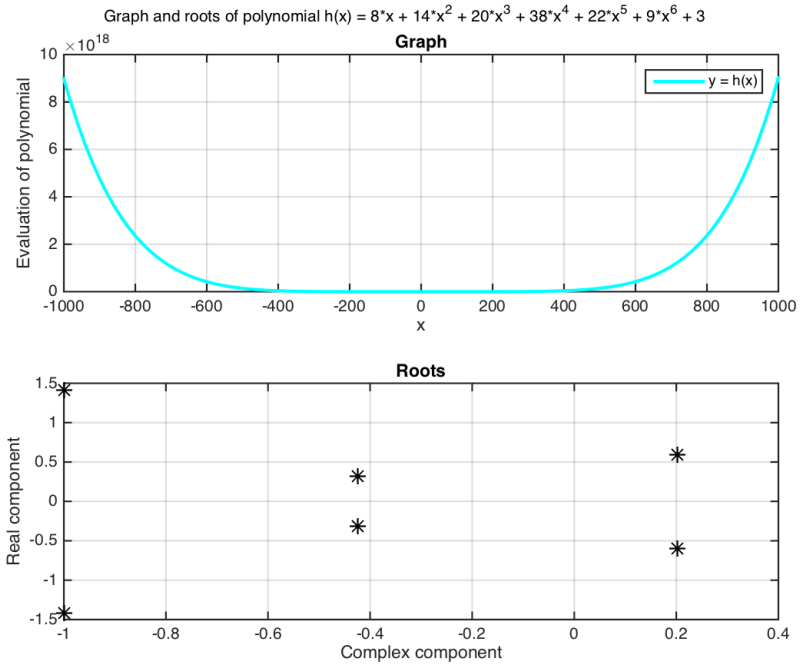

Figura 3

- **Gráfica y raíces de** $kr(x)$ A través de la invocación: `plotpolroots(kr, 'kr', 'r', -100, 100)`

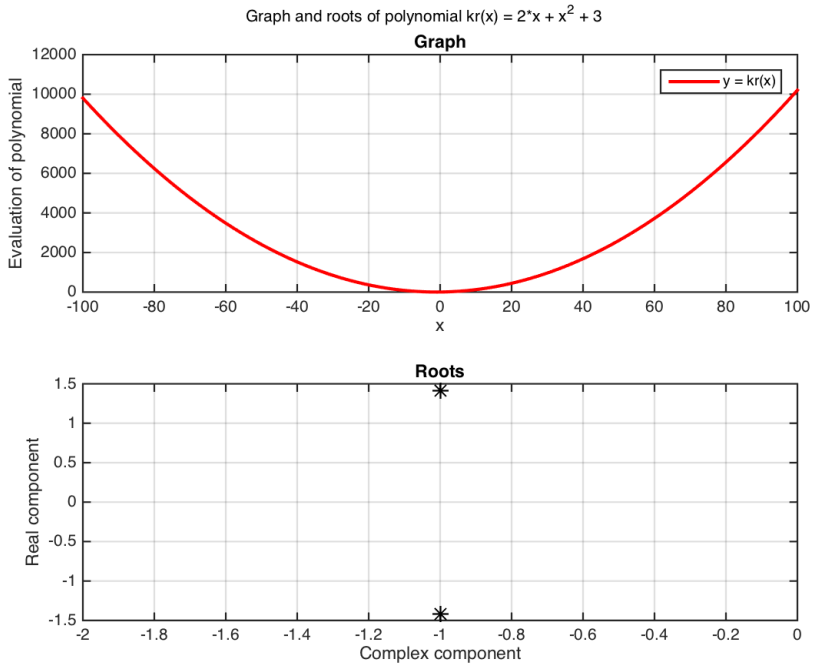

Figura 4

- **Gráfica y raíces de** $v(x)$ A través de la invocación: `plotpolroots(v, 'v', 'b', -100, 100)`

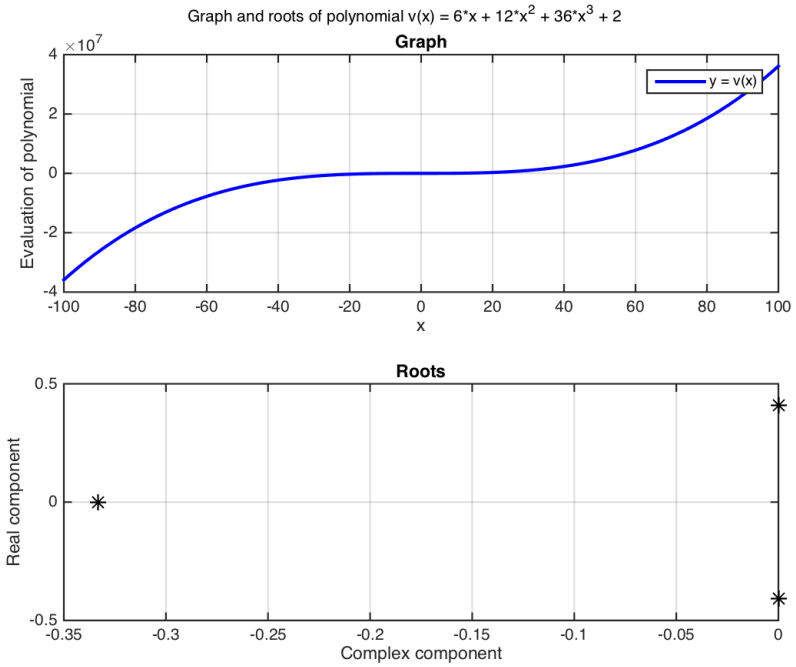

Figura 5

6. Por último, se hace uso de la función `factor`, para clasificar las raíces de cada uno de los polinomios. Para cada uno de los polinomios, se ejecuta la instrucción `factor(poly2sym(pol), x, 'FactorMode', 'complex')`, donde `pol`, es la variable que apunta a la representación del polinomio de cada caso:

   - **Raíces de** $f(x)$

     ```
     [ x + 1.0 - 1.414i, x + 1.0 + 1.414i]
     ```

   - **Raíces de** $g(x)$

     ```
     [ 9.0, x - 0.202 + 0.595i, x + 0.424 + 0.317, x - 0.202 - 0.595i, x
     + 0.424 - 0.317i]
     ```

   - **Raíces de** $h(x)$

     ```
     [ 9.0, x + 1.0 - 1.414i, x - 0.202 + 0.595i, x + 0.424 + 0.317i, x +
     1.0 + 1.414i, x - 0.202 - 0.595i, x + 0.424 - 0.317i]
     ```

   - **Raíces de** $kr(x)$

     ```
     [ x + 1.0 - 1.414i, x + 1.0 + 1.414i]
     ```

   - **Raíces de** $v(x)$

     ```
     [ 36.0, x + 0.333, x - 0.408i, x + 0.408i]
     ```

4 Conclusiones

El desarrollo de los ejercicios propuestos para el laboratorio, sirvieron para conceptualizar los algoritmos fundamentales para el tratamiento de una estructura algebraica fundamental tal y como son los polinomios. Entender el manejo de polinomios en MATLAB, es fundamental, pues muchos problemas relacionados con la solución numérica de ecuaciones diferenciales ordinarias y parciales, pueden reducirse a problemas relacionados con polinomios. Además, resulta práctico obtener las raíces de un polinomio de grado mayor a dos, pues con cierta tolerancia numérica, se obtienen los ceros de un polinomio. Esto resulta interesante, si se considera que para polinomios de grado mayor o igual a cinco, es una tara imposible hallar expresiones algebraicas para determinar las raíces del polinomio (Galois y Abel).

Es interesante ver que MATLAB, el cual obtiene su nombre gracias a la expresión *Matrix laboratory*, reduce los problemas de polinomios a problemas matriciales, pues como se documento, hallar las raíces de un polinomio, es tratado en la plataforma como encontrar los valores propios de una matriz cuyos elementos de la diagonal, son los elementos del vector que representa el polinomio. Aparte de esto, la representación vectorial de polinomios en MATLAB, sugiere también este hecho. De manera puntual, cabe resaltar a manera de conclusión lo siguiente:

- Los ejercicios cuyo objetivo fue el de familiarizar al usuario con la administración de polinomios en MATLAB, sirvieron para afianzar los conceptos fundamentales sobre las operaciones principales sobre polinomios. Resulto inquietante la conveniente representación interna de los polinomios en MATLAB, lo cual ayuda con la eficiencia y escritura de las operaciones antes mencionadas.

5 Bibliografía

[1] Jaluria, Y. *Computer Methods for Engineering with MATLAB Applications, Second Edition.* 2011. Series in Computational and Physical Processes in Mechanics and Thermal Sciences - Taylor & Francis. Pags. 60 - 63.

[2] Yang, W.Y. and Cao, W. and Chung, T.S. and Morris, J. *Applied Numerical Methods Using MATLAB.* 2005. Wiley. Pags. 50 - 57.

[3] Lopez, C. *MATLAB Symbolic Algebra and Calculus Tools.* 2014. MATLAB Solutions Series - Apress. Pags. 73 - 75.

[4] Siauw, T. and Bayen, A. *An Introduction to MATLAB Programming and Numerical Methods for Engineers* 2014. Elsevier Science. Pags. 198 - 200.