Instituto Tecnológico de Buenos Aires

Trabajo Práctico Integrador

93.54 - Métodos Numéricos

AUTORES:
Agustin Vazquez (61420)
Camila Aldara Solari (61623)
Carolina Denise Silveri (61203)
Francisco Mendizabal (61454)
Valentino Venier Anache (60097)

PROFESORES:
Alejandro Hayes
César Jesús Espinoza
Juan Pablo Grisales Campeón

Noviembre 2023

1. Ejercicio 1

Dado el sistema de ecuaciones: $\mathbf{A}\vec{x} = \vec{b}$, donde:

$$\mathbf{A} = \begin{pmatrix} 3 & -1 & 0 & 0 & 0 \\ 0 & 5 & 0 & -1 & 2 \\ 1 & -1 & 9 & 0 & 0 \\ -2 & 3 & -1 & 12 & 1 \\ 0 & 0 & -1 & 1 & 15 \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} -1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

1. Resolviendo por eliminación de Gauss, con la función "eliminacion_gauss" proporcionadas por la cátedra, se obtiene que $\mathbf{T}\vec{x} = \vec{b}$, donde:

$$\mathbf{T} = \begin{pmatrix} 3 & -1 & 0 & 0 & 0 \\ 0 & 5 & 0 & -1 & 2 \\ 0 & 0 & 9 & -0.1333 & 0.2667 \\ 0 & 0 & 0 & 12.4519 & 0.0963 \\ 0 & 0 & 0 & 0 & 15.0220 \end{pmatrix}, \quad \vec{b} = \begin{pmatrix} -1 \\ 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

   Finalmente, con la función "sust_reg" se resuelve el sistema triangular, llegando a la solución:

$$\vec{x} = \begin{pmatrix} -0.3317361001 \\ 0.0047916997 \\ 0.1485030889 \\ 0.0386108031 \\ 0.0073261524 \end{pmatrix}$$

2. Para resolver el sistema por descomposición LU, fue necesario implementar una función propia. El código se muestra a continuación:

```
function [L, U, Y] = LU(A, B)
    [m, n] = size(A);

    % Inicializar matrices L, U
    L = eye(m);
    U = A;
    P = eye(m);

    for k = 1:min(m-1, n)
        % Buscar el índice de fila con el máximo valor en la columna k
        [~, pivot_row] = max(abs(U(k:m, k)));
        pivot_row = pivot_row + k - 1;

        % Intercambiar filas en U y P
        U([k, pivot_row], :) = U([pivot_row, k], :);
        P([k, pivot_row], :) = P([pivot_row, k], :);
```

```
17
18          % Eliminación gaussiana para calcular L y U
19          for i = k+1:m
20              factor = U(i, k) / U(k, k);
21              L(i, k) = factor;
22              U(i, :) = U(i, :) - factor * U(k, :);
23          end
24      end
25
26      LB=[L,B];
27      s=size(LB);
28      nfilas=s(1);
29      ncolumnas=s(2);
30      Y_aux=sust_reg_inf(LB(1:nfilas,1:ncolumnas-1),LB(:,ncolumnas));
31
32      ULB=[U,Y_aux];
33      s=size(ULB);
34      nfilas=s(1);
35      ncolumnas=s(2);
36      Y=sust_reg_sup(ULB(1:nfilas,1:ncolumnas-1),ULB(:,ncolumnas));
37  end
38
39  function x = sust_reg_inf(T, B)
40      s = size(T);
41      nfilas = s(1);
42
43      x = zeros(nfilas, 1);
44
45      x(1) = B(1) / T(1, 1);
46
47      for i = 2:nfilas
48          suma = B(i);
49          for j = 1:(i-1)
50              suma = suma - T(i, j) * x(j);
51          end
52          x(i) = suma / T(i, i);
53      end
54  end
55
```

Listing 1: LU.m

Cabe destacar que la función "sust_reg_sup" es la función "sust_reg" con el nombre modificado, y a partir de la misma se implementó "sust_reg_inf" para matrices triangulares inferiores.

Resolviendo por descomposición LU, se obtiene que $\mathbf{LU}\vec{x} = \vec{b}$, donde:

$$\mathbf{L} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0.3333 & -0.1333 & 1 & 0 & 0 \\ -0.6667 & 0.4667 & -0.1111 & 1 & 0 \\ 0 & 0 & -0.1111 & 0.0791 & 1 \end{pmatrix}$$

$$\mathbf{U} = \begin{pmatrix} 3 & -1 & 0 & 0 & 0 \\ 0 & 5 & 0 & -1 & 2 \\ 0 & 0 & 9 & -0.1333 & 0.2667 \\ 0 & 0 & 0 & 12.4519 & 0.0963 \\ 0 & 0 & 0 & 0 & 15.0220 \end{pmatrix}$$

En este caso se obtiene una solución exactamente igual a la anterior, al menos con 10 dígitos decimales.

3. Para resolver por factorización QR, también fue necesario implementar una nueva función:

```
function [Q, R, Y] = QR(A, B)
    [m, n] = size(A);
    Q = zeros(m, n);
    R = zeros(n);

    for j = 1:n
        % Proyección ortogonal sobre los vectores ya calculados
        v = A(:, j);
        for i = 1:j-1
            R(i, j) = Q(:, i)' * A(:, j);
            v = v - R(i, j) * Q(:, i);
        end

        % Normalizar y almacenar el vector en Q
        R(j, j) = norm(v);
        Q(:, j) = v / R(j, j);
    end

    QB = Q' * B;
    RQB=[R, QB];
    s=size(RQB);
    nfilas=s(1);
    ncolumnas=s(2);
    Y = sust_reg_sup(RQB(1:nfilas,1:ncolumnas-1), RQB(:,ncolumnas));
end

```

Listing 2: QR.m

Resolviendo por descomposición QR, se obtiene que $\mathbf{QR}\vec{x} = \vec{b}$, donde:

$$\mathbf{Q} = \begin{pmatrix} 0.8018 & 0.2127 & -0.2553 & 0.4892 & -0.0857 \\ 0 & 0.9308 & 0.0836 & -0.3524 & 0.0499 \\ 0.2673 & -0.0532 & 0.9517 & 0.0995 & 0.1010 \\ -0.5345 & 0.2925 & 0.0928 & 0.7836 & -0.0781 \\ 0 & 0 & -0.1164 & 0.1121 & 0.9868 \end{pmatrix}$$

$$\mathbf{R} = \begin{pmatrix} 3.7417 & -2.6726 & 2.9399 & -6.4143 & -0.5345 \\ 0 & 5.3719 & -0.7712 & 2.5796 & 2.1541 \\ 0 & 0 & 8.5885 & 0.9136 & -1.4866 \\ 0 & 0 & 0 & 9.8675 & 1.7606 \\ 0 & 0 & 0 & 0 & 14.8245 \end{pmatrix}$$

En este caso se obtiene nuevamente una solución exactamente igual a la anterior, al menos con 10 dígitos decimales.

4. Para implementar el método de Jacobi, se utilizó la función "metodo_jacobi" proporcionada por la cátedra. Ajustando $\vec{x}_0 = (0, 0, 0, 0, 0)^T$ y cotaerror = 10^{-6}, se obtiene que:

$$\vec{x} = \begin{pmatrix} -0.3317360001 \\ 0.0047916937 \\ 0.1485031241 \\ 0.0386107451 \\ 0.0073261769 \end{pmatrix}$$

con un error $E \leq 3.0607 \cdot 10^{-7}$, luego de 11 iteraciones.

5. Para implementar el método de Gauss-Seidel, se utilizó la función "metodo_gauss_seidel" proporcionada por la cátedra. Ajustando $\vec{x}_0 = (0, 0, 0, 0, 0)^T$ y cotaerror = 10^{-6}, se obtiene que:

$$\vec{x} = \begin{pmatrix} -0.3317360404 \\ 0.0047916723 \\ 0.1485030792 \\ 0.0386108181 \\ 0.0073261507 \end{pmatrix}$$

con un error $E \leq 5.3972 \cdot 10^{-7}$, luego de 7 iteraciones.

6. Para implementar el método SOR, primero se utilizó un código para analizar la velocidad de convergencia con distintos valores de ω, realizando 25 iteraciones y partiendo del mismo punto inicial en todos los casos. Los resultados se muestran a continuación:

   Cabe aclarar que los puntos en $\omega = 0.9$ y $\omega = 1$ no se grafican porque la escala es logarítmica, y el resultado obtenido para dichos valores fue 0.

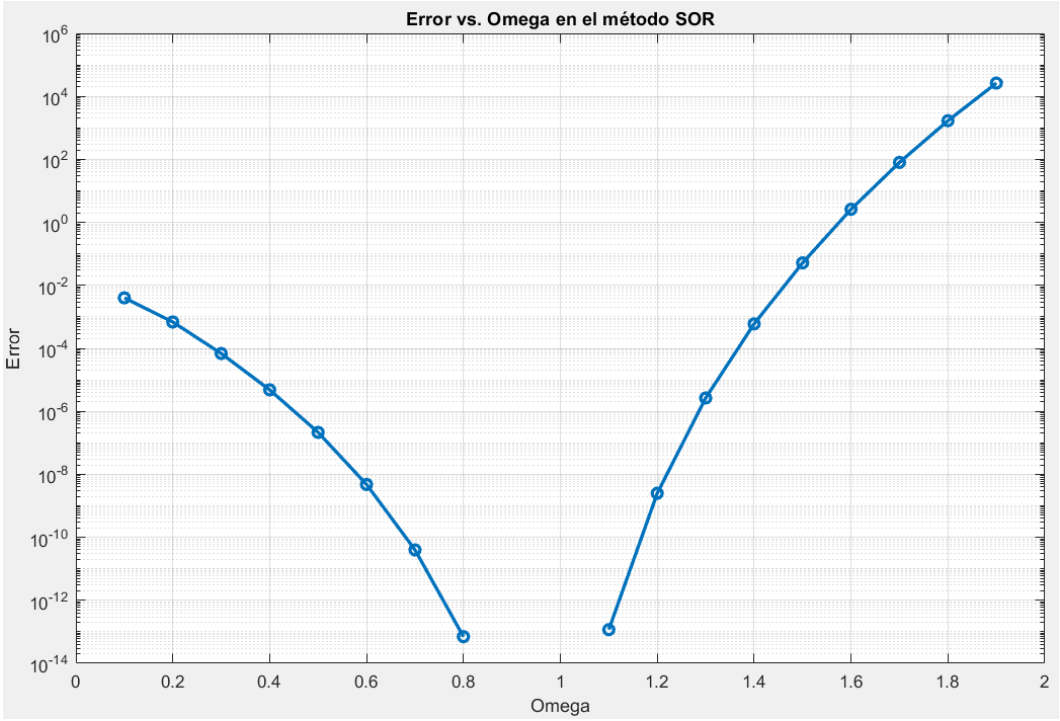

FIGURA 1.1: Error obtenido a partir de $\vec{x}_0 = \vec{0}^T$ luego de 25 iteraciones

Con esto, dado que aplicar $\omega = 1$ sería repetir el item anterior, se aplicó $\omega = 0.9$, utilizando el algoritmo "sor" proporcionado por la cátedra. El resultado obtenido fue el siguiente:

$$\vec{x} = \begin{pmatrix} -0.3317358969 \\ 0.0047917715 \\ 0.1485030623 \\ 0.0386108316 \\ 0.0073261456 \end{pmatrix}$$

con un error $E \leq 7.8362 \cdot 10^{-7}$, luego de 7 iteraciones. En términos del error, notamos una performance levemente peor a la de Gauss-Seidel.

2. Ejercicio 2

Dado un sistema de ecuaciones lineales $AX = B$ donde:

$$A = \begin{bmatrix} 4 & 1 & 2 \\ 1 & 2 & 0 \\ 2 & 0 & 5 \end{bmatrix} \tag{2.1}$$

$$B = \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \tag{2.2}$$

Se resolverá la el sistema presentado con la factorización de *Cholenky*.

El método de *Cholenky* propone que existe una matriz `C` que se puede modelar como la multiplicación de una matriz con su transpuesta, siendo esta matriz triangular. Por lo tanto, se puede partir desde la siguiente ecuación:

$$A^T AX = A^T B \tag{2.3}$$

Siendo:

$$C = A^T A = \begin{bmatrix} 21 & 6 & 18 \\ 6 & 5 & 2 \\ 18 & 2 & 29 \end{bmatrix} \tag{2.4}$$

$$D = A^T B = \begin{bmatrix} -2 \\ -1 \\ 3 \end{bmatrix} \tag{2.5}$$

Luego, se propone una matriz G que se caracteriza por ser una matriz triangular superior el cual forma parte de la matriz `C`.

$$C = GG^T \tag{2.6}$$

Resolviendo la matriz `G` por factorización de *Cholenky*, se tiene que:

$$G = \begin{bmatrix} 2 & 0 & 0 \\ 0.5 & 1.3229 & 0 \\ 1 & -0.378 & 1.964 \end{bmatrix} \tag{2.7}$$

Una vez obtenida la matriz `G`, se sigue despejando hasta llegar a la solución buscada.

$$GG^T X = D \tag{2.8}$$

$$G^T X = Y \tag{2.9}$$

Por lo tanto, de las ecuaciones dadas, se calcula que:

$$GY = D \tag{2.10}$$

Despejando queda que:

$$Y = \begin{bmatrix} -1.6839 \\ -0.3195 \\ 1.5275 \end{bmatrix} \tag{2.11}$$

Luego se despeja la matriz `X` de la ecuación (2.9) ya que se obtuvo `Y` recientemente. El resultado del sistema resulta ser:

$$X = \begin{bmatrix} -0.8419 \\ 0.0767 \\ 1.2212 \end{bmatrix} \tag{2.12}$$

Todos los cálculos realizados se hicieron mediante un código de `MATLAB` que se muestra en el siguiente *script*.

```
clc
clear all

A = [4, 1, 2;
     1, 2, 0;
     2, 0, 5];

B = [-1;
      0;
      1];

C = A'*A

D = A'*B

% Verifico si la matriz A es simétrica y definida positiva
if issymmetric(A) && all(eig(A) > 0)
    % Factorización de Cholesky
    G = chol(A, 'lower')
    % Resolvo GY = D
    Y = G'\D
    % Resolvo G'X = Y
    X = G\Y
else
    disp('La matriz A no es simétrica y definida positiva. El método de
    Cholesky no se puede aplicar.');
end
```

Listing 3: Main: `Cholenky.m`

Luego de correr el código, se obtuvieron los siguientes resultados:

```
C =

    21     6    18
     6     5     2
    18     2    29


D =

    -2
    -1
     3


G =

    2.0000         0         0
    0.5000    1.3229         0
    1.0000   -0.3780    1.9640


Y =

   -1.6839
   -0.3195
    1.5275


X =

   -0.8419
    0.0767
    1.2212
```

3. Ejercicio 3

Un investigador reporta los datos tabulados a continuación, de un experimento para determinar la tasa de crecimiento de bacterias k, como función de la concentración de oxígeno x. Se sabe que dichos datos pueden modelarse por medio de la siguiente ecuación:

$$k(x) = K_{max} * \left(\frac{x^2}{x^2 + C_s}\right) \tag{3.1}$$

donde C_s y K_{max} son parámetros. Use una transformación para hacer lineal esta ecuación. Después utilice regresión lineal para estimar C_s y K_{max}, y pronostique la tasa de crecimiento para $x = 2mg/L$.

x	0.5	0.8	1.5	2.5	4
k	1.1	2.4	5.3	7.6	8.9

Tabla 3.1: Información Dada.

Para poder resolver el ejercicio propuesto, decidimos comenzar por plantear y(x) a partir de la ecuación dada para k(x) así nos queda de la siguiente manera:

$$y(x) = \left(\frac{1}{Kmax}\right) * \left(\frac{x^2 + C}{x^2}\right) \tag{3.2}$$

Entonces tenemos:

$$y(x) = \left(\frac{a}{x^2}\right) + b \tag{3.3}$$

Siendo

$$c = a * KmaxyKmax = 1/b$$

Entonces, para poder plantear la matriz A vamos a reemplazar los valores de x para la primera columna y completando con unos la segunda columna. A partir de la ecuación planteada antes, vamos a querer encontrar una expresión para poder ajustar los puntos dados a la ecuación. Luego buscamos aplicar cuadrados mínimos con `QR` para que quede de la forma

$$||A * x^2 - c||^2 \tag{3.4}$$

. Con la matriz `A` y el vector `c` de la siguiente forma:

$$\mathbf{A} \begin{pmatrix} 4 & 1 \\ 1.5625 & 1 \\ 4/9 & 1 \\ 0.16 & 1 \\ 0.0625 & 1 \end{pmatrix}, \quad \vec{c} = \begin{pmatrix} 1.1 \\ 2.4 \\ 5.3 \\ 7.6 \\ 8.9 \end{pmatrix}$$

Como la matriz A no es cuadrada, se busca la matriz B, donde:

$$B = A^T A = \begin{bmatrix} 18.67 & 6.23 \\ -6.23 & 5 \end{bmatrix} \tag{3.5}$$

Además, tenemos:

$$D = A^T c = \begin{bmatrix} 12.28 \\ 25.3 \end{bmatrix} \tag{3.6}$$

A continuación, veremos el código utilizado en Octave para la resolución del ejercicio:

```
%Creo una funcion para calcular y
function[y]= transformacion (x)
    y= 1./ (x.^2);
end
%Con los valores obtenidos se arma la matriz A
function[matriz]= armamatriz(y)
    n=length(y);
    matriz=[y,ones(n,1)];
end

matrizA=armamatriz(y)

%Hago una funcion para trasponer la matriz
function[A_transpose]= traspuesta(A)
    A_transpose= transpose(A)
end

matrizAT= traspuesta(matrizA)

%Calculo la matriz B
matrizB= matrizAT*matrizA

function[resultado]=producto_matricial(matriz,vector)
    resultado=matriz*vector;
end

%Obtengo el vectorD
vectorD= producto_matricial(matrizAT,k)

%Programa para factorizar una matriz por el metodo de Cholesky

function L = factorizacionCholesky(A)
    % Dimensiones de la matriz A
    n = size(A, 1);
    % Inicializar la matriz L
    L = zeros(n);
    % Realizar la factorización de Cholesky
    for j = 1:n
```

```
        for i = 1:j
            if i == j
                L(j, j) = sqrt(A(j, j) - sum(L(j,1:j-1).^2));
            else
                L(j, i) = (A(j, i) - sum(L(i, 1:i-1) * L(j, 1:i-1)')) / L(i,
    i);
            end
        end
    end
end

L=factorizacionCholesky(matrizB)

function x = despejo_cholesky (L, b)
    % Dimensiones de la matriz L y el vector b
    n = size(L, 1);
    % Resolución de Ly = b mediante sustitución hacia adelante
    y = zeros(n, 1);
    for i = 1:n
        y(i) = (b(i) - L(i, 1:i-1) * y(1:i-1)) / L(i, i);
    end
    % Resolución de L'x = y mediante sustitución hacia atrás
    x = zeros(n, 1);
    for i = n:-1:1
    x(i) = (y(i) - L(i+1:n, i)' * x(i+1:n))/ L(i,i);
    end
end

x=despejo_cholesky(L, vectorD)
%Para pronosticar la tasa de crecimiento para x=2 mg/L
function tasa_crecimiento(2,kmax,c)
    tasa=kmax * x^2 / (x^2+c)
end

tasa=tasa_crecimiento (2,kmax,c)
```

Listing 4: Codigo en Octave

4. Ejercicio 4

En este ejercicio práctico se buscará un intervalo que contenga la solución de $f(x)$ y además se buscará resolverlo por el método de bisección con un error menor que 10^{-6}.

$$e^{-x^2} = 1 - \frac{1}{x^2 + 1} \tag{4.1}$$

Dada esta fórmula, se propone una función de la cuál se hallará la solución:

$$f(x) = e^{-x^2} - 1 + \frac{1}{x^2 + 1} \tag{4.2}$$

Se puede notar que la función es continua en los reales. Dado a esto, se calculan dos posibles valores en los cuales se sospecha en donde podría estar la raíz, $f(0) = 1$ y $f(1) = -0.1321$. Se observa la diferencia entre ambos resultados, hay cambio de signo, por lo tanto, por Teorema de *Bolzano* existe un $c \in (0, 1) / f(c) = 0$.

Como consecuencia, se puede tomar el intervalo $[0, 1]$ para iniciar la búsqueda de la solución.

Como consiguiente, se programó un código en `MATLAB` para poder calcular la solución adecuada respetando las restricción ya mencionada del error menor que 10^{-6}. Se tienen dos archivos, uno el cuál contiene la función y otro el que contiene los parámetros calculados y la llamada a la función.

```
clc
clear

% Define los extremos del intervalo [a, b] y la cota de error
a = 0;
b = 1;
cotaerror = 1e-6;  % Por ejemplo, puedes usar 1e-6 para una tolerancia de
    10^-6
% Número máximo de iteraciones
maxiter = 1000;

disp('Biseccion')
% Llama a la función de bisección y almacena los resultados en las variables
    de salida
[sol, niter, error] = biseccion(a, b, cotaerror, maxiter);
```

Listing 5: Main: `test.m`

```
%Programa para el metodo de biseccion
%a,b extremos del intervalo
function [sol,niter,error]=biseccion(a,b,cotaerror, maxiter)
cont=0;
xant=a;
xsig=b;
while (abs(xsig-xant)>cotaerror) && (cont < maxiter)
    cont=cont+1;
    if fun(xant)*fun(xsig)<0
        xmed=(xant+xsig)/2;
        if fun(xmed)*fun(xant)<0
            xsig=xmed;
        elseif fun(xsig)*fun(xmed)<0
            xant=xmed;
        else
            sol=xmed;
            error=0;
            niter=cont;
            break
        end
    elseif fun(xant)*fun(xsig)>0
        disp('Intervalo Incorrecto')
        break
    elseif fun(xant)==0
       sol=xant;
       error=0;
       niter=cont;
       break
   else
       sol=xsig;
       error=0;
       niter=cont;
       break
   end
end
sol = xmed;
niter = cont;
error = abs(xsig - xant);
% Muestra los resultados
fprintf('La solución aproximada es: %f\n', sol);
fprintf('Número de iteraciones: %d\n', niter);
fprintf('Error estimado: %e\n', error);
end

function y = fun(x)      %funcion f(x)
    y = exp(-(x^2)) - 1 + (1/((x^2)+1));
end
```

Listing 6: Funcion de Bisección: `biseccion.m`

Luego de correr el código, se obtuvieron los siguientes resultados:

```
Biseccion
La solución aproximada es: 0.898034
Número de iteraciones: 20
Error estimado: 9.536743e-07
```

Concluyendo el ejercicio, el resultado visto en la terminal y el numero de iteración fueron los correctos con un error menor que 10^{-6}.

5. Ejercicio 5

La ecuación:

$$c(t) = c_e \left(1 - e^{-0.04t}\right) + c_0 e^{-0.06t} \tag{5.1}$$

permite calcular la concentración de un químico en un reactor donde se tiene una mezcla completa. Si la concentración inicial es $c_0 = 5$ y la concentración de entrada es $c_e = 12$, calcule el tiempo requerido para que c sea el 85 % de c_e con una cota de error menor a 10^{-6}. Para ello

1. Hallar algún intervalo que contenga la solución.

   Para esta parte del ejercicio planteamos la siguiente función

   $$f(t) = c(t) - c_{req} = c(t) - 0.85c_e \tag{5.2}$$

   para luego poder observar la misma para hallar el intervalo que nos solicitan, siendo este el que contenga una raíz de $f(t)$.

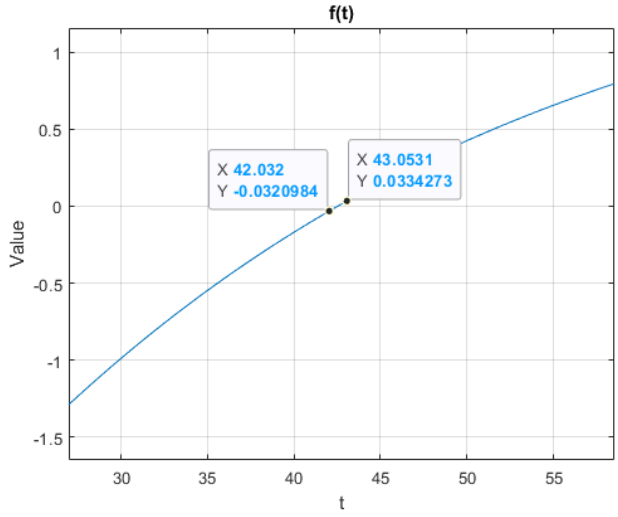

FIGURA 5.1: Raíz de $f(t)$

   por lo que queda definido un intervalo que contiene $f(t) = 0$, siendo este $I = [42 : 43]$

```
%% Primero expresamos la ecuacion que rige el sistema

Co = 5;
Ce = 12;
c = @(t) Ce.*(1-exp(-0.04*t)) + Co.*exp(-0.06*t);
cReq = 0.85*Ce;
error = 1e-6;

%% A)
%Para esto solo es requerito plotear f(t) = c(t) - cReq.
f = @(t)c(t)-cReq ;
```

```
12 x = linspace(0,85,1000);
13 plot(x,f(x))
14 %Notamos que la raiz se encentra entre 42 y 43.
```

Listing 7: `Ej5.m`

2. Realizar un programa que permita resolver el problema por el método de punto fijo.

   Lo primero que debemos hallar es la función $g(t)$ tal que podamos aplicar el método, por lo que proponemos

$$g(t) = t - f(t) = t + 12 * e^{-t/25} - 5 * e^{-3t/50} - 9/5 \tag{5.3}$$

   Donde notamos a simple vista que la función es C^{∞} por lo que cumple con ser C^1 en el intervalo I.

   Luego, evaluamos la función en el intervalo para verificar si cumple que $g([42:43]) \in [42:43]$

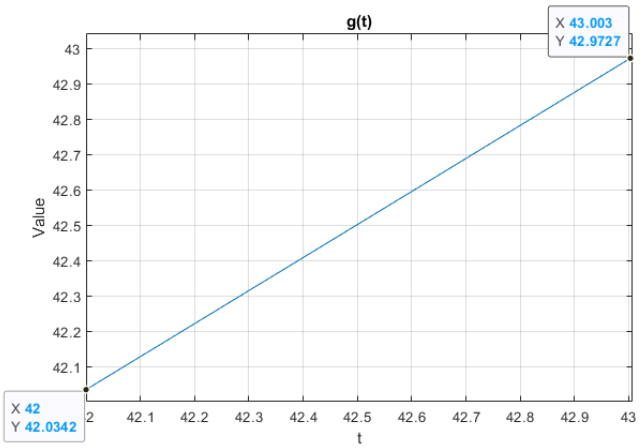

Figura 5.2: $g(t)$

   Observamos que se cumple también esta hipótesis, como también de aquí podemos extrapolar que la derivada de la función es acortada y es menor a 1, no obstante vamos a verificarlo.

$$g'(t) = \frac{3\,\mathrm{e}^{-\frac{3t}{50}}}{10} - \frac{12\,\mathrm{e}^{-\frac{t}{25}}}{25} + 1 \tag{5.4}$$

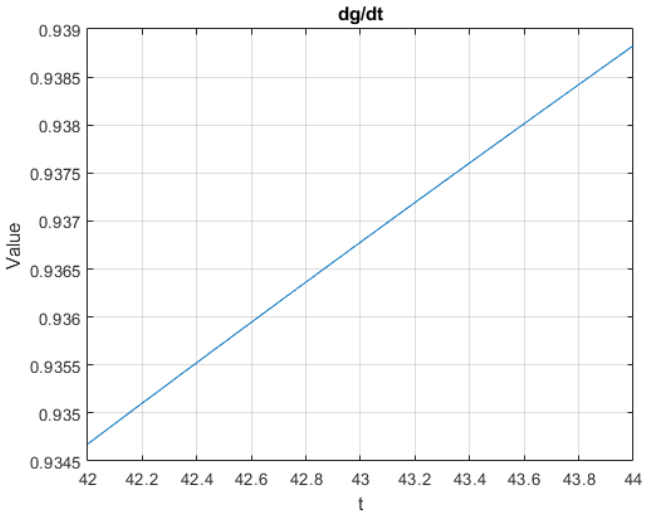

FIGURA 5.3: $g'(t)$

Finalmente podemos acotar la derivada por $|g'(t)| < 0.95 < 1$ cumpliéndose así la tercer y última hipótesis para poder realizar el método de punto fijo.

El programa para las soluciones anteriores y aplicar el método es el siguiente.

```
%% B)
%Propongo como g(t) = t - f(t)
g = @(t) t - f(t);
x = linspace(42,44,1000);
plot(x,g(x))
%Al plotear la función g(t), notamos que cumple dos de las hipotesis.
%Ahora observamos que ocurre con la derivada.
figure
syms n
dg = diff(g(n));
plot(x,subs(dg,n,x));
%Como cumple tambien con esta hipotesis, puedo aplicar el metodo
xi =43;
while(1)
    xi = g(xi);
    if (abs(f(xi)) < error)
        break
    end
end

```

LISTING 8: `Ej5.m`

Siendo el resultado obtenido el siguiente:

```
El valor de la raíz es: 42.5279435816
El módulo del error es: 9.7581e-07
```

3. Realizar un programa que permita resolver el problema por el método de *Newton-Raphson*.

   En este caso, redefinimos a la función $g(t)$ de la siguiente manera:

$$g(t) = t - \frac{f(t)}{f'(t)} \tag{5.5}$$

   y el programa que se realizó fue el siguiente

```
%% C
% Ahora hay que redefinir g(t) = t - g(t)/g'(t)
syms n
df = diff(f(n));
df = @(t) subs(df,n,t)
g = @(t) t - f(t)./df(t)

xi =43;

while(1)
    xi = g(xi)
    if (abs(f(xi)) < error)
        break
    end
end

```

Listing 9: `Ej5.m`

   Siendo el resultado obtenido el siguiente:

```
El valor de la raíz es: 42.5279281628
El módulo del error es: 1.4154e-08
```

6. Ejercicio 6

A partir de la información que nos brinda el siguiente cuadro, se hallará un polinomio de grado tres que interpole la función *f(x)* con el método de *Lagrange*.

x	0	1	2	4
f(x)	4	2	0	8

Tabla 6.1: Información Dada.

Primeramente, se parte de la ecuación de interpolación de *Lagrange*.

$$p_n(x) = \sum_{i=0}^{n} f(x_i)L_i(x) \tag{6.1}$$

Como se debe llegar a un polinomio de orden tres, se necesita calcular la siguiente información:

$$p_3(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x) + f(x_3)L_3(x) \tag{6.2}$$

Por un lado, se calculan los polinomios *L(x)* correspondientes con la siguiente fórmula:

$$L_i(x) = \prod_{i=0, j\neq i}^{3} \frac{x - x_j}{x_i - x_j} \tag{6.3}$$

$$L_0(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} = -\frac{1}{8}(x^3 - 7x^2 + 14x - 8) \tag{6.4}$$

$$L_1(x) = \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} = \frac{1}{3}(x^3 - 6x^2 + 8x) \tag{6.5}$$

$$L_2(x) = \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} = -\frac{1}{4}(x^3 - 5x^2 + 4x) \tag{6.6}$$

$$L_3(x) = \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} = \frac{1}{24}(x^3 - 3x^2 + 2x) \tag{6.7}$$

Una vez calculado los polinomios anteriores, reemplazando en la fórmula (6.2) se halla el polinomio de interpolación buscado, sabiendo que los $f(x_i)$ son los resultados de la tabla 6.1.

$$p_3(x) = \frac{1}{2}x^3 - \frac{3}{2}x^2 - x + 4 \tag{6.8}$$

Por último, se calcula en forma aproximada $f(3)$ a partir del polinomio hallado.

$$f(3) \approx p_3(3) = 1 \tag{6.9}$$

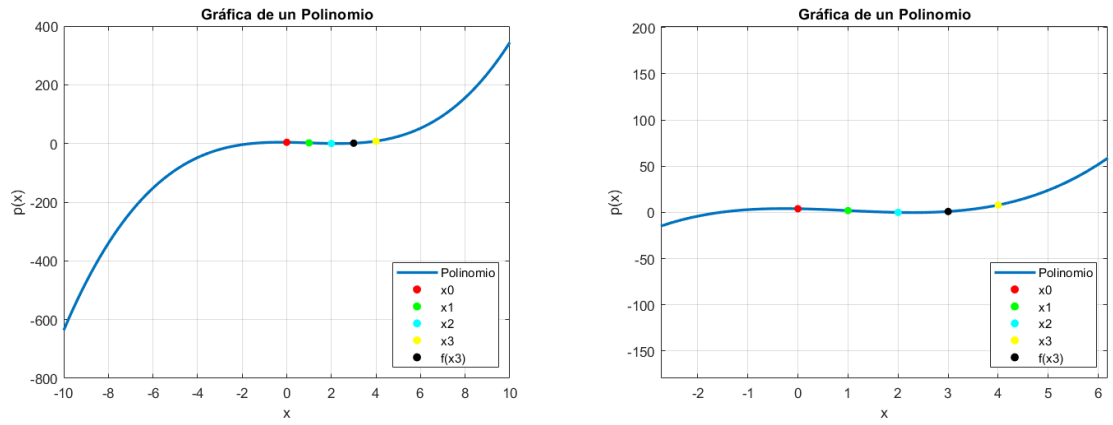

FIGURA 6.1: Polinomio Resultante.

7. Ejercicio 7

En la termodinámica estadística el modelo que describe la capacidad calórica de un sólido según *Debyte* responde a la siguiente función.

$$\Phi(x) = \int_0^x \frac{t^3}{e^t - 1}\, dt \tag{7.1}$$

Se hallaron los valores de $\Phi(x)$ para $x \in [1, 10]$ en pasos de a uno utilizando el método de *Simpson* con un error menor que 10^{-3}.

```
%% Dado que me piden error menor a 10^-3, tomo 1000 puntos por cada intervalo
%Es decir que  h = 1/1000
N = 1000;
h = 1/N;
n = linspace(h,10-h,10*N);
Ts = @(f,n,k) h/3 * (f(n(1)) + f(n(k)) +  4 * sum(f(n(2:2:k-1))) + 2 * sum(f(
    n(3:2:k-1))) );


f = @(t) t.^3 ./ (exp(t)-1);

phi = @(x) Ts(f,n,x*N);

%creo la tabla de valores
phis = [phi(1),phi(2),phi(3),phi(4),phi(5),phi(6),phi(7),phi(8),phi(9),phi
    (10)]';
errors = [integral(f,0,1) - phi(1),
    integral(f,0,2) - phi(2),
    integral(f,0,3) - phi(3),
    integral(f,0,4) - phi(4),
    integral(f,0,5) - phi(5),
    integral(f,0,6) - phi(6),
    integral(f,0,7) - phi(7),
    integral(f,0,8) - phi(8),
```

```
    integral(f,0,9) - phi(9),
    integral(f,0,10) - phi(10)];
tb = table(phis,errors);
disp('Tabla:');
disp(tb);
```

Listing 10: `Ej.7m`

Luego de correr el programa, se obtiene la siguiente tabla de valores:

```
    phis         errors
   2.2458e-01   2.2949e-04
   1.1758e+00   5.4997e-04
   2.5516e+00   6.4066e-04
   3.8766e+00   4.8797e-04
   4.8997e+00   2.1670e-04
   5.5859e+00  -5.7581e-05
   6.0034e+00  -2.7681e-04
   6.2401e+00  -4.2927e-04
   6.3671e+00  -5.2575e-04
   6.4325e+00  -5.8275e-04
```

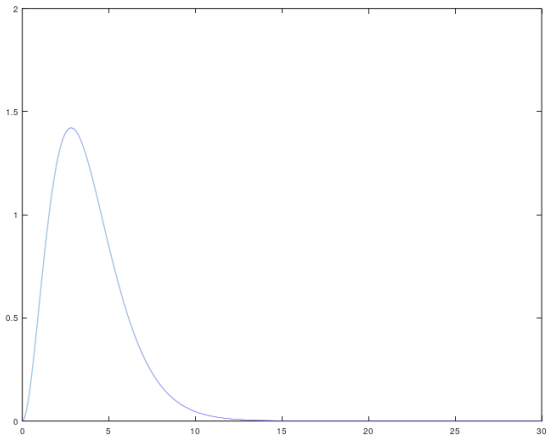

FIGURA 7.1: Función del integrando

Además se graficó la función del integrando para $x \in [0, 30]$ en `MATLAB`.

Para hallar un polinomio que interpole a phi se utilizó el siguiente código otorgado por la cátedra:

```
%Funcion que genera y grafica el polinomio interpolador.
%x:vector de primeras coordenadas de los datos debe ser columna
%y:vector de segundas coordenadas de los datos debe ser columna
function p=interpolador(x,y)
V=vander(x);
sol=V\y';
p=sol';
%graficas
plot(x,y,'*k','linewidth',9)
grid minor
hold on
xlabel('x')
ylabel('y')
title('Polinomio Interpolador de datos')
a=min(x);
b=max(x);
t=a:0.01:b;
gp=polyval(p,t);
%z=exp(t);
plot(t,gp,'r','linewidth',3)
%plot(t,z,'g')
```

Donde ingreso como vector x los valores obtenidos de phi en el punto a) y en el vector y se ingresaron los valores obtenidos de evaluar la función con los valores del vector x. Se obtuvo el siguiente gráfico:

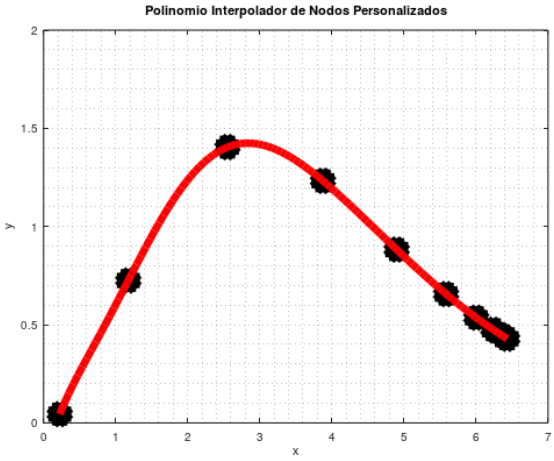

FIGURA 7.2: Polinomio interpolador

Adjudicamos la leve diferencia al primer gráfico al método utilizado.

8. Ejercicio 8

En estudios de óptica es muy común la aparición de las integrales de *Fresnel* dadas por:

$$C(u) = \int_0^u \cos \frac{\pi x^2}{2} dx \qquad S(u) = \int_0^u \sin \frac{\pi x^2}{2} dx \tag{8.1}$$

1. Usando OCTAVE calcular $C(10)$ y $S(10)$ con la regla de los trapecios para $N = 200$

   Primero lo que hay que realizar es la distribución equiespaciada de los puntos en el intervalo de interés, siendo este $I = [0 : 10]$.

```
%% Primero defino el intervalo, los nodos a usar y la funcion a integrar
    .
u = 10;
N = 200;
n = linspace(0,u,N);
h = n(2) - n(1); %Al ser equiespaciadas, puedo consiguer h asi
c = @(t) cos(pi/2 * t.^2);
s = @(t) sin(pi/2 * t.^2);
```

LISTING 11: `Ej8.m`

   Luego lo que resta es aplicar la función que aproxima a la integral por la regla de trapecios. Para poder comparar la performance de los métodos, decidimos calcular la integral con la máxima precisión que nos permite `Matlab/Octave`

```
%% Aplico la Tt(f) de trapceios
Tt = @(f,n) h/2 * (f(n(1)) + f(n(N)) + 2 * sum(f(n(2:N-1))));
C = Tt(c,n)
S = Tt(s,n)
Cmatlab = integral(c,0,10);
Smatlab = integral(s,0,10);
errorC = Cmatlab - C
errorS = Smatlab - S
```

LISTING 12: `Ej8.m`

   Siendo los resultados obtenidos, los siguientes :

```
C = 0.4999019111
S = 0.4750728844
errorC = -3.2169e-06
errorS = -0.0069
```

2. Usando OCTAVE calcular $C(10)$ y $S(10)$ con la regla de *Simpson* para $N = 200$

   Para este caso, vamos a usar la misma distribución de nodos, y lo único que cambiaremos es el método para poder estimar el valor de la integral.

```
%% Aplico Ts(f) de Simpson
% Es anti-intuitivo, pero los pares son los impares en matlab por arrancar a contar desde el 1
Ts = @(f,n) h/3 * (f(n(1)) + f(n(N)) +  4 * sum(f(n(2:2:N-1))) + 2 * sum(f(n(3:2:N-1))) );
C = Ts(c,n)
S = Ts(s,n)
errorC = Cmatlab - C
errorS = Smatlab - S
```

Listing 13: `Ej8.m`

Siendo los resultados obtenidos, los siguientes :

```
C = 0.4831852372
S = 0.4835141276
errorC = 0.0167
errorS = -0.0153
```

9. Ejercicio 9

Dada la ecuación:

$$\frac{dv}{dt} = 9.8 - 0.18[v + (\frac{v}{46})^{2.2}]; v(0) = 0 \tag{9.1}$$

Se busca resolver la misma usando T= 15 con diferentes métodos numéricos.

a) Usando el método de *Euler* con h=0.1s tenemos:

```
function y=metodo_euler(a,b,y0,N)
h=(b-a)/N;
t=a:h:b;
lt=length(t);
y=zeros(1,lt);
y(1)=y0;
for i=1:N
   y(i+1)=y(i)+h*Fed(t(i),y(i));
end
plot(t,y)
xlabel('Tiempo')
ylabel('Velocidad')
title('Grafico de velocidad en funcion del tiempo')
end

function v=Fed(t,v)
v=9.8-0.18*(v+8.3*(v/46)^2.2);
end

%Defino variables para este ejercicio
a=0
b=15
y0=0
N=150

x=metodo_euler(0,15,0,150);
%Vemos cual es la velocidad en t=15 segundos
y=x(151)
```

Listing 14: Metodo de euler

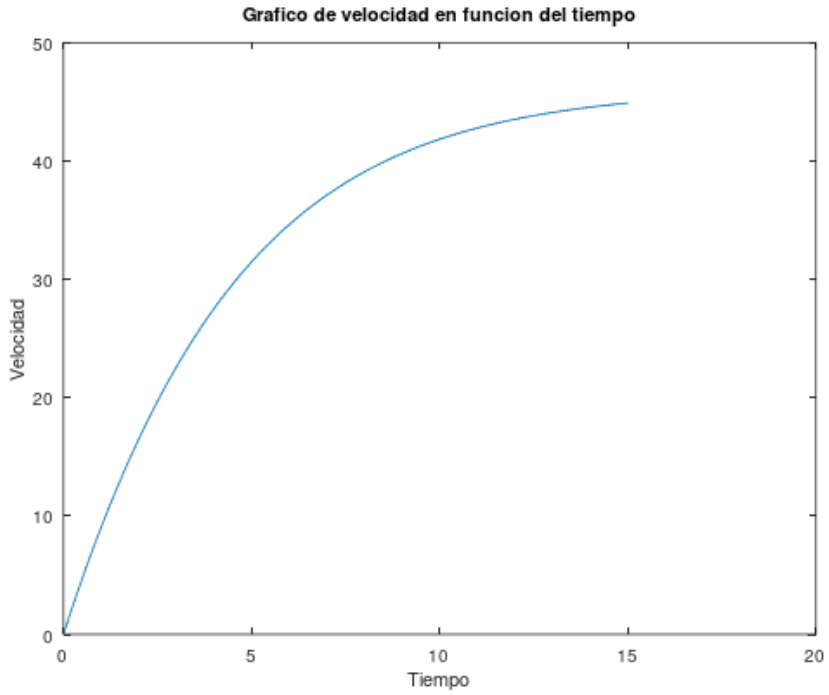

FIGURA 9.1: Velocidad vs Tiempo

b) Usando el método de *Runge-Kutta* de orden cuatro con $h = 0.1s$ tenemos:

```
function y=metodo_runge_kutta_4(a,b,y0,N)
h=(b-a)/N;
t=a:h:b;
lt=length(t);
y=zeros(1,lt);
y(1)=y0;
for i=1:N
    k1=Fed(t(i),y(i));
    k2=Fed(t(i)+0.5*h,y(i)+0.5*k1*h);
    k3=Fed(t(i)+0.5*h,y(i)+0.5*k2*h);
    k4=Fed(t(i)+h,y(i)+k3*h);
    y(i+1)=y(i)+(h/6)*(k1+2*k2+2*k3+k4);
end
plot(t,y)
xlabel('Tiempo')
ylabel('Velocidad')
title('Grafico de velocidad en funcion del tiempo')
end
%Usando los valores del ejercicio
x=metodo_runge_kutta_4(0,15,0,150);
%Velocidad en t=15
y=x(151)
```

LISTING 15: Metodo de Runge-Kutta de orden 4

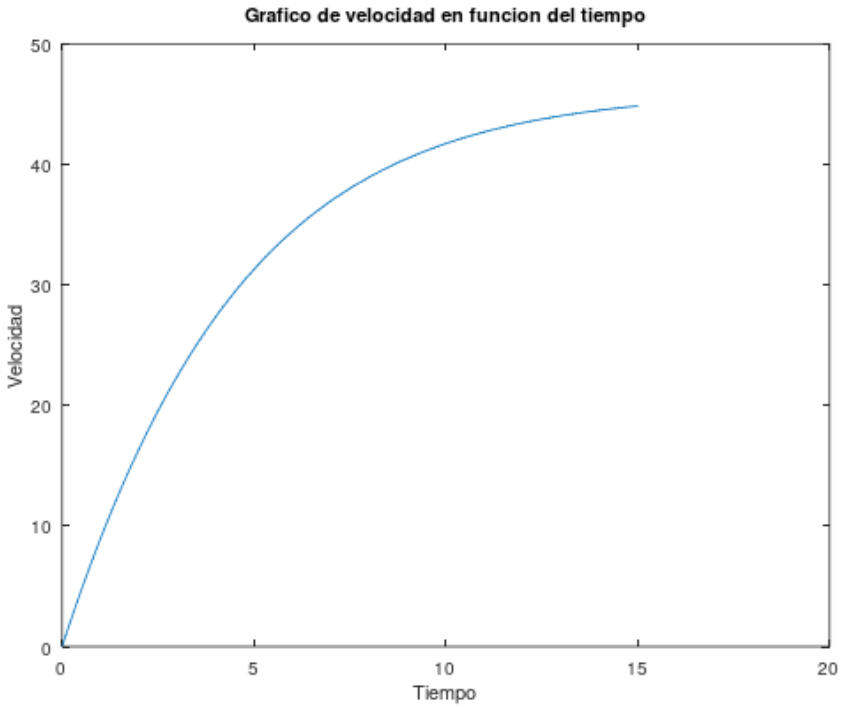

FIGURA 9.2: Velocidad vs Tiempo

c) Para realizar este ítem, se utilizaron las funciones ode23 y ode45 de Octave. Dichas funciones son utilizadas para resolver ecuaciones diferenciales ordinarias (EDOs) como las de este ejercicio. En primer lugar, ode23 es un solucionador de EDOs que utiliza un método de Runge-Kutta implícito de segundo y tercer orden. Analogamente, ode45, es un solucionador mas avanzado que ode23, y usa el método de *Runge-Kutta* de cuarto y quinto orden para lograr mayor precisión.

En primer lugar, veremos el código correspondiente a ode23:

```
function dydt = odefunc(t, y)
    dydt = 9.8 - 0.18 * (y + 8.3 * (y / 46) ^ 2.2);
end

% Condiciones iniciales y tiempo de integración
tspan = [0 20];

% Resolviendo la ecuación diferencial
[t, y] = ode23(@odefunc, tspan, y0);

% Grafico la solución
plot(t, y)
xlabel('Tiempo (s)')
ylabel('Velocidad (m/s)')
title('Solución de la ecuación diferencial usando ode45')

x= y(length(y))
```

LISTING 16: ode23

El código para ode45 resulta:

```
function dydt = odefunc(t, y)
    dydt = 9.8 - 0.18 * (y + 8.3 * (y / 46) ^ 2.2);
end

% Condiciones iniciales y tiempo de integración
tspan = [0 20];

% Resolviendo la ecuación diferencial
[t, y] = ode45(@odefunc, tspan, y0);

% Grafico la solución
plot(t, y)
xlabel('Tiempo (s)')
ylabel('Velocidad (m/s)')
title('Solución de la ecuación diferencial usando ode45')

x= y(length(y))
```

Listing 17: ode45

d) Los valores de la velocidad en t=15 varían dependiendo del método utilizado. En primer lugar, usando el método de *Euler* tenemos v(15)= 44.898 m/s. Usando el método de *Runge-Kutta* de orden 4 se obtuvo v(15)= 44.845 m/s. Finalmente, al calcular la velocidad mediante las funciones elegidas ode23 y ode45 se obtuvieron v(15)= 45.753 m/s y v(15)= 45.744 m/s respectivamente. Las pequeñas diferencias entre los valores especificados anteriormente se deben al método elegido para su calculo. Como fue mencionado en el inciso anterior, la función ode45 permite obtener el resultado de mayor precisión, dado que usa el método de *Runge-Kutta* de cuarto y quinto orden, y entonces se puede concluir que la velocidad en t=15 segundos es cercana a 45.744 m/s.