GRUMAI - GRUPO DE MATEMÁTICAS APLICADAS E INGENIERÍA

Métodos Numéricos Básicos para Ingeniería

Con implementaciones en MATLAB y Excel

Carlos Armando De Castro Payares

Notas sobre los métodos numéricos básicos más útiles para la Ingeniería, sin mucha profundización matemática y con aplicaciones prácticas y ejemplos de implementaciones en MATLAB y Excel.

CONTENIDOS

1. INTERPOLACIÓN
   1.1. Polinomios de Lagrange
   1.2. Interpolación lineal
2. APROXIMACIÓN
   2.1. Mínimos cuadrados
   2.2. Transformada rápida de Fourier
3. RAÍCES DE ECUACIONES
   3.1. Método de punto fijo
   3.2. Método de Newton-Raphson
   3.3. Método de la secante
4. SISTEMAS DE ECUACIONES NO LINEALES
   4.1. Método de Newton-Raphson multidimensional
5. DIFERENCIACIÓN NUMÉRICA
   5.1. Diferencias finitas
6. INTEGRACIÓN NUMÉRICA
   6.1. Método de los trapecios
7. ECUACIONES DIFERENCIALES CON VALOR INICIAL
   7.1. Método de Euler
   7.2. Método de Runge-Kutta de cuarto orden

PRÓLOGO

En el presente escrito se muestran los métodos numéricos más sencillos y útiles de implementar en problemas comunes de ingeniería. En los temas presentados no se hacen deducciones matemáticas complejas o profundas ni discusiones largas sobre los métodos sino que se muestra el método con alguna sencilla forma de ver el por qué funciona, se presenta el algoritmo en lenguaje MATLAB o en pseudocódigo, y luego se procede a ilustrar con ejemplos, algunos tomados de problemas de ingeniería, implementados en MATLAB o Excel.

Los métodos numéricos mostrados han sido utilizados por el autor en algún momento en el desarrollo de sus estudios en Ingeniería Mecánica, trabajo como consultor y proyectos personales.

La mayoría de los ejemplos hechos con Excel se anexan en el archivo Metodos Numericos Basicos para Ingenieria.xls, descargable de la página web del libro.

1. INTERPOLACIÓN

En la práctica de la ingeniería se utilizan mucho las tablas de datos, como en el caso de las tablas de vapor saturado en la termodinámica. En la mayoría de los casos el dato necesario no se encuentra explícito en la tabla sino entre dos valores de ésta, para lo cual es necesario estimarlo de entre los valores que presenta la tabla en un proceso conocido como *interpolación*.

La idea básica de la interpolación es hallar un polinomio o función que cumpla con pasar por todos los puntos de un conjunto de datos $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, y poder estimar los valores entre ellos por medio del polinomio.

1.1. POLINOMIOS DE LAGRANGE

Para ilustrar la interpolación por polinomios de Lagrange considérese un conjunto de datos de tres puntos $(x_1, y_1), (x_2, y_2), (x_3, y_3)$. El polinomio interpolador en este caso es

$$P(x) = \frac{(x - x_2)(x - x_3)}{(x_1 - x_2)(x_1 - x_3)} y_1 + \frac{(x - x_1)(x - x_3)}{(x_2 - x_1)(x_2 - x_3)} y_2 + \frac{(x - x_1)(x - x_2)}{(x_3 - x_1)(x_3 - x_2)} y_3$$

Obsérvese que en el punto $x = x_1$ sólo queda el primer término con su numerador y denominador cancelándose entre sí, por lo cual $P(x_1) = y_1$. Lo mismo sucede con los demás puntos, por lo que se ve que el polinomio cumple con la condición de pasar por todos los puntos de datos. En general, para *n* puntos de datos, el polinomio de Lagrange es

$$\boxed{P(x) = \sum_{i=1}^{n} y_i \prod_{\substack{j=1 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}} \qquad (1.1)$$

Una forma mucho más sencilla de ver la ec. 1.1 es en forma de un algoritmo, el cual se muestra escrito para MATLAB en el algoritmo 1.1.

Algoritmo 1.1: Polinomios de Lagrange en MATLAB
Entradas: valor a interpolar x, vectores conteniendo los puntos X y Y.
Salidas: valor interpolado y.

```
function [y]=PoliLagrange(x,X,Y)
y=0;
for i=1:numel(X)
    L=1;
    for j=1:numel(X)
        if j~=i
            L=L*(x-X(j))/(X(i)-X(j));
        end
    end
    y=y+L*Y(i);
end
```

A continuación se muestra un ejemplo para ilustrar la implementación del algoritmo 1.1.

Ejemplo 1.1.

Se tiene el conjunto de datos $\{(1,1),(2,3),(3,-1),(4,0),(5,3),(6,2)\}$. En MATLAB se introducen entonces como los vectores X=[1 2 3 4 5 6], Y=[1 3 -1 0 3 2]. Un ciclo implementando el algoritmo 1.1 muestra el polinomio interpolador de Lagrange

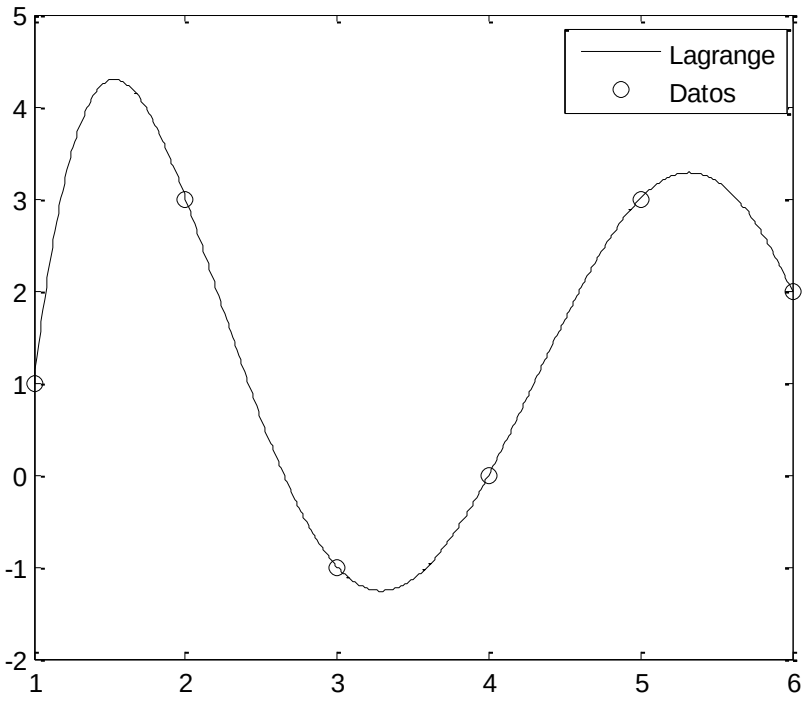

Figura 1.1. Polinomio de Lagrange interpolando los datos.

1.2. INTERPOLACIÓN LINEAL

La interpolación lineal es la más utilizada en el manejo de datos de tablas. Consiste en trazar un recta entre cada par de los puntos de datos, razón por la cual también es llamada interpolación por *trazadores lineales* o *splines de primer orden*. Considérese un conjunto de datos $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, entre dos puntos consecutivos del conjunto de datos se tiene una recta cuya pendiente es $m = (y_{i+1} - y_i)/(x_{i+1} - x_i)$ y que pasa por el punto inicial (x_i, y_i), entonces la ecuación de la recta que interpola entre ese par de puntos es

$$\boxed{y = \frac{y_{i+1} - y_i}{x_{i+1} - x_i}(x - x_i) + y_i} \qquad (1.2)$$

Hay que tener en cuenta que la interpolación lineal se hace por pedazos y no entrega un solo polinomio para todo el conjunto de datos como en el caso de los polinomios de Lagrange.

La implementación de la interpolación lineal en MATLAB teniendo en cuenta que es a pedazos se muestra en el algoritmo 1.2.

Algoritmo 1.2: Interpolación lineal en MATLAB
Entradas: valor a interpolar x, vectores conteniendo los puntos X y Y.
Salidas: valor interpolado y.

```
function [y]=IntLineal(x,X,Y)
for i=1:numel(X)-1
    if x>=X(i) && x<=X(i+1)
        y=(Y(i+1)-Y(i))/(X(i+1)-X(i))*(x-X(i))+Y(i);
    end
end
```

Ejemplo 1.2.

Utilizando los mismos valores del ejemplo 1.1 se tiene la implementación del algoritmo 1.2.

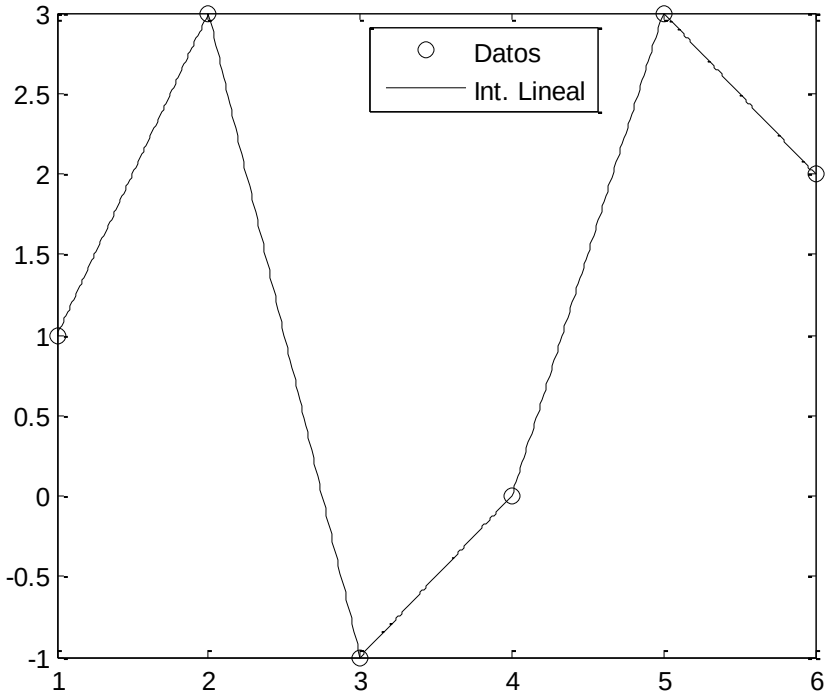

Figura 1.2. Interpolación lineal de los datos.

2. APROXIMACIÓN

En ocasiones se tiene un conjunto de datos experimentales y se desea hallar una función analítica que los represente de forma satisfactoria. Para ello es necesario hacer una aproximación de la función a los datos. En el presente capítulo se muestran dos métodos para hacerlo.

2.1. MÍNIMOS CUADRADOS

Los mínimos cuadrados es un método basado en minimizar el error entre los datos y la función de aproximación. Para un conjunto de datos $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$, si $y = f(x)$ es la función de aproximación, la suma del cuadrado de los errores es

$$R = \sum_{i=1}^{n} (f(x_i) - y_i)^2$$

Y debe minimizarse. En el caso de una recta, se tiene

$$R = \sum_{i=1}^{n} (mx_i + b - y_i)^2$$

Para minimizar el error debe cumplirse $\frac{\partial R}{\partial m} = 0, \frac{\partial R}{\partial b} = 0$, entonces

$$\frac{\partial R}{\partial m} = 2\sum_{i=1}^{n} (mx_i + b - y_i)x_i = 0$$

$$\frac{\partial R}{\partial b} = 2\sum_{i=1}^{n} (mx_i + b - y_i) = 0$$

De donde surge el sistema de ecuaciones lineal

$$m\sum_{i=1}^{n} x_i^2 + b\sum_{i=1}^{n} x_i = \sum_{i=1}^{n} x_i y_i$$

$$m\sum_{i=1}^{n} x_i + bn = \sum_{i=1}^{n} y_i$$

Algoritmo 2.1: Aproximación lineal por mínimos cuadrados en MATLAB

Entradas: vectores conteniendo los puntos X y Y.

Salidas: pendiente *m* e intercepto *b* de la recta de aproximación.

```
function [m,b]=mincuadlin(X,Y)
n=numel(X);
A(2,2)=n;
B=zeros(2,1);
for i=1:n
    A(1,1)=A(1,1)+X(i)^2;
    A(1,2)=A(1,2)+X(i);
    A(2,1)=A(2,1)+X(i);
    B(1,1)=B(1,1)+X(i)*Y(i);
    B(2,1)=B(2,1)+Y(i);
end
sol=A\B;
m=sol(1,1);
b=sol(2,1);
```

El algoritmo 2.1 puede ser optimizado, se deja al lector la optimización del mismo como ejercicio. El análisis hecho para la aproximación por medio de una recta puede hacerse de manera análoga para cualquier función *f*.

Ejemplo 2.1. Ensayo de tensión

Se tienen los siguientes datos de la parte elástica de un ensayo de tensión realizado con una probeta, y se desea conocer el módulo de elasticidad del material:

ε [mm/mm]	σ [MPa]
0	0
0.001	20.5
0.002	25.2
0.003	35.4
0.004	41.6
0.005	44.2
0.006	50.3

El módulo de elasticidad viene dado por la pendiente de la recta que aproxima los datos, entonces, aplicando el algoritmo 2.1 a los datos se tiene que la pendiente de la recta es E = 76.7 GPa, y el material que más se aproxima a tal módulo es una aleación de aluminio.

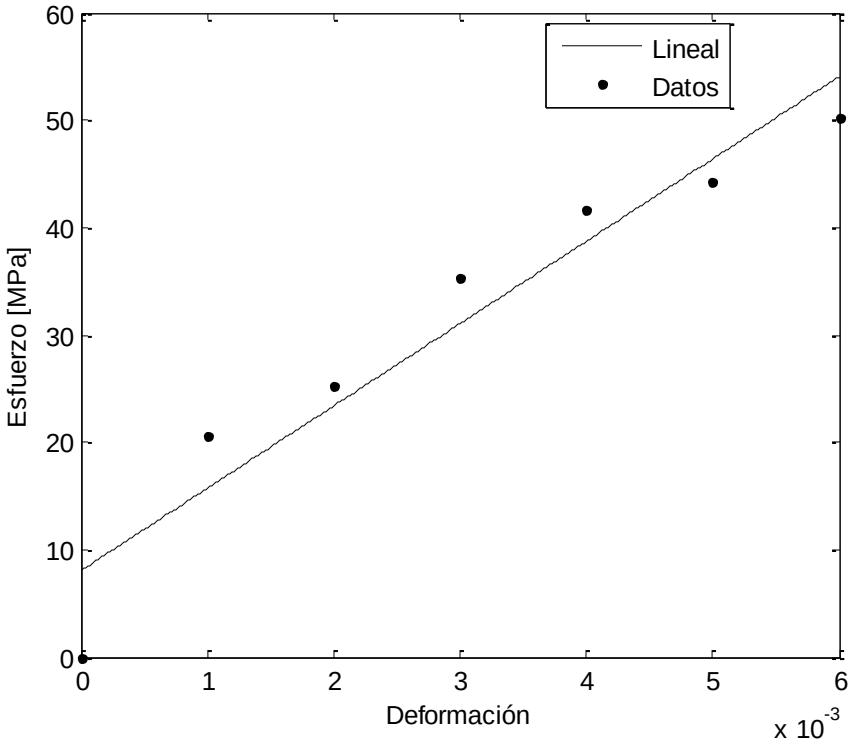

Figura 2.1. Aproximación lineal por mínimos cuadrados.

2.2. TRANSFORMADA RÁPIDA DE FOURIER

Las series de Fourier son útiles para representar cualquier onda como una sumatoria de senos y cosenos. En este caso se tratará únicamente con el manejo de datos experimentales por medio de la transformada rápida de Fourier.

Cuando se tiene una serie de datos en el tiempo $(t_1, y_1), (t_2, y_2), \dots, (t_n, y_n)$, con una frecuencia de muestreo (es decir, inverso del intervalo de tiempo entre datos medidos) f_s y un número de datos N par, se define una resolución en frecuencia $\Delta f = f_s/N$ y la función que aproxima los datos es

$$\boxed{f(t) = \sum_{n=0}^{N/2} [A_n \cos(2\pi n \Delta f t) + B_n \sin(2\pi n \Delta f t)]} \qquad (2.1)$$

Los coeficientes A_n y $B_n$se calculan de la siguiente manera

$$A_n = \frac{2}{N} \sum_{r=1}^{N} y_r \cos\left(\frac{2\pi r n}{N}\right)$$

$$B_n = \frac{2}{N}\sum_{r=1}^{N} y_r \sin\left(\frac{2\pi r n}{N}\right)$$

Una aplicación útil de la transformada rápida de Fourier es el análisis de espectros de frecuencias, que son gráficas de la amplitud total de la onda en función del armónico *n* o la frecuencia. La amplitud de la onda es

$$C_n = \sqrt{A_n^2 + B_n^2}$$

En el algoritmo 2.2 se saca el espectro de frecuencia de una onda.

Algoritmo 2.2: Espectro de frecuencias de una onda en MATLAB
Entradas: vector de valores de la onda x.
Salidas: frecuencias f y amplitud C para la gráfica del espectro de frecuencias.

```
function [f,C]=TFourier(x)
%Carlos Armando De Castro P.

N=numel(x);

%Coeficientes armónicos.

for n=0:N/2;
   A(n+1)=0;
   B(n+1)=0;

   for r=1:N;
        A(n+1)=A(n+1)+2/N*x(r)*cos(2*pi*r*n/N);
        B(n+1)=B(n+1)+2/N*x(r)*sin(2*pi*r*n/N);
   end
   C(n+1)=sqrt((A(n+1))^2+(B(n+1))^2);
   f(n+1)=r*n/N;
end
```

Algoritmo 2.3: Análisis de Fourier de una grabación en MATLAB
Entradas: grabación hecha con el computador.
Salidas: espectro de frecuencias y onda reconstruida por Fourier.

```
function [time, C, y]=voz_fourier
%Carlos Armando De Castro P.

%Toma de datos.

fs=input('Frecuencia de muestreo [Hz] fs  = ');
N=input('Número de datos (par) N = ');

sw=0;
while sw==0;
    sw=input('Oprima cualquier número diferente de cero y ENTER para
grabar: ');
    voz=wavrecord(N,fs);
end
```

```
Df=fs/(N);
time=1/fs:1/fs:N/fs;
X(:,2)=voz;
X(:,1)=time;

%Coeficientes armónicos.

for n=0:N/2;
   A(n+1)=0;
   B(n+1)=0;

   for r=1:N;
        A(n+1)=A(n+1)+2/N*X(r,2)*cos(2*pi*r*n/N);
        B(n+1)=B(n+1)+2/N*X(r,2)*sin(2*pi*r*n/N);
   end

   C(n+1)=sqrt((A(n+1))^2+(B(n+1))^2);

end

C=C';

%Función y(t)

for r=1:N;

    y(r)=A(1)/2+A(N/2+1)/2*cos(2*pi*N/2*Df*X(r,1));

     for n=0:N/2;

y(r)=y(r)+A(n+1)*cos(2*pi*n*Df*X(r,1))+B(n+1)*sin(2*pi*n*Df*X(r,1));
     end

end

y=y';
```

Ejemplo 2.2: Análisis de la voz humana

Considérese por ejemplo una grabación de una palabra, en la figura 2.2 se muestra la gráfica de una grabación digitalizada tomada y analizada con el algoritmo 2.3. En la figura 2.3 se muestra el espectro de frecuencias de la grabación.

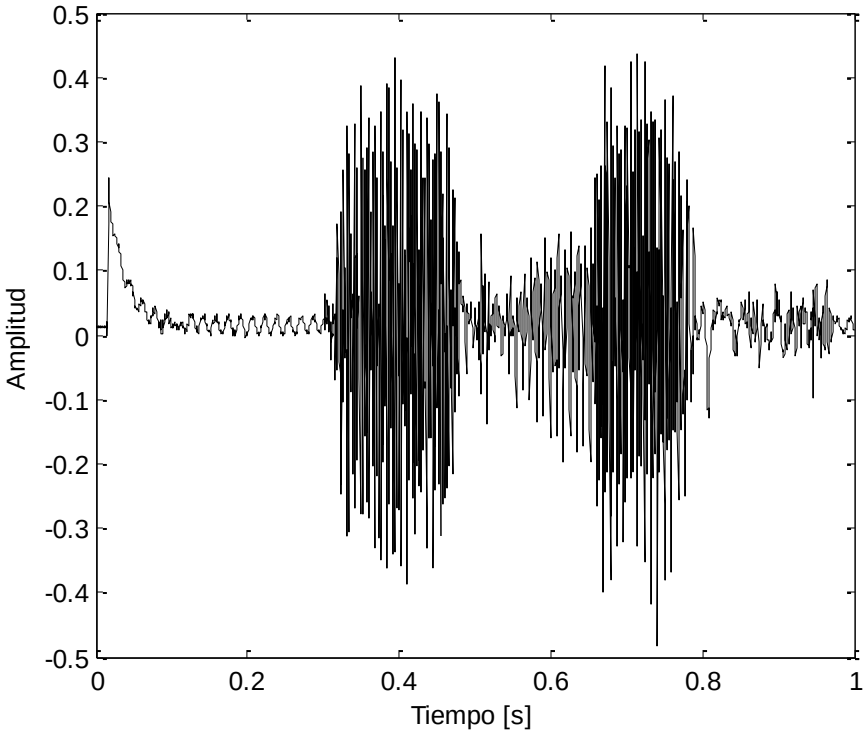

Figura 2.2. Grabación de una palabra.

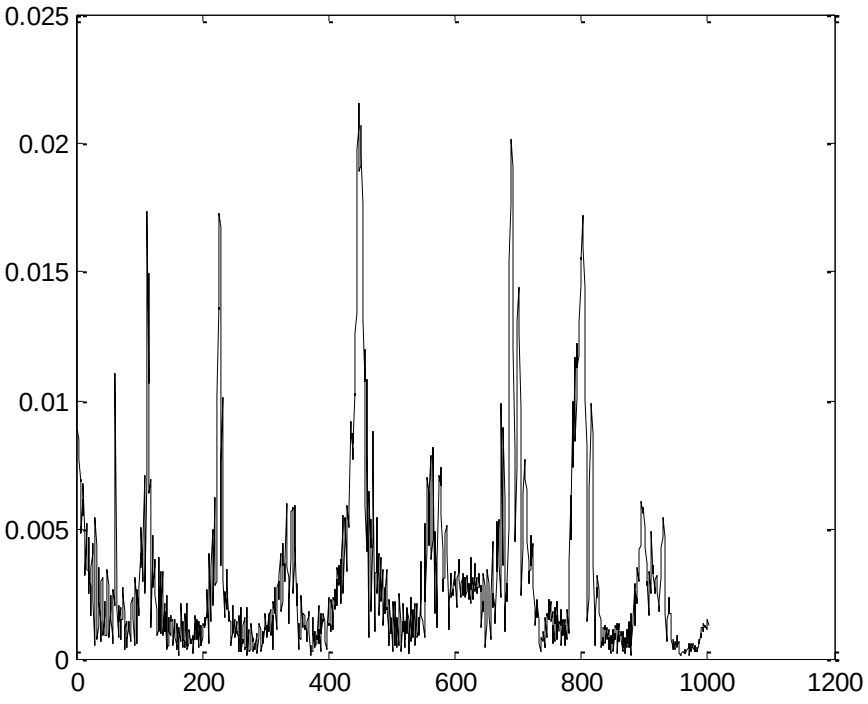

Figura 2.3. Espectro de frecuencias de la grabación.

3. RAÍCES DE ECUACIONES

En ocasiones en el ámbito de la ingeniería es necesario resolver ecuaciones no lineales que no tienen solución analítica o que es muy complicado hallarlas, como en el caso de la fórmula de la secante para pandeo de columnas con carga excéntrica. Para estos casos, deben utilizarse métodos de solución numérica de ecuaciones.

3.1. MÉTODO DE PUNTO FIJO

El método de punto fijo consiste en una forma iterativa de resolver una ecuación de la forma $f(x) = x$. El método consiste en elegir una aproximación inicial x_0 y realizar la iteración

$$\boxed{x_{k+1} = f(x_k)} \quad (3.1)$$

Hasta que la diferencia $|x_{k+1} - x_k|$ sea muy cercana a cero, para lo cual se establece una tolerancia a criterio del usuario. En el algoritmo 3.1 se muestra el algoritmo de punto fijo en pseudocódigo parecido a MATLAB, debido a que no hay forma de escribir un código en MATLAB general para este método.

Algoritmo 3.1: Método de punto fijo (pseudocódigo)
Entradas: aproximación inicial x0, tolerancia TOL
Salidas: valor x tal que f(x)=x

```
sw=1;
x1=x0;
while sw==1
    x2=f(x1);
    if abs(x2-x1)<=TOL
        x=x2;
        sw=0;
    end
    x1=x2;
end
```

Ejemplo 3.1. Mecánica de la fractura

La ecuación de factor de intensidad de esfuerzos para una placa de ancho w y espesor t con una grieta en el borde de largo a es

$$K = \sigma Y \sqrt{a} \quad (E3.1)$$

Donde Y es un factor geométrico que depende del ancho de la placa y el tamaño de grieta, siendo

$$Y = \left[1.99 - 0.41\left(\frac{a}{w}\right) + 18.70\left(\frac{a}{w}\right)^2 - 38.48\left(\frac{a}{w}\right)^3 + 53.85\left(\frac{a}{w}\right)^4\right] \quad (E3.2)$$

La falla catastrófica de la placa se produce cuando el factor de intensidad de esfuerzos K iguala o supera a la tenacidad a la fractura K_{Ic}, entonces el tamaño de grieta crítica es

$$a_f = (K_{IC}/\sigma Y)^2 \quad (E3.3)$$

Como el factor geométrico *Y* depende de a_f, la ecuación E3.3 debe resolverse por el método de punto fijo. La iteración es entonces:

$$a_{k+1} = \left(\frac{K_{IC}}{\sigma\left[1.99 - 0.41\left(\frac{a_k}{w}\right) + 18.70\left(\frac{a_k}{w}\right)^2 - 38.48\left(\frac{a_k}{w}\right)^3 + 53.85\left(\frac{a_k}{w}\right)^4\right]}\right)^2$$

$$(E3.4)$$

Se tiene un caso de una placa sujeta a tensión donde w = 2.5in, σ = 24.89ksi, K_{Ic} = 52ksi√in. Se elige como aproximación inicial a_0 = 0.250in. Una tabla de Excel programada para este caso particular con el método de punto fijo entrega la solución con tres cifras decimales:

a(i)	Y	a (i+1)	a (i+1) - a (i)	Iteración
0.250	2.103	0.987	0.737	1
0.987	3.684	0.322	-0.665	2
0.322	2.180	0.919	0.597	3
↓	↓	↓	↓	↓
0.620	2.655	0.619	-0.001	118
0.619	2.654	0.620	0.001	119
0.620	2.655	0.620	0.000	120

La tabla muestra que la iteración converge en 120 pasos al valor $a_f = 0.620$in.

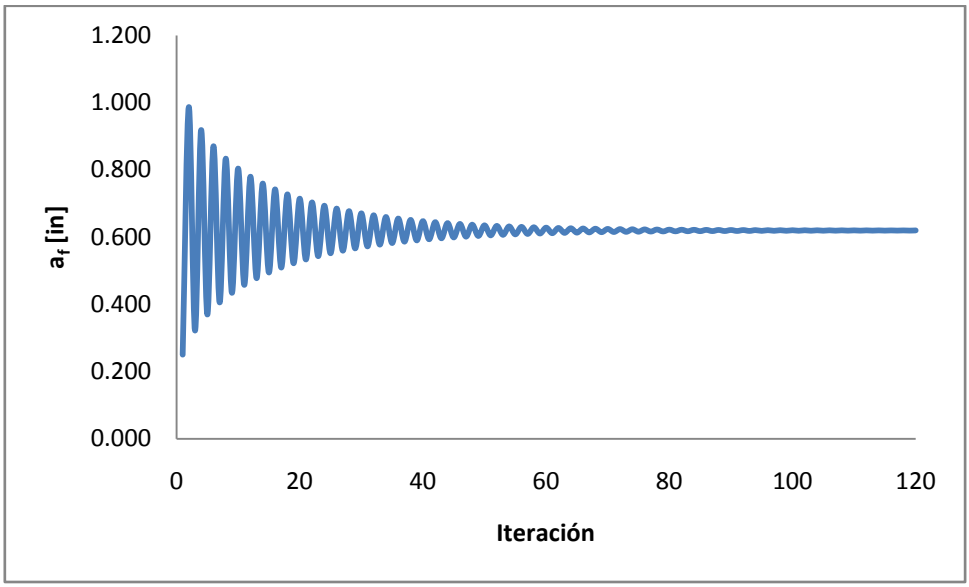

Figura 3.1. Valor calculado de a_f en cada iteración.

3.2. MÉTODO DE NEWTON-RAPHSON

Se utiliza para resolver ecuaciones de la forma $f(x) = 0$. El método consiste en elegir una aproximación inicial x_0 y realizar la iteración

$$\boxed{x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}} \qquad (3.2)$$

Hasta que la diferencia $|x_{k+1} - x_k|$ sea muy cercana a cero, para lo cual se establece una tolerancia a criterio del usuario. En el algoritmo 3.2 se muestra el algoritmo de Newton-Raphson en pseudocódigo parecido a MATLAB, debido a que no hay forma de escribir un código en MATLAB general para este método.

Algoritmo 3.2: Método de Newton-Raphson (pseudocódigo)
Entradas: aproximación inicial x0, tolerancia TOL
Salidas: valor x tal que f(x)=0

```
sw=1;
x1=x0;
while sw==1
    x2=x1-f(x1)/f'(x1);
    if abs(x2-x1)<=TOL
        x=x2;
        sw=0;
    end
    x1=x2;
end
```

Ejemplo 3.2.

Se desea resolver la ecuación $3x^3 - 2x + 8 = 0$ con tres cifras significativas, entonces se tiene $f(x) = 3x^3 - 2x + 8$ y $f'(x) = 9x^2 - 2$. La iteración es entonces

$$x_{k+1} = x_k - \frac{3x_k^3 - 2x_k + 8}{9x_k^2 - 2}$$

Se escoge una aproximación inicial x_0 = -10, entonces, una sencilla tabulación en Excel da la raíz $x = -1.546$

x(i)	x(i+1)	x(i+1)-x(i)
-10.000	-6.690	3.310
-6.690	-4.502	2.188
-4.502	-3.079	1.423
-3.079	-2.198	0.881
-2.198	-1.729	0.469
-1.729	-1.566	0.162
-1.566	-1.546	0.020
-1.546	-1.546	0.000

3.3. MÉTODO DE LA SECANTE

Se utiliza para resolver ecuaciones de la forma $f(x) = 0$. El método consiste en elegir dos aproximaciones iniciales x_0, x_1 y realizar la iteración

$$x_{k+1} = x_k - (x_k - x_{k-1})\frac{f(x_k)}{f(x_k) - f(x_{k-1})} \qquad (3.3)$$

Hasta que la diferencia $|x_{k+1} - x_k|$ sea muy cercana a cero, para lo cual se establece una tolerancia a criterio del usuario. El método de la secante tiene la ventaja de que no se debe derivar la ecuación. En el algoritmo 3.3 se muestra el algoritmo del método de la secante en pseudocódigo parecido a MATLAB, debido a que no hay forma de escribir un código en MATLAB general para este método.

Algoritmo 3.3: Método de la secante (pseudocódigo)
Entradas: aproximaciones iniciales x0, x1, tolerancia TOL
Salidas: valor x tal que f(x)=0

```
sw=1;
while sw==1
    x2=x1-(x1-x0)*f(x1)/(f(x1)-f(x0));
    if abs(x2-x1)<=TOL
        x=x2;
        sw=0;
    end
end
```

Ejemplo 3.3.

Se necesita resolver la ecuación trigonométrica $\sin(x) + 3\cos(x) = 0$ entre $0 \leq x \leq 10$ con tres cifras significativas. La iteración en este caso es entonces

$$x_{k+1} = x_k - (x_k - x_{k-1}) \frac{\sin(x_k) + 3\cos(x_k)}{\sin(x_k) + 3\cos(x_k) - [\sin(x_{k-1}) + 3\cos(x_{k-1})]}$$

Con una tabla de Excel programada con el método de la secante se hallan las tres soluciones $x = 1.893, x = 5.034, x = 8.176$.

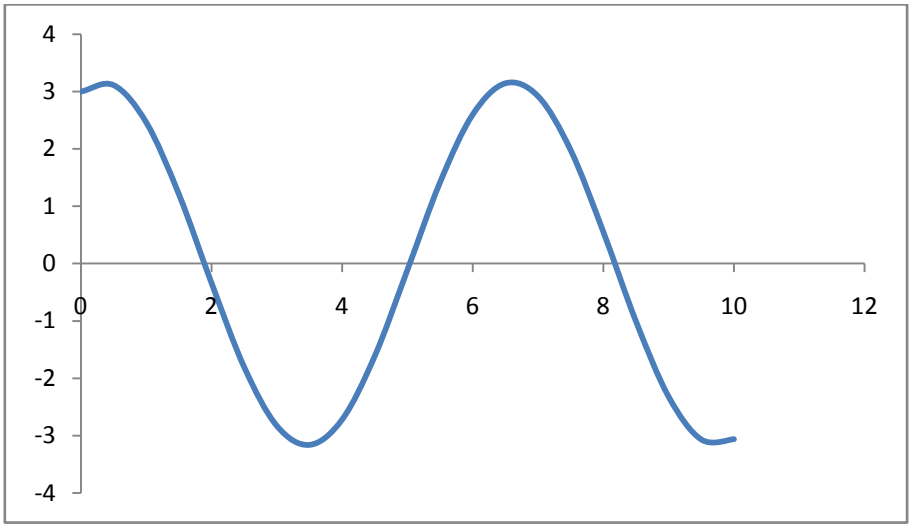

Figura 3.2. Gráfica de la ecuación del ejemplo 3.3, donde se observan las tres raíces de la ecuación.

4. SISTEMAS DE ECUACIONES NO LINEALES

En ocasiones surgen sistemas de ecuaciones no lineales que deben resolverse. Para ello existen métodos iterativos, de los cuales se presenta uno muy útil.

4.1. MÉTODO DE NEWTON-RAPHSON MULTIDIMENSIONAL

El método de Newton-Raphson multidimensional sirve para resolver un sistema de ecuaciones de la forma

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= 0 \\ f_2(x_1, x_2, \dots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \dots, x_n) &= 0 \end{aligned}$$

El método procede definiendo dos vectores $\mathbf{x}$ y $\mathbf{F}(\mathbf{x})$, tal que

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}, \quad \mathbf{F}(\mathbf{x}) = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{bmatrix}$$

Luego procede a hallarse el jacobiano del sistema, siendo éste definido como

$$J_{i,j} = \frac{\partial f_i}{\partial x_j} \qquad (4.1)$$

Debe hacerse una aproximación inicial de la solución $\mathbf{x}_0$, y luego se hace la iteración

$$\boxed{\mathbf{x}_{i+1} = \mathbf{x}_i - J(\mathbf{x}_i)^{-1}\mathbf{F}(\mathbf{x}_i)} \qquad (4.2)$$

Hasta que el valor $\|\mathbf{x}_{i+1} - \mathbf{x}_i\|$ sea muy cercano a cero, para lo cual se define una tolerancia a criterio del usuario. La iteración 5.2 es recomendable hacerla en dos pasos, primero resolviendo el sistema lineal

$$J(\mathbf{x}_i)\mathbf{y} = \mathbf{F}(\mathbf{x}_i)$$

Y luego realizando la iteración

$$\mathbf{x}_{i+1} = \mathbf{x}_i - \mathbf{y}$$

5. DIFERENCIACIÓN NUMÉRICA

La diferenciación numérica es muy útil en casos en los cuales se tiene una función que es muy engorrosa de derivar, o en casos en los cuales no se tiene una función explícita sino una serie de datos experimentales.

5.1. DIFERENCIAS FINITAS

Para entender de una manera sencilla la discretización por diferencias finitas de una derivada debe tenerse en cuenta la interpretación geométrica de la derivada en un punto, que es la pendiente de la curva en el punto de interés. Considérense tres puntos intermedios en una curva como se muestra en la figura 5.1:

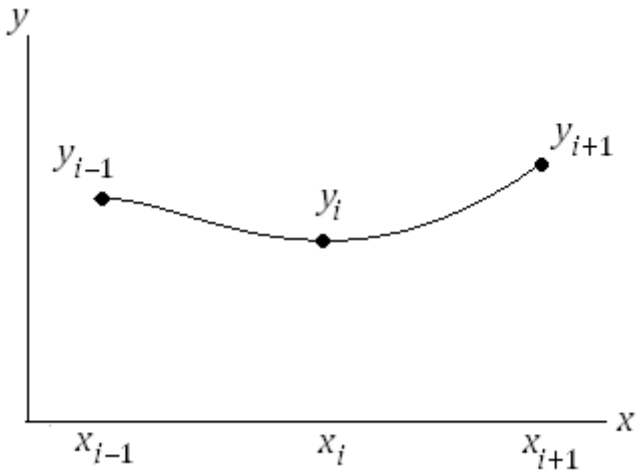

Figura 5.1. Curva discretizada.

Supóngase que interesa la derivada en el punto (x_i, y_i), tres formas de aproximar la pendiente por recta en ese punto son:

$$\left.\frac{dy}{dx}\right|_{x=x_i} \approx \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \qquad (5.1)$$

$$\left.\frac{dy}{dx}\right|_{x=x_i} \approx \frac{y_i - y_{i-1}}{x_i - x_{i-1}} \qquad (5.2)$$

$$\left.\frac{dy}{dx}\right|_{x=x_i} \approx \frac{y_{i+1} - y_{i-1}}{x_{i+1} - x_{i-1}} \qquad (5.3)$$

Las ecuaciones 5.1 a 5.3 son llamadas *diferencias finitas*. La ecuación 5.1 se recomienda para hallar la derivada del punto inicial de una curva, la ecuación 5.2 se recomienda para hallar la derivada del punto final de una curva, y la ecuación 5.3 es la ecuación de *diferencias finitas centrales*, y se recomienda para hallar la derivada en los puntos intermedios de una curva.

En el caso cuando las diferencias $x_i - x_{i-1} = x_{i+1} - x_i = \Delta x$ son constantes para todo el dominio, las ecuaciones de diferencias finitas quedan

$$\left.\frac{dy}{dx}\right|_{x=x_i} \approx \frac{y_{i+1} - y_i}{\Delta x} \qquad (5.4)$$

$$\left.\frac{dy}{dx}\right|_{x=x_i} \approx \frac{y_i - y_{i-1}}{\Delta x} \qquad (5.5)$$

$$\left.\frac{dy}{dx}\right|_{x=x_i} \approx \frac{y_{i+1} - y_{i-1}}{2\Delta x} \qquad (5.6)$$

La ecuación 5.4 se recomienda para hallar la derivada del punto inicial de una curva, la ecuación 5.5 se recomienda para hallar la derivada del punto final de una curva, y la ecuación 5.6 se recomienda para hallar la derivada en los puntos intermedios de una curva.

El método de derivación por diferencias finitas implementado en MATLAB se muestra en el algoritmo 5.1.

Algoritmo 5.1: Derivación numérica en MATLAB
Entradas: vectores conteniendo los puntos X y Y.
Salidas: vector con el valor de las derivadas, df.

```
function [df]=derivada(X,Y)
N=numel(X);
df(1)=(Y(2)-Y(1))/(X(2)-X(1));
df(N)=(Y(N)-Y(N-1))/(X(N)-X(N-1));
for n=2:N-1
    df(n)=(Y(n+1)-Y(n-1))/(X(n+1)-X(n-1));
end
plot(X,df,'k-')
```

La derivación numérica también puede implementarse de forma muy sencilla en tablas de Excel. Ver el archivo de Excel incluido en la página web de donde se descarga el libro para ver el ejemplo 5.1 resuelto con Excel.

Ejemplo 5.1. Curvas SVAJ para levas

Se tienen los siguientes datos de la curva de movimiento de un seguidor de leva. Determinar si la leva cumple con la continuidad de las curvas de velocidad, aceleración y sobre-aceleración (también conocida como *jerk*).

t [s]	x [m]
0	0.00
0.1	0.08
0.2	0.17
0.3	0.26
0.4	0.35
0.5	0.45
0.6	0.54
0.7	0.65
0.8	0.76
0.9	0.87
1	0.98
1.1	1.09
1.2	1.20
1.3	1.30
1.4	1.39
1.5	1.46
1.6	1.51
1.7	1.53
1.8	1.50
1.9	1.41
2	1.26
2.1	1.02
2.2	0.69
2.3	0.27
2.4	0.24
2.5	0.00

Se procede a derivar numéricamente la curva con los datos de la tabla para hallar la velocidad, la aceleración y la sobre-aceleración del seguidor de la leva. La velocidad, la aceleración y la sobre-aceleración del seguidor de hallan con el algoritmo 5.1 en MATLAB de la forma

```
>> v=derivada(t,x);

>> a=derivada(t,v);

>> j=derivada(t,a);
```

Los resultados que se muestran en las figuras 5.2 a 5.5 son las llamadas gráficas SVAJ.

Se observa que en una parte del ciclo hay un cambio abrupto de velocidad, lo que se confirma al observar el pico discontinúo en la curva de aceleración, lo cual puede traer problemas de vibración, golpes y esfuerzo en el sistema leva-seguidor analizado.

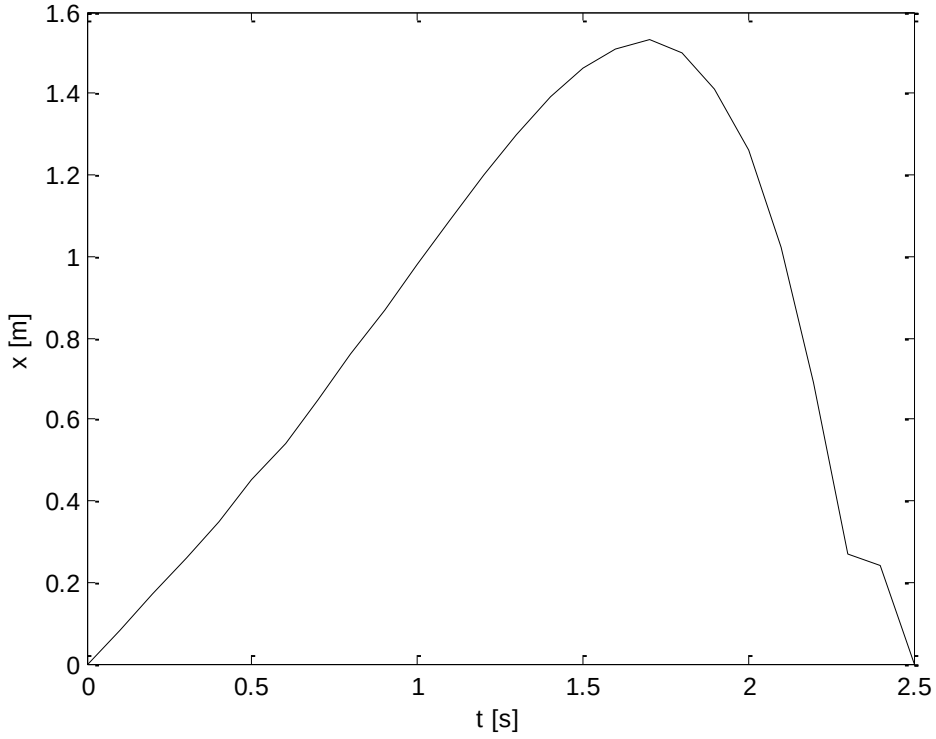

Figura 5.2. Posición del seguidor.

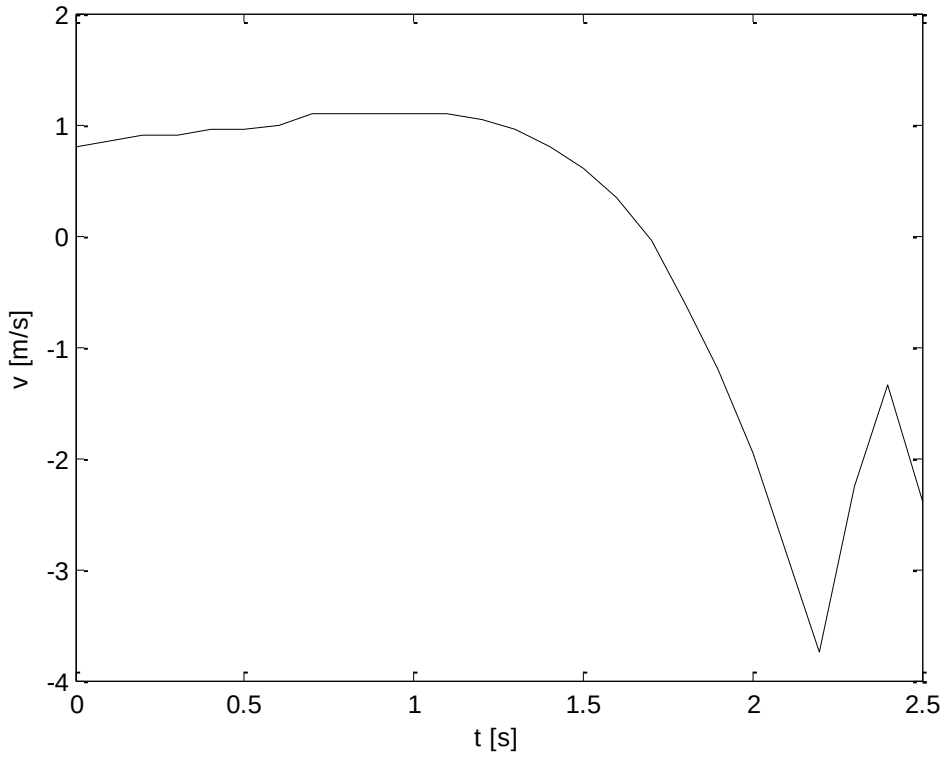

Figura 5.3. Velocidad del seguidor hallada por derivación numérica.

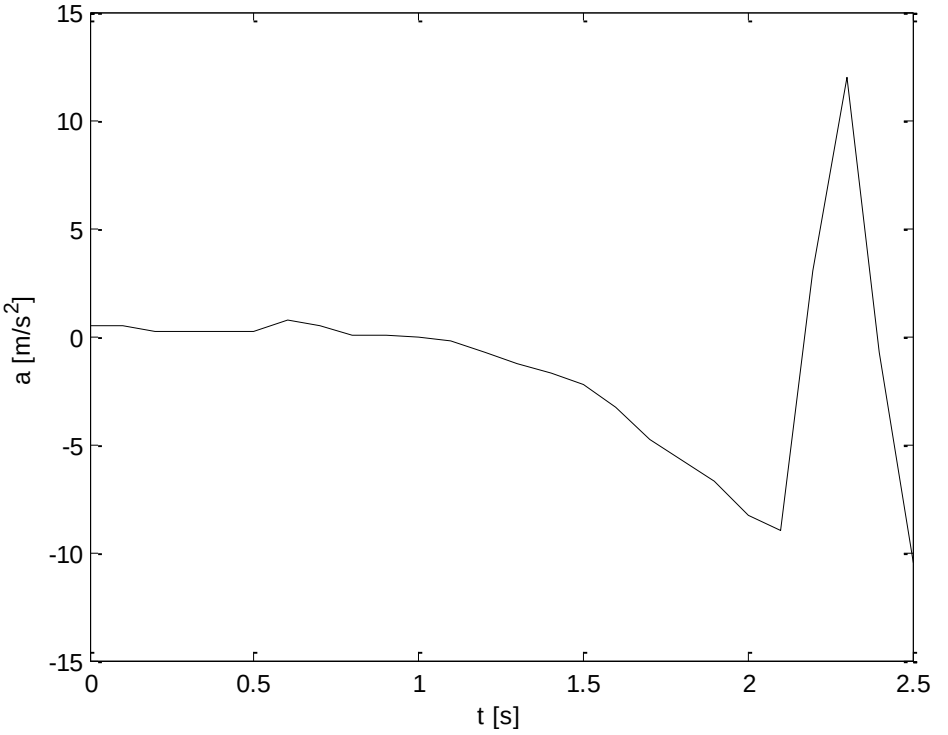

Figura 5.4. Aceleración del seguidor hallada por derivación numérica.

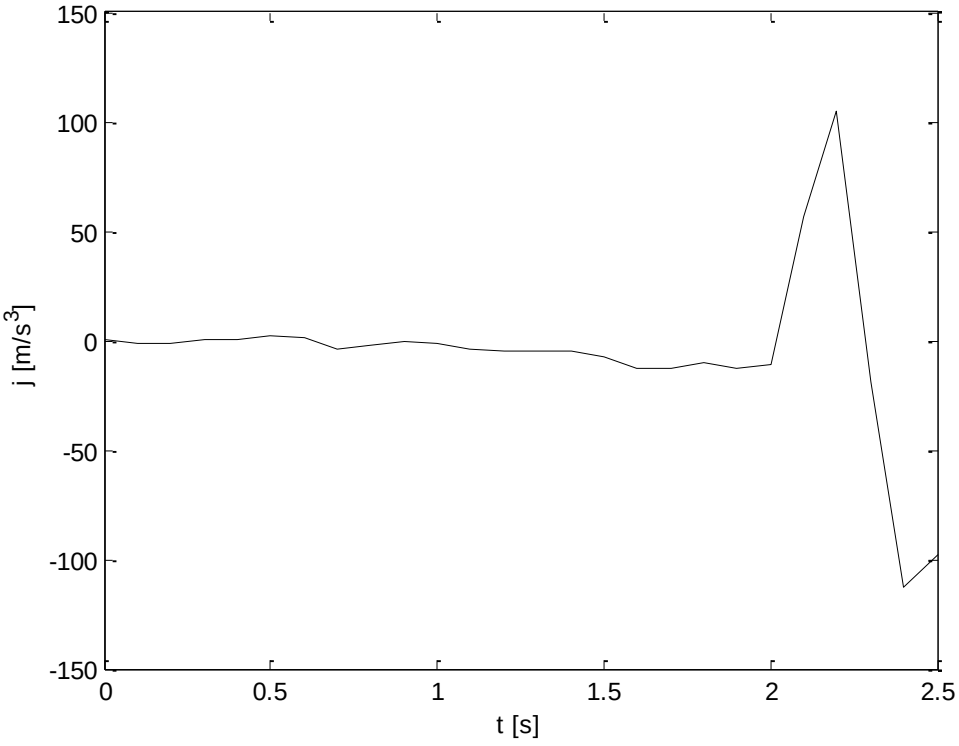

Figura 5.5. Sobre-aceleración del seguidor hallada por derivación numérica.

6. INTEGRACIÓN NUMÉRICA

La integración numérica es muy útil en casos en los cuales se tiene una función que es muy engorrosa de integrar o que no posee anti-derivada, o en casos en los cuales no se tiene una función explícita sino una serie de datos experimentales. Aunque hay varios métodos de integración numérica, acá solo se mostrará en método de los trapecios, ya que es el más sencillo de implementar y de entender.

6.1. MÉTODO DE LOS TRAPECIOS

Para entender el método de los trapecios debe tomarse en cuenta la interpretación geométrica de una integral como área bajo la curva, siendo así, considérese el área de un trapecio entre dos puntos de una curva como se muestra en la figura 6.1.

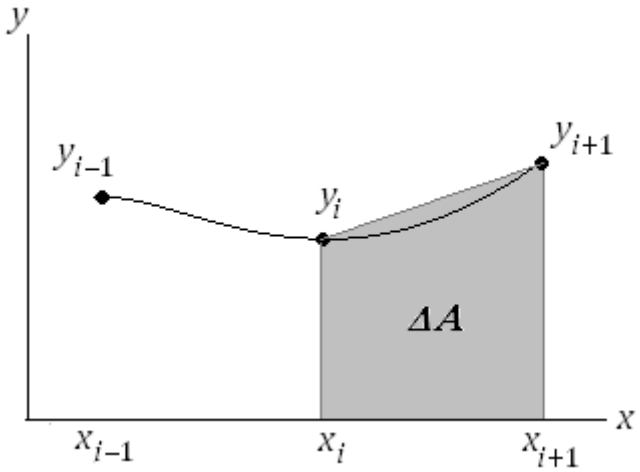

Figura 6.1. Área de un trapecio entre dos puntos de una curva.

El área del trapecio es

$$\Delta A_{i+1} = \frac{1}{2}(x_{i+1} - x_i)(y_{i+1} + y_i) \qquad (6.1)$$

El valor de la integral en un intervalo con n+1 puntos x_0 a x_n es entonces la suma de las distintas áreas por sub-intervalo:

$$I = \int_{x_0}^{x_n} y(x)dx \approx \sum_{i=0}^{n-1} \Delta A_i \qquad (6.2)$$

En el caso que el tamaño de sub-intervalo sea un valor Δx constante, la ecuación 6.2 resulta

$$I = \int_{x_0}^{x_n} y(x)dx \approx \frac{1}{2}\Delta x(y_0 + y_n) + \Delta x \sum_{i=1}^{n-1} y_i \qquad (6.3)$$

Otra forma de verlo, y más fácil de programar en una hoja de Excel, es la siguiente: el valor acumulado de la integral en el intervalo *i* (notado como I_i) es

$$\boxed{I_i = I_{i-1} + \frac{1}{2}(x_i - x_{i-1})(y_i + y_{i-1})} \qquad (6.4)$$

Y el valor *I* de la integral en el dominio de interés es el valor final acumulado de la ecuación 6.4. La ecuación 6.4 puede ponerse fácilmente en términos de fórmula de celdas en una hoja de Excel, descargar el archivo de Excel de la página donde se descarga el libro para ver el ejemplo 6.1 resuelto en Excel.

La implementación en MATLAB del método de los trapecios con la ecuación 6.4 se muestra en el algoritmo 6.1.

Algoritmo 6.1: Método de los trapecios en MATLAB
Entradas: valor inicial de la integral I0, vectores conteniendo los puntos X y Y.
Salidas: vector con el valor acumulativo de la integral, I.

```
function [I]=integral(I0,X,Y)
N=numel(X);
I(1)=I0;
for n=2:N
    I(n)=I(n-1)+0.5*(Y(n)+Y(n-1))*(X(n)-X(n-1));
end
plot(X,I,'k-');
```

Ejemplo 6.1. Mecánica de la fractura

El crecimiento de una grieta en el borde de una placa por ciclo de esfuerzos viene dado por la ecuación de Paris

$$\frac{da}{dN} = A\left(\Delta\sigma Y\sqrt{a}\right)^m \qquad (E6.1)$$

Donde *N* es el número de ciclos, *A* y *m* son constantes del material, $\Delta\sigma$ es la diferencia de esfuerzos a tensión sobre la pieza y *Y* viene dado por la ecuación E3.2. Cuando se observa una grieta de tamaño a_0, el número de ciclos restante para fractura catastrófica de la pieza se obtiene separando las variables e integrando la ecuación E6.1:

$$N_f = \int_{a_0}^{a_f} \frac{da}{A\left(Y\Delta\sigma\sqrt{a}\right)^m} \qquad (E6.2)$$

Supóngase que se tiene la placa del ejemplo 3.1 con $\Delta\sigma = 17.78\text{ksi}, A = 6.6 \times 10^{-9}, m = 2.25$, con una grieta inicial de $a_0 = 0.25\text{in}$. El número de ciclos restante para falla es

$$N_f = \int_{0.25}^{0.62} \frac{da}{6.6 \times 10^{-9}\left(17.78\left[1.99 - 0.41\left(\frac{a}{2.5}\right) + 18.70\left(\frac{a}{2.5}\right)^2 - 38.48\left(\frac{a}{2.5}\right)^3 + 53.85\left(\frac{a}{2.5}\right)^4\right]\sqrt{a}\right)^{2.25}}$$

En este caso, la solución en MATLAB con el algoritmo 6.1 se implementa de la manera siguiente:

```
>> a=0.25:0.01:0.62;

>> for i=1:numel(a)

f(i)=1/(6.6e-9*(17.78*(1.99-0.41*(a(i)/2.5)+18.70*(a(i)/2.5)^2-
38.48*(a(i)/2.5)^3+53.85*(a(i)/2.5)^4)*sqrt(a(i)))^2.25);

end

>> I0=0;

>> [N]=integral(I0,a,f);
```

El número de ciclos hasta la falla así calculado es N_f = 37109 ciclos. La gráfica de crecimiento de la grieta con los ciclos se muestra en la figura 6.2.

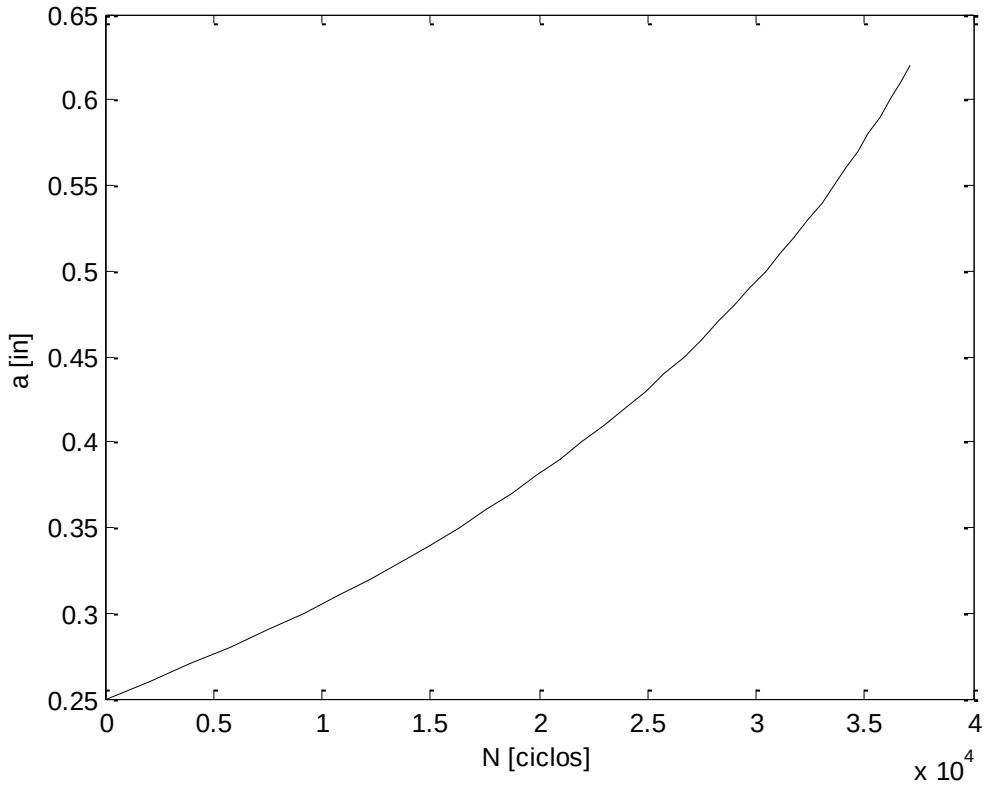

Figura 6.2. Crecimiento de la grieta con el número de ciclos hasta la falla.

7. ECUACIONES DIFERENCIALES CON VALOR INICIAL

Los métodos numéricos para ecuaciones diferenciales que se presentan aplican para ecuaciones diferenciales ordinarias con condiciones iniciales de tipo

$$\frac{dy}{dt} = f(t, y)$$
$$y(t_0) = y_0 \qquad (7.1)$$

Estos métodos son muy útiles cuando se tienen ecuaciones diferenciales que no pueden resolverse por los métodos analíticos o cuya solución analítica es muy engorrosa.

7.1. MÉTODO DE EULER

El método de Euler consiste en aproximar la derivada de la ecuación 7.1 por diferencias finitas como en la ecuación 5.4, entonces la ecuación diferencial resulta

$$\frac{y_{n+1} - y_n}{\Delta t} = f(t_n, y_n)$$

Por lo cual el valor de la función en intervalo de tiempo n+1 es

$$y_{n+1} = y_n + \Delta t \cdot f(t_n, y_n) \qquad (7.2)$$

El método de Euler tiene la desventaja de que se vuelve inestable y la solución diverge si el tamaño de paso de tiempo Δt es muy grande.

En el algoritmo 7.1 se muestra el algoritmo del método de Euler en pseudocódigo parecido a MATLAB, debido a que no hay forma de escribir un código en MATLAB general para este método.

Algoritmo 7.1: Método de Euler (pseudocódigo)
Entradas: valor inicial y0, tiempo inicial t0, tamaño de paso dt, número de puntos N
Salidas: valores y tal que dy/dt = f(t,y)

```
y(1)=y0;
t(1)=t0;
for n=2:N
    y(n)=y(n-1)+dt*f(t(n-1),y(n-1));
    t(n)=t(n-1)+dt;
end
```

Ejemplo 7.1. Mecánica de la fractura (otra vez…)

El crecimiento de una grieta en el borde de una placa por ciclo de esfuerzos viene dado por la ecuación de Paris (E6.1). Una forma de estimar el crecimiento de una grieta con el número de ciclos diferente a la integración numérica del ejemplo 6.1 es resolviendo la ecuación E6.1 por el método de Euler, en este caso se tiene la ecuación diferencial discretizada

$$\frac{a_{n+1}-a_n}{\Delta N} = A(\Delta\sigma)^m a_k^{m/2}\left[1.99 - 0.41\left(\frac{a_n}{2.5}\right) + 18.7\left(\frac{a_n}{2.5}\right)^2 - 38.48\left(\frac{a_n}{2.5}\right)^3 + 53.85\left(\frac{a_k}{2.5}\right)^4\right]^m$$

Y despejando a_{n+1} se tiene el valor del tamaño de la grieta al siguiente ciclo de carga

$$a_{n+1} = a_n + \Delta N A(\Delta\sigma)^m a_k^{\frac{m}{2}}\left[1.99 - 0.41\left(\frac{a_n}{2.5}\right) + 18.7\left(\frac{a_n}{2.5}\right)^2 - 38.48\left(\frac{a_n}{2.5}\right)^3 + 53.85\left(\frac{a_k}{2.5}\right)^4\right]^m$$

$$(E7.1)$$

Una forma de implementar la solución es con un algoritmo de MATLAB que se detenga al alcanzar el valor de la grieta crítica. Con los valores del ejemplo 6.1 y 3.1 se tiene el algoritmo en MATLAB que implementa la ecuación E7.1:

Algoritmo E7.1. Crecimiento de grieta en MATLAB por el método de Euler

```
function [N,a]=CrecimientoGrieta
A=6.6e-9;
w=2.5; %[in]
ds=17.78; %[ksi]
m=2.25;
N(1)=0;
a(1)=0.25; %[in]
dN=1;
n=1;
while a<0.6200 %[in] Grieta Crítica
    a(n+1)=a(n)+dN*A*ds^m*a(n)^(m/2)*(1.99-
0.41*(a(n)/w)+18.7*(a(n)/w)^2-38.48*(a(n)/w)^3+53.85*(a(n)/w)^4)^m;
    N(n+1)=N(n)+dN;
    n=n+1;
end
```

El algoritmo así implementado dice que el número de ciclos para alcanzar la grieta crítica y la falla es N = 37102 ciclos. Compárese este valor con el valor hallado por integración numérica en el ejemplo 6.1. La gráfica del crecimiento de grieta por el método de Euler se muestra en la figura 7.1, compárese con la figura 6.2 del crecimiento de la grieta hallado por integración numérica en el ejemplo 6.1.

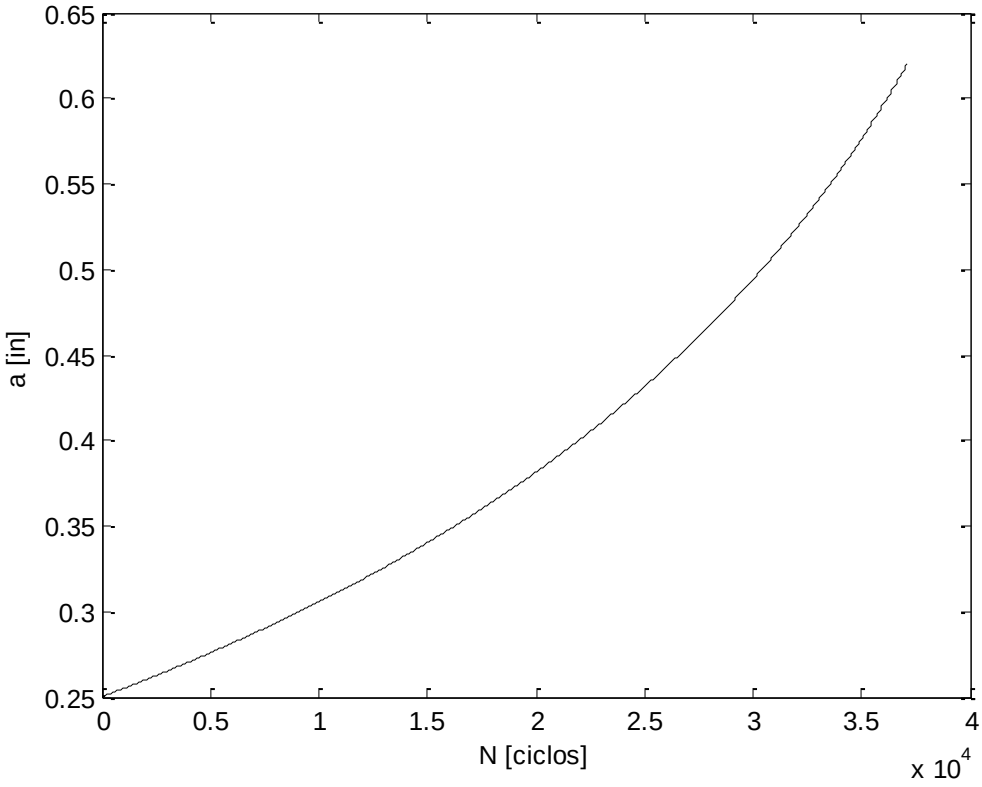

Figura 7.1. Crecimiento de la grieta con el número de ciclos hasta la falla.

7.2. MÉTODO DE RUNGE-KUTTA DE ORDEN 4

Uno de los métodos más utilizados para resolver numéricamente problemas de ecuaciones diferenciales ordinarias con condiciones iniciales es el método de Runge-Kutta de cuarto orden, el cual proporciona un pequeño margen de error con respecto a la solución real del problema y es fácilmente programable en un software para realizar las iteraciones necesarias. Hay variaciones en el método de Runge-Kutta de cuarto orden pero el más utilizado es el método en el cual se elige un tamaño de paso Δt y un número máximo de iteraciones N tal que

$$k_1 = \Delta t \cdot f(t_k, y_k)$$

$$k_2 = \Delta t \cdot f\left(t_k + \frac{\Delta t}{2}, y_k + \frac{k_1}{2}\right)$$

$$k_3 = \Delta t \cdot f\left(t_k + \frac{\Delta t}{2}, y_k + \frac{k_2}{2}\right)$$

$$k_4 = \Delta t \cdot f(t_k + \Delta t, y_k + k_3)$$

$$y_{k+1} = y_k + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Para k = 0,..., N-1. La solución se da a lo largo del intervalo (t_o, t_o+ Δt N). En el algoritmo 7.2 se muestra el algoritmo del método de Runge-Kutta de orden 4 en pseudocódigo parecido a MATLAB, debido a que no hay forma de escribir un código en MATLAB general para este método.

Algoritmo 7.2: Método de Runge-Kutta de orden 4 (pseudocódigo)
Entradas: valor inicial y0, tiempo inicial t0, tamaño de paso dt, número de puntos N
Salidas: valores y tal que dy/dt = f(t,y)

```
y(1)=y0;
t(1)=t0;
for n=2:N
    k1=dt*f(t(n-1),y(n-1));
    k2=dt*f(t(n-1)+dt/2,y(n-1)+k1/2);
    k3=dt*f(t(n-1)+dt/2,y(n-1)+k2/2);
    k4=dt*f(t(n-1)+dt,y(n-1)+k3);
    y(n)=y(n-1)+1/6*(k1+2*k2+2*k3+k4);
    t(n)=t(n-1)+dt;
end
```

Ejemplo 7.2. Velocidad en medios con arrastre

La ecuación diferencial que rige la velocidad v de un cuerpo de masa m y área proyectada A que cae en un medio de densidad ρ es

$$\frac{dv}{dt} = g - \frac{\rho A v^2}{2m} \qquad (E7.2)$$

El cuerpo adquiere su velocidad terminal de caída cuando no acelera más, es decir, la derivada de la velocidad es cero. De acuerdo a E7.2, la velocidad terminal teórica es

$$v_{f,teórica} = \sqrt{\frac{2mg}{\rho A}} \qquad (E7.3)$$

Supóngase una moneda con m = 0.010kg y A = 3.1416×10^{-4} m^2, que cae de un edificio, entonces ρ = 1kg/m^3. La velocidad terminal según E7.3 es $v_{f,teórica} = 24.98 m/s$.

Resolver la ecuación E7.2 por el método de Runge-Kutta y compara la velocidad terminal así hallada con la velocidad terminal teórica.

La iteración de Runge-Kutta se hace como sigue para este caso particular:

$$k_1 = \Delta t[9.8 - 0.0157 v_n^2]$$

$$k_2 = \Delta t\left[9.8 - 0.0157\left(v_n + \frac{k_1}{2}\right)^2\right]$$

$$k_3 = \Delta t\left[9.8 - 0.0157\left(v_n + \frac{k_2}{2}\right)^2\right]$$

$$k_4 = \Delta t[9.8 - 0.0157(v_n + k_3)^2]$$

$$v_{n+1} = v_n + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

Tomando intervalo Δt=1s y velocidad inicial nula, se tiene el método implementado en una hoja de Excel con los valores en la tabla, y se muestra cómo se desarrolla la velocidad n la figura 7.2:

t [s]	V [m/s]	k1	k2	k3	k4
0	0.00	9.80	9.42	9.45	8.40
1	9.32	8.43	6.92	7.23	5.49
2	16.36	5.59	4.03	4.49	2.97
3	20.63	3.11	2.07	2.43	1.45
4	22.89	1.57	1.00	1.21	0.68
5	24.00	0.75	0.47	0.58	0.31
6	24.52	0.35	0.22	0.27	0.14
7	24.77	0.16	0.10	0.12	0.07
8	24.88	0.08	0.05	0.06	0.03
9	24.93	0.03	0.02	0.03	0.01
10	24.96	0.02	0.01	0.01	0.01
11	24.97	0.01	0.00	0.01	0.00
12	24.97	0.00	0.00	0.00	0.00
13	24.98	0.00	0.00	0.00	0.00
14	24.98	0.00	0.00	0.00	0.00
15	24.98	0.00	0.00	0.00	0.00
16	24.98	0.00	0.00	0.00	0.00
17	24.98	0.00	0.00	0.00	0.00
18	24.98	0.00	0.00	0.00	0.00
19	24.98	0.00	0.00	0.00	0.00
20	24.98	0.00	0.00	0.00	0.00

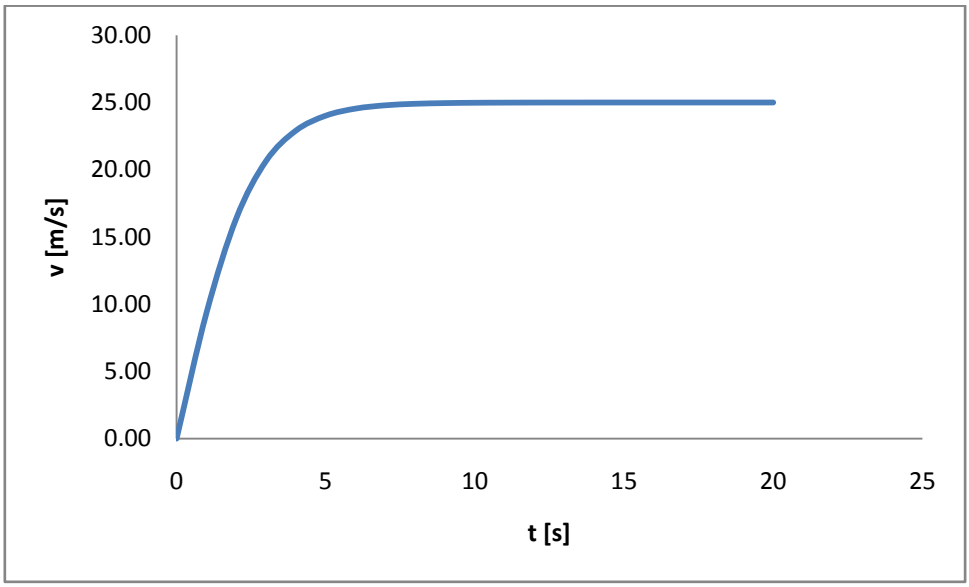

Figura 7.2. Velocidad de la moneda a medida que cae.

Se observa que la velocidad terminal hallada por el método numérico es de 24.98m/s, la cual es la misma teórica, lo que demuestra el poderío del método de Runge-Kutta para la solución numérica de ecuaciones diferenciales.

NOTA: la velocidad terminal hallada es de 89.92 km/h, por lo cual NO se recomienda arrojar monedas desde un edificio.

BIBLIOGRAFÍA RECOMENDADA

- *MÉTODOS NUMÉRICOS PARA INGENIEROS*. Chapra/Canale. Mc-Graw Hill. 4a Edición.
- *ANÁLISIS NUMÉRICO Richard* L. Burden / J. Douglas Faires

SOBRE EL AUTOR

Carlos Armando De Castro - Ingeniero Mecánico de la Universidad de los Andes de Bogotá, Colombia.