Asignatura	Datos del alumno	Fecha
Métodos Numéricos I	**Apellidos:** Avilés Cahill	27/02/2024
	Nombre: Adán	

Práctica Grupal, PVI

1. Actividad 1

Un sistema resonante de muelles sobre el que se ejerce una fuerza externa periódica se modela mediante la ecuación:

$$x''(t) = 4\sin(5t) - 25x(t);$$
$$x(0) = x'(0) = 0$$

Transformaremos el PVI en un sistema de ecuaciones diferenciales de primer orden, donde se realiza el siguiente cambio de variable:

$$\left.\begin{aligned} x_1(t) &= x(t) \\ x_2(t) &= x'(t) \end{aligned}\right\} \implies \left\{\begin{aligned} x_1'(t) &= x'(t) = x_2(t) \\ x_2'(t) &= x''(t) = 4\sin(5t) - 25x_1(t) \end{aligned}\right.$$

con $x(0) = x'(0) = 0 \implies x_1(t) = x_2(t) = 0$. Por tanto, escribiremos en Matlab:

```
function F= PVI1(t,X)
    X1 = X(1);
    X2 = X(2);
    F = [X2;4*sin(5*t)-25*X1];
end
```

Hemos de usar el método de Heun de orden 2 para resolver el PVI en el intervalo $[0,2]$ con 40 subintervalos. Representaremos la solución $x(t)$ para $t \in [0,2]$, indicando en una tabla los valores de $x(t)$ para $t \in \{0, 0{,}25, 0{,}5, 0{,}75, 1, 1{,}25, 1{,}5, 1{,}75, 2\}$.

```
a = 0; b = 2; N = 40; Ya = [0;0];
[time, YH] = Heun_sistemas('PVI1', a, b, Ya, N);
% Hacemos la gráfica
```

```
plot(time, Y(:,1), '-')
ylabel('t')
xlabel('x(t)')
title('Heun')
grid on
```

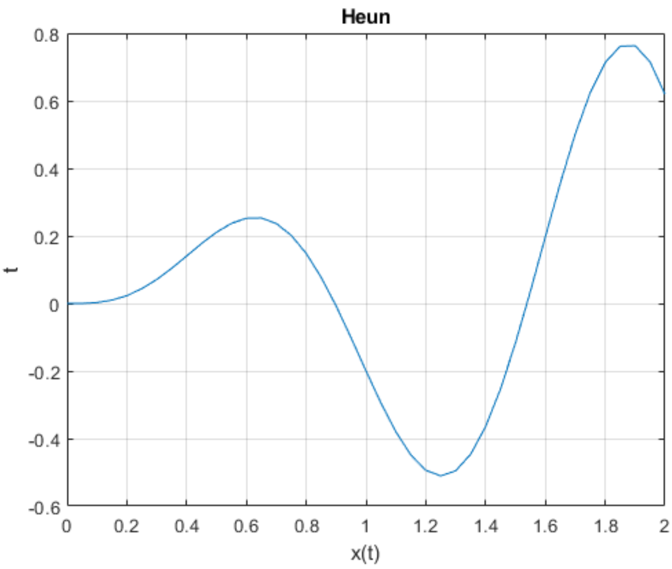

Figura 1: Plot para Heun

```
% Hacemos la tabla resumen
t = zeros(9,1);
for i = 1:length(t)
    t(i)=(i-1)*0.25;
end
T1 = tabla(time,t,YH)
```

```
T1 =

  9×2 table

     t          x_t
    ____    __________

       0             0
    0.25      0.043576
     0.5      0.210805
    0.75      0.201433
       1     -0.199876
    1.25     -0.510669
     1.5      -0.11685
    1.75      0.624296
       2      0.619067
```

Figura 2: Tabla para Heun

Operando de manera análoga para el método de Runge-Kutta, obtenemos el siguiente plot y tabla de resultados:

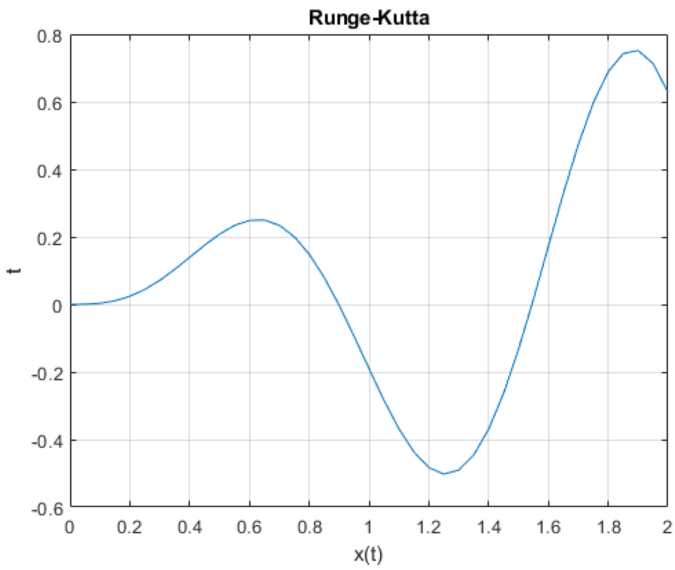

Figura 3: Plot para Runge-Kutta

```
T2 =

  9×2 table

     t          x_t
    ____    _________

       0            0
    0.25      0.04439
     0.5     0.208096
    0.75     0.200436
       1    -0.190148
    1.25    -0.502346
     1.5    -0.132985
    1.75     0.596476
       2     0.627747
```

Figura 4: Tabla para Runge-Kutta

cuyo código de ejecución es:

```
a = 0; b = 2; N = 40; Ya = [0;0];
[time, YR] = Runge_sistemas('PVI1', a, b, Ya, N);
% Hacemos la gráfica
plot(time,Y(:,1),'-')
ylabel('t')
xlabel('x(t)')
title('Runge-Kutta')
grid on
% Hacemos la tabla resumen
t = zeros(2/0.25+1,1);
for i = 1:length(t)
    t(i) = (i-1)*0.25;
end

T2 = tabla(time,t, YR)
```

Proporcionaremos ahora una estimación numérica del orden de los métodos utilizados en los apartados anteriores. Se tomarán los intervalos $N \in \{10, 20, 30, 80, 160\}$

```
n = 5;
N = ones(1,n);
for i = 1:n
    N(i) = (2^(i-1))*10;
end
% Se va a aproximar del orden sin conocer la solución analítica
[e_H,orden_e_H] = eval_fun_N_aprox(@Heun_sistemas, @PVI1, a, b, Ya, N);
[e_R,orden_e_R] = eval_fun_N_aprox(@Runge_sistemas, @PVI1, a, b, Ya, N);
N = N(:);
T = table(N, e_H, orden_e_H, e_R, orden_e_R)
```

T =

5×5 table

N	e_H	orden_e_H	e_R	orden_e_R
10	NaN	NaN	NaN	NaN
20	0.659151	NaN	0.033683	NaN
40	0.173001	1.929827	0.002789	3.594065
80	0.057774	1.582294	0.000238	3.551607
160	0.020093	1.523739	2.1e-05	3.52544

Figura 5: Tabla de estimación del orden

2. Actividad 2

Consideramos el problema del valor inicial

$$y''(x) = \lambda(-y + sin(x))\ y(0) = 0\ 0 < x < 2\pi$$

cuya solución exacta es

$$y(x) = \frac{\lambda}{1+\lambda^2}e^{-\lambda x} + \frac{\lambda^2}{1+\lambda^2}sin(x) - \frac{\lambda}{1+\lambda^2}cos(x)$$

Resolveremos el problema usando el método explícito e impícito de Euler, con $\lambda = 50$ y considerando $N = 32$. Definimos las funciones:

```
function  f = PVI2_explicito(x, y)
    f = 50*(-y+sin(x));
end
function  [f,df] = PVI2_implicito(x, y)
    f = 50*(-y+sin(x));
    df = -50;
end
```

Y procedemos a resolver

```
% Calculo con Euler explicito
a = 0; b = 2*pi; ya = 0; N = 32;
[t1,y1] = Euler('PVI2_explicito', 0, 2*pi, 0, 32);
solex = (50/(1+50^2))*(exp(-50.*t1) + 50*sin(t1)-cos(t1));
error_max_euler = max(abs(y1-solex));
% Plot
plot(t1,y1,'o-')
hold on
plot(t1,solex,'*-')
```

```
xlabel('x'); ylabel('y')
legend('Aproximacion por Euler', 'Valores Exactos')
title('Euler explícito')
hold off
```

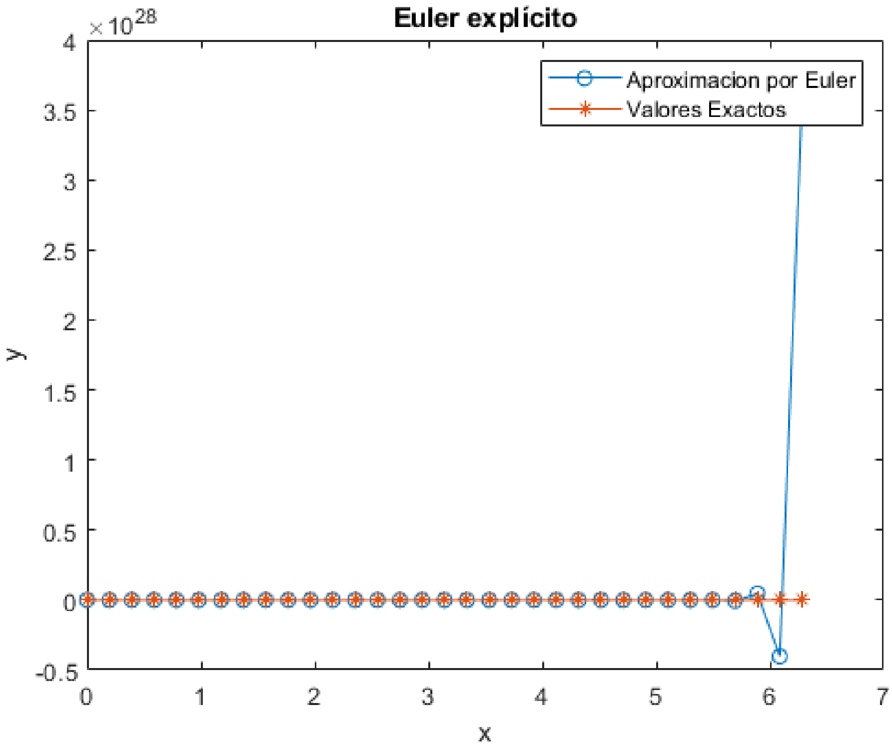

Figura 6: Plot de la aproximación por Euler explícito

con error_max_euler = 3.554546e+28; y para el método implícito

```
% Calculo con Euler implícito
[t2,y2] = Euler_implicito('PVI2_implicito', 0,2*pi,0,32,1e-6,30);
error_max_euler_imp = max(abs(y2-solex));
% Plot
plot(t2,y2,'o-')
hold on
plot(t2,solex,'*-')
```

```
xlabel('x');ylabel('y')
legend('Aproximacion por Euler implicito', 'Valores Exactos')
title('Euler implícito')
hold off
```

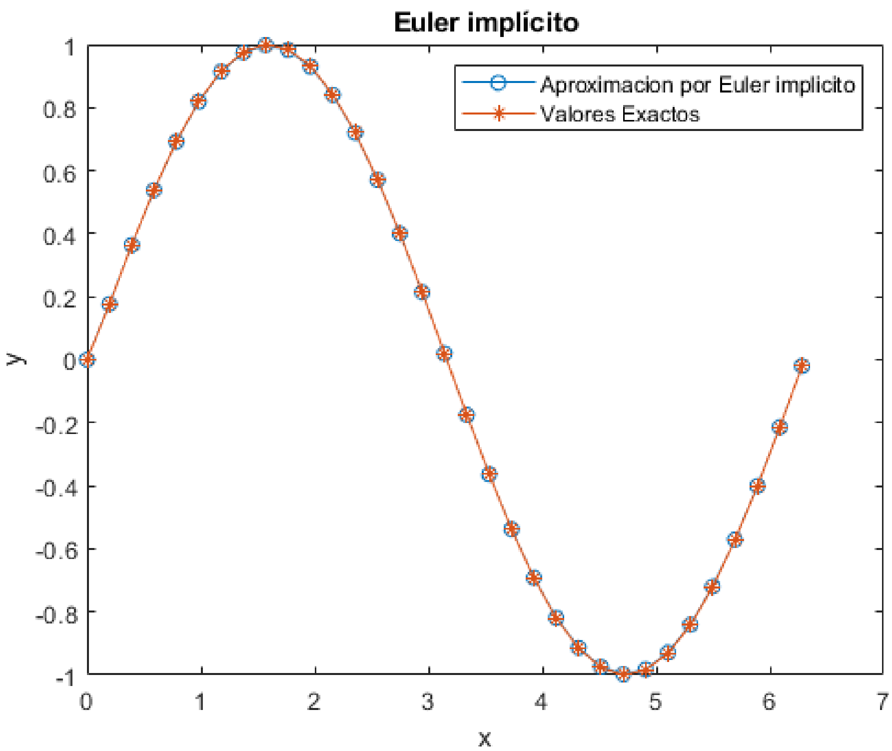

Figura 7: Plot de la aproximación por Euler impícito

con error_max_euler_imp $= 0.001949$.

Como podemos observar, el método explícito lleva a una mejor solución, aunque su coste computacional sea mayor en el mismo número de intervalos, su exactitud lo convierte en una mejor opción en este caso.

Nos preguntamos ahora, ¿Cuántos subintervalos son necesarios para poder asegurar que el método de Euler explícito proporcione una aproximación del problema para $\lambda = 50$?

Probaremos con $N \in \{1, \ldots, 200\}$

```
% Cogiendo N de 0 a 200
```

```
N = linspace(1,200,200)'; %Traspuesta
s = @(x) ((50./(1+50^2)).*(exp(-50.*x)+50*sin(x)-cos(x)));
[E,orden_E] = eval_fun_N_ana(@Euler, s, @PVI2_explicito, a, b, ya, N);
T3 = table(N,E,orden_E);
T3(155:160,:)
```

```
ans =

  6×3 table

     N         E         orden_E
    ___    ________    ________

    155    1.212352    2.791378
    156    0.170761    2.827761
    157    0.023465      2.8634
    158    0.022502    0.060466
    159    0.022286    0.013887
    160    0.022074    0.013819
```

Figura 8: Tabla con algunos intervalos para Euler explicito

Representando algunos de los casos, podemos ver como varía entre 155 y 158

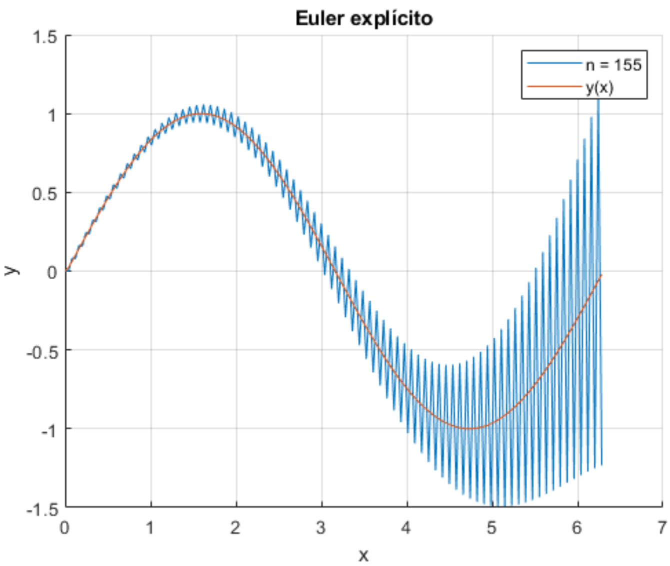

Figura 9: Plot con Euler, N=155

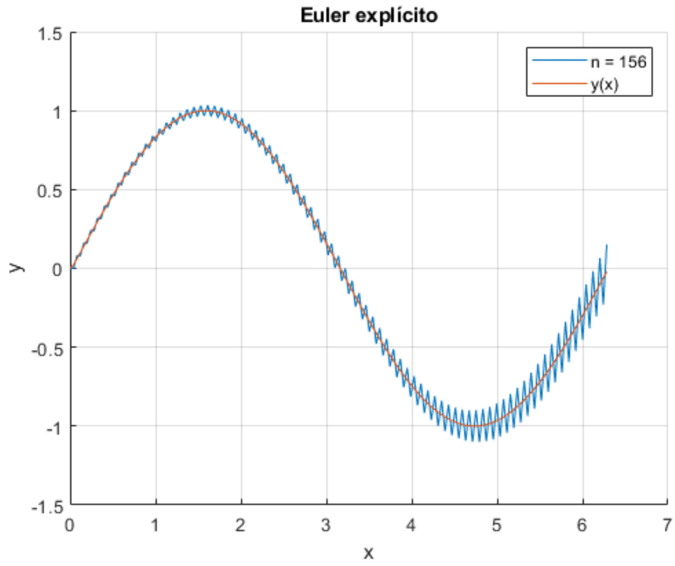

Figura 10: Plot con Euler, N=156

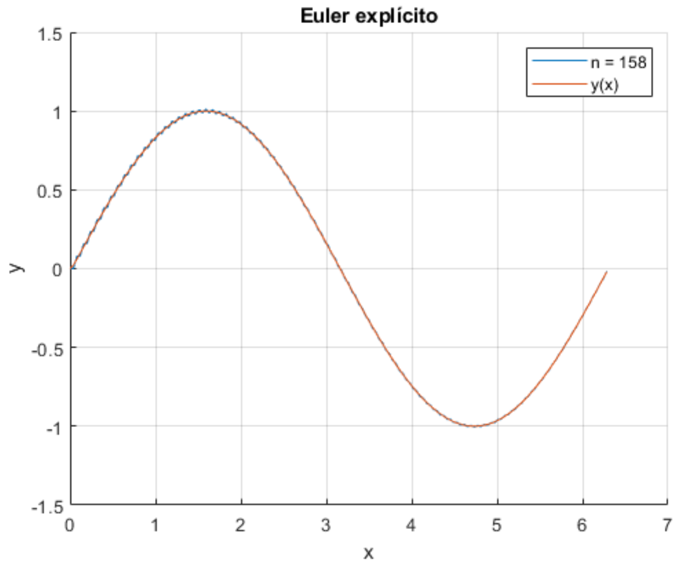

Figura 11: Plot con Euler, N=158

También podriamos haber hecho un bucle *while* poniendo como criterio de parada un error a exigir por nosotros.

```
iter = 1
error = 100 %Aseguramos la entrada
```

```
while  error > 0.02
    [t1,y1] = Euler('PVI2_explicito', 0,2*pi,0,iter);
    solex = (50/(1+50^2))*(exp(-50.*t1) + 50*sin(t1)-cos(t1));
    error  = max(abs(y1-solex))
    iter = iter+1;
end
iter
```

3. Actividad 3

En la última actividad, tenemos la expersión del oscilador de Van der Pol

$$x''(t) - \mu(1 - x^2)x'(t) + x(t) = 0$$

Donde $x(t)$ es la posición x en función del tiempo t y μ representa la amortiguación y no linealidad.

$$\left.\begin{aligned} x_1(t) &= x(t) \\ x_2(t) &= x'(t) \end{aligned}\right\} \implies \left\{\begin{aligned} x_1'(t) &= x'(t) = x_2(t) \\ x_2'(t) &= x''(t) = -x_1(t) + \mu(1 - x_1(t)^2)x_2(t) \end{aligned}\right.$$

con $x(0) = 2,\ x'(0) = 0 \implies x_1(t) = 2,\ x_2(t) = 0$. Resuelve por los métodos de Adams-Bashforth de órdenes 2 y 4 el problema de valor inicial para el caso no amortiguado $(\mu = 0)$ en $t \in [0, 20]$, tomando como valor inicial $x(0) = 2$, $x'(0) = 0$, y como paso $h = 0{,}1$. Representando la evolución de $x(t)$ e indicando en una tabla los valores para $t = \{2, 8, 14, 16\}$. Primero resolveremos

$$x''(t) - +x(t) = 0$$

con el cambio de variable

$$x(t) = x_1(t) \implies x_1'(t) = x_2(t)$$
$$x'(t) = x_2(t) \implies x_12(t) = -x_1(t)$$

Implementamos en Matlab la función a resolver

```
function f = apa(t,x)
    x1 = x(1);
    x2 = x(2);
    f = [x2;-x1];
end
```

y procedemos a resolver el ejercicio

```
a = 0; b = 20; N = 200; Xa = [2, 0];
[t,x] = AB2sist(@apa, a, b, N, Xa)
plot(t, x(:,1), 'o-')
xlabel('t_k');
ylabel('x_k');
title('Método Adam Bashfort Orden 2')
sol = round([x(21), x(81), x(141), x(161)],7)
T = table([2 8 14 16]', sol')
T.Properties.VariableNames = {'t' 'X_t'}
```

que nos devuelve

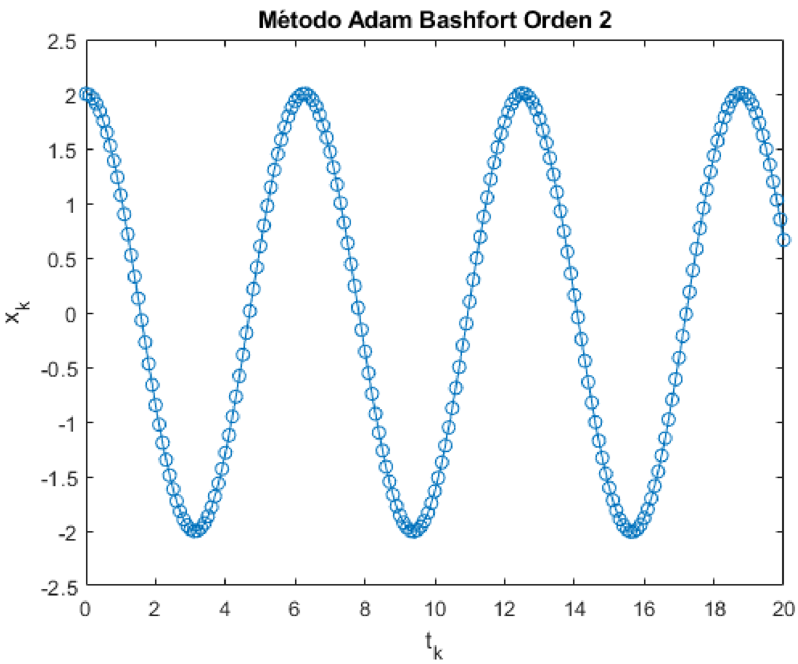

Figura 12: Plot para Adam-Bashfort orden 2

```
T =

  4×2 table

     t         X_t
    ___    ___________

     2     -0.8474831
     8     -0.3574213
    14      0.1578447
    16     -1.8802477
```

Figura 13: Tabla para Adam-Bashfort orden 2

realizamos el mismo procedimiento para el orden 4, obteniendo:

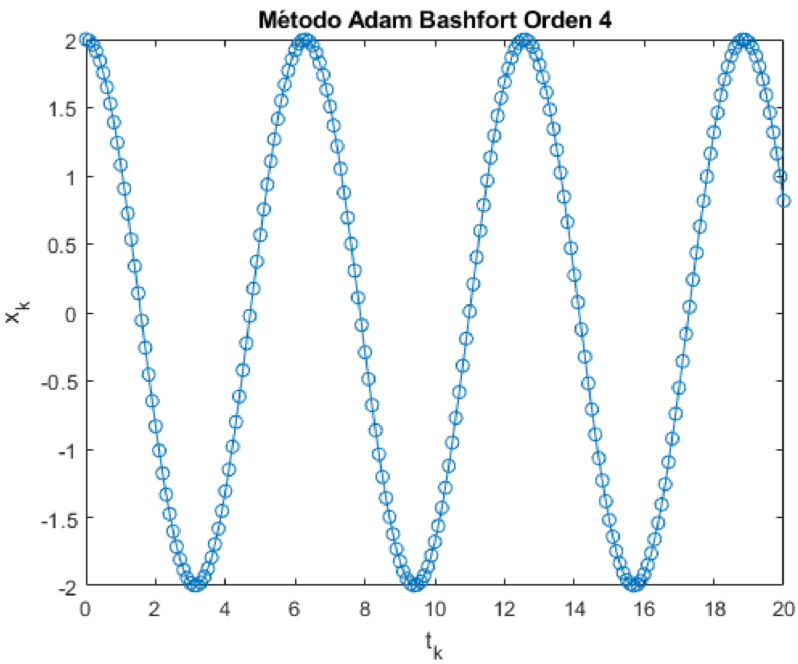

Figura 14: Plot para Adam-Bashfort orden 4

```
T =

  4×2 table

     t         X_t
    ___    __________

     2     -0.8321792
     8     -0.2904634
    14      0.2743883
    16     -1.9154677
```

Figura 15: Tabla para Adam-Bashfort orden 4

Resolveremos ahora el mismo problema, pero tomando $\mu = 1$. Implementamos en Matlab la función a resolver

```
function f = apb(t,x)
    x1 = x(1);
    x2 = x(2);
    f = [x2;((1-x1.^2).*x2)-x1];
end
```

y procedemos a resolver el ejercicio

```
a = 0; b = 20; N = 200; Xa = [2, 0];
[t,x] = AB2sist(@apb, a, b, N, Xa)
plot(t, x(:,1), 'o-')
xlabel('t_k');
ylabel('x_k');
title('Método Adam Bashfort Orden 2')
sol = round([x(21), x(81), x(141), x(161)],7)
T = table([2 8 14 16]', sol')
T.Properties.VariableNames = {'t' 'X_t'}
```

que nos devuelve

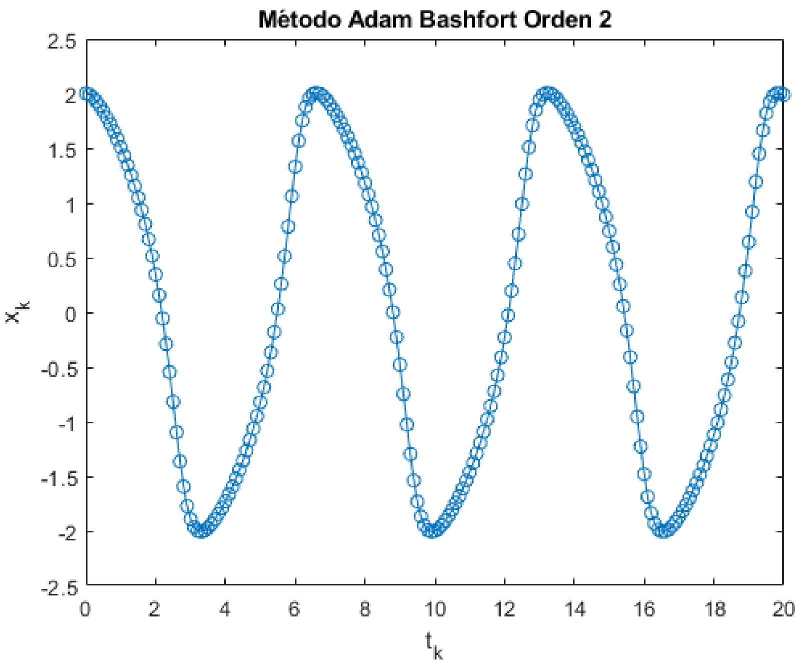

Figura 16: Plot para Adam-Bashfort orden 2

```
T =

  4×2 table

    t        X_t
    __    __________

     2     0.3444267
     8     1.1813699
    14     1.6894306
    16    -1.4802526
```

Figura 17: Tabla para Adam-Bashfort orden 2

realizamos el mismo procedimiento para el orden 4, obteniendo:

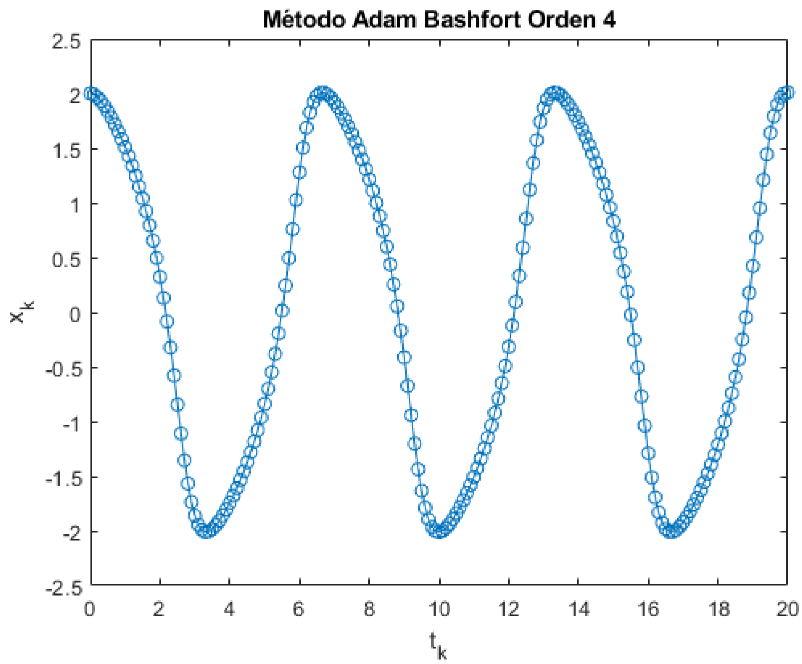

Figura 18: Plot para Adam-Bashfort orden 4

```
T =

  4×2 table

    t         X_t
    __    __________

     2     0.3234019
     8     1.2140095
    14     1.7396691
    16    -1.2894213
```

Figura 19: Tabla para Adam-Bashfort orden 4

Como parte final del ejercicio, realizaremos una estimación del orden de convergencia de ambos métodos para el caso no amortiguado. Para calcular el error en el caso amortiguado y el orden de convergencia, aprovechamos que podemos resolver fácilmente la solución exacta resolviendo analíticamente el PVI, que consiste en una ecuación diferencial homogénea lineal de segundo grado, cuya solución es:

$$x(t) = C_1cos(t) + C_2sin(t)$$

Aplicando las condiciones iniciales:

$$x(0) = 2 = C_1$$

$$x'(0) = 0 = C_2$$

Obtenemos la solución exacta del PVI

$$x(t) = 2cos(t)$$

que utilizaremos para calcular el error exacto, con el siguiente código:

```
% Para AB2
solex200 = 2*cos(t);
[t,y] = AB2sist(@apa, a, b, N, Xa)
E200 = max(abs(solex200-y));
% Error
E200(1)
% Orden de convergencia
N = 400;
[t400,y400] = AB2sist('am',a,b,N,ya)
solex400 = 2*cos(t400);
E400 = max(abs(solex400-y400))
%Error
E400(1)
convergencia = log2(E200(1)/E400(1))
```

de donde obtenemos:

```
error_200 =

                     0.1519411


error_400 =

                     0.0377275


convergencia =

                     2.0098242
```

Figura 20: Convergencia para orden 2

Utilizando las mismas insrtucciónes para la aproximación de orden 4, obtendremos:

```
error_200 =

                     0.0011956


error_400 =

                       7.6e-05


convergencia =

                      3.975946
```

Figura 21: Convergencia para orden 4

Tal y como podíamos esperar, el método tiene un orden de convergencia de 4 y los errores cometidos son mucho menores que los errores del mismo método de orden 2.

4. Anexos: Código

4.1. Funciones auxiliares

Función que devuelve la tabla con ciertas restricciones.

```
function [T]=tabla(time,t,f)
    f = f(:);
    n = length(t);
    x_t=zeros(n,1);
    j = 1;
    for i = 1:length(time)
        if t(j,1) == time(i,1)
            x_t(j,1)=f(i);
            j=j+1;
        end
    end
    x_t = round(x_t,6);
    T = table(t,x_t);
end
```

Función para aproximar el orden cuando conocemos la función analítica.

```
function [E,orden_E] = eval_fun_N_ana(f,solex,PVI,a,b,ya,N)
    n = length(N);
    E = zeros(1,n);
    for i = 1:n
        [t,y] = f(PVI,a,b,ya,N(i));
        E(i) = max(abs(solex(t)-y));
    end
    orden_E = log2(E(1:end-1)./E(2:end));
```

```
    E = round(E,6);
    E = E(:);
    orden_E = [missing orden_E]';
    orden_E = round(orden_E,6);
end
```

Función para aproximar el orden cuando no conocemos la función analítica.

```
function [e,orden_e]=eval_fun_N_aprox(f,PVI,a,b,Ya,N)
    n = length(N);
    e = zeros(1,n-1);
    for i = 1:n
        [~,Y] = f(PVI,a,b,Ya,N(i));
        if i < n
            [~,Y2] = f(PVI,a,b,Ya,N(i+1));
            e(i) = norm(Y(:,1)-Y2(1:2:end,1));
        end
    end
    orden_e = log2(e(1:end-1)./e(2:end));
    e = round(e,6);
    orden_e = round(orden_e,6);
    e = [missing e]';
    orden_e = [missing missing orden_e]';
end
```

4.2. Ejercicio 2

```
function [t, y] = Euler(f,a,b,ya,N)
    h = (b-a)/N;
    t = a:h:b;
    t = t(:);
    y = zeros(N+1,1);
    y(1) = ya;
    for k = 1:N
        fx = feval(f, t(k), y(k));
        y(k+1) = y(k)+h*fx;
    end
end

function [t,y] = Euler_implicito(f,a,b,ya,N,tol,maxiter)
    h = (b-a)/N;
    t = a:h:b;
    t = t(:);
    y = zeros(N+1, 1);
    y(1) = ya;
    for k = 1:N
        x0 = y(k);
        iter = 1;
        dif = tol+1;
        while iter < maxiter && dif > tol
            [fx0, dfx0] = feval(f, t(k+1), x0);
            g = x0-y(k)-h*fx0;
            dg = 1-h*dfx0;
            x1 = x0-g/dg;
            dif = abs(x1-x0);
```

```
            iter = iter+1;
            x0 = x1;
        end
        y(k+1) = y(k)+h*fx0;
    end
end
```

4.3. Ejercicio 3

AB para orden 2

```
function [t,y]=AB2sist(fun,a,b,N,Ya)
    h = (b-a)/N;
    t = a:h:b;
    t = t(:); %Discretizamos t con el paso correcto
    n = length(Ya); %Damos el número de columnas en función de las variables
    y = zeros(N+1,n); %Creamos la matriz solución
    y(1,:) = Ya; %Inicializamos la matriz y

    k1 = h*feval(fun,t(1),y(1,:))'; %Empleamos el método de Heun para sacar el primer p
    k2 = h*feval(fun,t(2),y(1,:)+k1)';
    y(2,:) = y(1,:)+k1/2+k2/2;

    for k = 2:N
        k1 = (h/2)*feval(fun,t(k),y(k,:))'; %Empleamos el algoritmo de AB2 para los dem
        k2 = (h/2)*feval(fun,t(k-1,:),y(k-1,:))';
        y(k+1,:) = y(k,:)+3*k1-k2;
    end
end
```

AB para orden 4

```
function [t,y]=AB4sist(fun,a,b,N,Ya)
    h = (b-a)/N;
    t = a:h:b;
    t = t(:); %Discretizamos t con el paso correcto
    n = length(Ya); %Damos el número de columnas en función de las variables
    y = zeros(N+1,n); %Creamos la matriz solución
```

```
    y(1,:) = Ya; %Inicializamos la matriz y

    for k = 1:3 %Empleamos el método de Runge-Kutta para obtener los 4 primeros puntos
        k1 = feval(fun,t(k),y(k,:))';
        k2 = feval(fun,t(k)+(h/2),y(k,:)+(h/2)*k1)';
        k3 = feval(fun,t(k)+(h/2),y(k,:)+(h/2)*k2)';
        k4 = feval(fun,t(k+1),y(k,:)+h*k3)';
        y(k+1,:) = y(k,:)+(h/6)*(k1+2*k2+2*k3+k4);
    end

    for k = 4:N %Empleamos el algoritmo de AB4 para los demás puntos
        y(k+1,:) = y(k,:)+(h/24)*(55*feval(fun,t(k),y(k,:))' - 59*feval(fun,t(k-1),y(k-1
    end

end
```

4.4. Código Varias Variables Trapecios

El código tiene la muletilla de *optimizadas* pues en un primer lugar, usaba dos bucles *for* para iterar por los distintos $f(x_i, y_k)$ haciéndolo mucho menos eficiente, esto se puede evitar utilizando la traspuesta de y y tomándola de menor a mayor.

```
function I = trapecios_varias_variables_optimizadas(f, a, b, c, d, n, m)
    % Devuelve la integral de la funcion F
    % [a,b] = limites para x
    % [c,d] = limites para y
    % n = intervalos para x
    % n = intervalos para y
    h = (b-a)/n; %calculo paso h
    k = (d-c)/m; %calculo paso h
    x = a:h:b; % calculo los pasos de a b
    y = d:-k:c; % calculo los pasos de c d
    % Crearé la matriz de pesos
    pesos_x = [1 2*ones(1, n-1) 1]; %Matriz fila
    pesos_y = [1 2*ones(1, m-1) 1]'; %Matriz columna
    pesos = pesos_y*pesos_x; %Matriz de pesos
    matriz_valores = f(x,y');
    % Tendré 1f(x0,ym) 2f(x2,ym)....1f(xn,ym)
    % ....   2f(x0,ym-1) 4(x1,ym-2)....
    % Para finalmente sumar todos los valores
    I = ((h*k)/4)*sum(sum(pesos.*matriz_valores));
    %I = ((h*k)/4)*sum(pesos.*matriz_valores, 'all'); %añado el . para multiplicar vec
end
```