Asignatura	Datos del alumno	Fecha
Métodos Numéricos II	**Apellidos:** Avilés Cahill	25/04/2022
	Nombre: Adán	

Laboratorio: Problemas de contorno Unidimensionales, métodos de disparo.

1. Introducción

Con esta actividad vas a conseguir poner en práctica los conceptos relacionados con los problemas de frontera unidimensionales estudiados en la asignatura. Concretamente, se aplicará el método de disparo con secante y con Newton a un problema de frontera cuyas condiciones no son de tipo Dirichlet.

1.1. Descripción:

Consideramos el siguiente problema de frontera:

$$\begin{aligned} y'' - xyy' + xcos(x)y + sin(x) = 0,\ x \in [0, \pi] \\ y(0) + y'(0) = 1,\ y(\pi) - 2y'(\pi) = 2, \end{aligned} \tag{1}$$

Vamos a resolverlo utilizando el método de disparo con secante y con Newton.

2. Actividades

2.1. Método de disparo, secante

En primer lugar, aplicaremos el método de disparo utilizando el método de la secante.Para realizar esto, vamos a utilizar el PVI:

$$\begin{aligned} y'' - xyy' + xcos(x)y + sin(x) = 0,\ x \in [0, \pi] \\ y(0) = t,\ y'(0) = 1 - t \end{aligned} \tag{2}$$

Cuya solución $y(t,x)$ queremos que satisfaga

$$|y(t,\pi) - 2y'(t,\pi) - 2| < 10^{-8} \tag{3}$$

Exigimos las condiciones iniciales a 2 para que $y(0) + y'(0) = t + 1 - t = 1$, cumple la condición de frontera del problema inicial. Además, con el método de la secante se iterará buscando un t que cumpla 3 y así habremos aproximado la solución del problema de contorno. Despejamos y''

$$y'' = f(x, y, y') = xyy' - xcos(x)y - sin(x)$$

Sean ahora

$$\left.\begin{array}{l} y_1 = y \\ y_2 = y' \end{array}\right\} \implies \begin{array}{ll} y_1' = y_2 & \\ & , \\ y_2' = xy_1y_2 - xcos(x)y_1 - sin(x) & \end{array} \left.\begin{array}{l} y_1(0) = t \\ y_2(0) = 1 - t \end{array}\right\}$$

Definimos además

$$F(t_k) := y(t_k,\pi) - 2y'(t_k,\pi) - 2 \tag{4}$$

Por el método de la secante, sabemos que

$$t_{k+1} = t_k - \frac{F(t_k)(t_k - t_{k-1})}{F(t_k) - F(t_{k-1})}$$

así que sustituyendo 4 en el método anterior, tenemos:

$$t_{k+1} = t_k - \frac{(y(t_k,\pi) - 2y'(t_k,\pi) - 2)(t_k - t_{k-1})}{y(t_k,\pi) - 2y'(t_k,\pi) - y(t_{k-1},\pi) + 2y'(t_{k-1},\pi)}$$

Implementamos el PVI en matlab.

```
function dy = ejercicio_practica(x,y)
dy=[y(2) ; x*y(1)*y(2)-x.*cos(x)*y(1)-sin(r)];
```

```
end
```

Implementamos también el método de disparo adaptado:

```
function [x, Y, t, iter, incre] = ...
        DisparoSecantePractica(f, a, b, alfa,beta ,n , tol, maxiter)
    h=(b-a)/(n) ; x=a:h:b; x=x(:); %el vector x tiene n+2 componentes
    t0 = 0; %tomo t inicial
    [x, Y] = ode45(f, x, [t0, alfa-t0]'); %calculo y(x) para el pvi pedido
    ub0 = Y(end, 1) ; %tomo el ultimo valor, y(t0)
    upb0 = Y(end, 2); %tomo la derivada
    t1 = 1;
    [x, Y]= ode45(f, x, [t1 ,alfa-t1]');
    ub1 = Y(end, 1) ; %tomo el ultimo valor, y(t0)
    upb1 = Y(end, 2); %tomo la derivada
    iter = 2;
    incre  = abs(ub1-2*upb1-beta);
    while incre > tol && iter <maxiter
        t = t1-((t1-t0)*(ub1-2*upb1-beta))./(ub1-2*upb1-ub0+2*upb0);
        [x, Y]= ode45(f, x, [t ,alfa-t]');
        t0 = t1; t1 = t;
        ub0 = ub1; upb0 = upb1;
        ub1 = Y(end, 1); upb1 = Y(end, 2) ;
        incre  = abs(ub1-2*upb1-beta); %necesitoque el esto sea menor a la tolerancia
        iter= iter+1;
    end
    if incre > tol
        disp('se necesitan mas iteraciones')
    end
end
```

Ejecutaremos el código con:

```
[x, Y, t, iter, incre] = ...
DisparoSecantePractica('ejercicio_practica', 0, pi, 1, 2, 10, 1e-8, 100)
```

obteniendo así la solución aproximada:

```
solaprox_secante =

  -0.000001890486749   1.000001890486749
   0.309015638419666   0.951058551004207
   0.587784223712039   0.809019026418072
   0.809016084904228   0.587786554919273
   0.951055908061074   0.309017360198323
   1.000000273700096   0.000001129819622
   0.951058403273128  -0.309012560037805
   0.809021158881650  -0.587776422299194
   0.587791950360454  -0.809005272470042
   0.309025762268878  -0.951045290821290
   0.000011320190490  -0.999994339905795
```

Figura 1: Valores aproximados

Que tenemos también en formato tabla en 4.1 veamos también el número de iteraciones y el t_k obtenido:

```
>> t_secante

t_secante =

    -1.890486749205626e-06

>> iter_secante

iter_secante =

     7
```

Figura 2: Número de iteraciones y t_k

cuya gráfica es

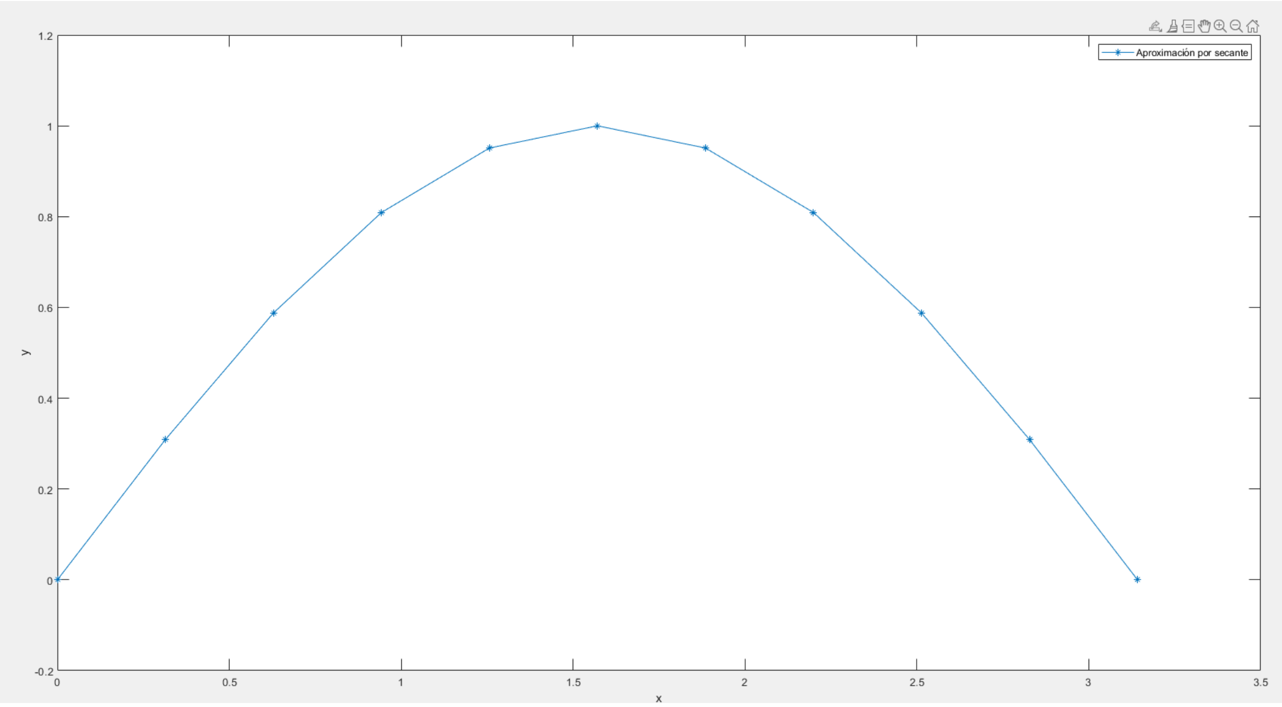

Figura 3: Gráfica de la aproximación usando secante.

2.2. Método de disparo, Newton

En este caso, hemos de encontrar la aproximación de t_k, que sea solución de 4 utilizando el método de Newton. Planteamos entonces:

$$t_{k+1} = t_k - \frac{F(t_k)}{F'(t_k)} = t_k - \frac{y(t_k, \pi) - 2y'(t_k, \pi) - 2}{\frac{\partial}{\partial t} y(t_k, \pi) - 2\frac{\partial}{\partial t} y'(t_k, \pi)}$$

Como no tenemos una expresión analítica para $y(t, x)$, definiremos $z(t, x) = \frac{\partial}{\partial t} y(t_k, x)$, entonces la expresión anterior, queda definida como

$$t_{k+1} = t_k - \frac{y(t_k, \pi) - 2y'(t_k, \pi) - 2}{z(t_k, x) - 2z'(t_k, x)}$$

Tal y como se ha visto en clase, para encontrar z, hemos de resolver el siguiente problema:

$$z'' = \frac{\partial f}{\partial y} z + \frac{\partial f}{\partial y'} z', \quad z(t, 0) = \frac{\partial}{\partial t} y(t_k, 0) = 1, \quad z'(t, 0) = \frac{\partial}{\partial t} y'(t_k, 0) = -1$$

Encontremos en primer lugar las parciales de f, que habíamos despejado ya en el método de la secante.

$$\begin{aligned} \frac{\partial f}{\partial y} &= xyz - xcos(x)z + xyz' \\ \frac{\partial f}{\partial y'} &= xy \end{aligned} \tag{5}$$

Por tanto,

$$z'' = xy'z - xcos(x)z + xyz'$$

Con estos datos en mano, definiremos entonces el sistema a resolver:

$$\left.\begin{aligned} y_1 &= y \\ y_2 &= y' \\ y_3 &= z \\ y_4 &= xy_2y_3 - xcos(x)y_3 + xy_1y_4 \end{aligned}\right\} \implies \left.\begin{aligned} y_1 &= y_2 \\ y_2 &= xy_1y_2 - xcos(x)y_1 - sin(x) \\ y_3 &= y_4 \\ y_4 &= xy_1y_2 - xcos(x)y_1 - sin(x) \end{aligned} \quad , \quad \begin{aligned} y_1(0) &= t \\ y_2(0) &= 1-t \\ y_3(0) &= 1 \\ y_4(0) &= -1 \end{aligned}\right\}$$

Que implementaremos en Matlab

```
function dy = ejercicio_practica_newton(x,y)
dy=[y(2);
    x.*y(1)*y(2)-x.*cos(x)*y(1)-sin(x);
    y(4);
    x.*y(2)*y(3)-x.*cos(x)*y(3)+x.*y(1)*y(4);
   ];
end
```

Junto al método de disparo adaptado a Newton

```
function [nodos, solaprox, t, iter] =...
    DisparoNewtonPractica(f, a,b ,alfa, beta ,n , tol, maxiter)
```

```
    h=(b-a)/(n) ; x=a:h:b; x=x(:); %el vector x tiene n+2 componentes
    t0 = 0; %tomo t inicial
    [x, Y] = ode45(f, x, [t0, alfa-t0, 1, -1]'); %calculo y(x) para el pvi pedido
    yb1 = Y(end, 1); yb2 = Y(end, 2);
    zb1 = Y(end, 3); zb2 = Y(end, 4);
    iter = 1;
    incre = tol +1;
    while incre > tol && iter <maxiter
        t = t0-(yb1-2*yb2-beta)/(zb1-2*zb2);
        [x, Y] = ode45(f, x, [t, alfa-t, 1, -1]');
        t0 = t;
        yb1 = Y(end, 1); yb2 = Y(end, 2);
        zb1 = Y(end, 3); zb2 = Y(end, 4);
        %una vez he actualizado los valores, calculo incre
        incre = abs(yb1-2*yb2-beta);
        iter= iter+1;
    end
    if incre <= tol
        nodos=x;
        solaprox=Y;
    else
        disp('se necesitan mas iteraciones')
    end
end
```

Ejecutando el código como en el caso anterior, obtenemos los los siguientes resultados:

```
solaprox_newton =

  -0.000001890651322   1.000001890651322   1.000000000000000  -1.000000000000000
   0.309015638306928   0.951058551170473   0.685032763534710  -1.010286315788151
   0.587784223653000   0.809019026596256   0.358739003974427  -1.082708155926336
   0.809016084905754   0.587786555131658  -0.009273571689806  -1.290517847855979
   0.951055908139845   0.309017360487226  -0.478645776496755  -1.755472945894445
   1.000000273891600   0.000001130266983  -1.163647243817050  -2.718318141485182
   0.951058403649264  -0.309012559275063  -2.285532315206288  -4.634677595051508
   0.809021159580553  -0.587776420953029  -4.246768263650964  -8.179742248331138
   0.587791951618295  -0.809005270207018  -7.643059569941641 -13.750885817557659
   0.309025764404038  -0.951045287522051 -12.973947484575090 -20.047280041687362
   0.000011323466073  -0.999994336097312 -19.903538985870068 -23.141613137356121
```

Figura 4: Valores aproximados

que tendremos en formato tabla en 4.2. Tenemos también el número de iteraciones y el t_k obtenido:

```
>> t_newton

t_newton =

    -1.890651322468298e-06

>> iter_newton

iter_newton =

     2
```

Figura 5: Número de iteraciones y t_k

y podemos representar gráficamente:

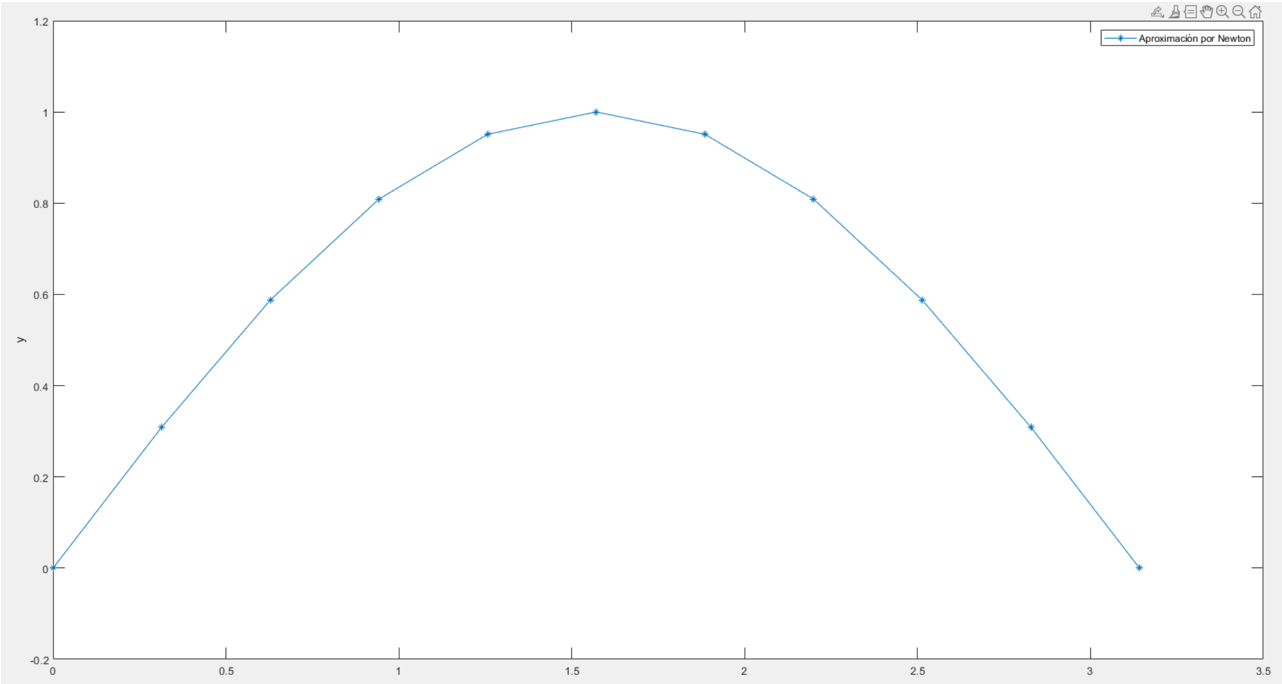

Figura 6: Gráfica de la aproximación usando Newton.

Cabe mencionar, que los resultados van a depender mucho del t_0 elegido. Si tomamos $t_0 = 2504$ vemos que no llega a aproximar la solución, pues no converge.

```
>> iter_newton

iter_newton =

    11

>> solaprox_newton

solaprox_newton =

   0.068408583726416   0.931591416273584   1.000000000000000  -1.000000000000000
   0.355858573281694   0.881759646655931   0.684432413095141  -1.015686627647757
   0.612188315194653   0.734372225167250   0.354708968538356  -1.099634539283744
   0.808001228073841   0.498566706403583  -0.020427318312679  -1.317857791699733
   0.917657697711728   0.188371811011709  -0.497752737117258  -1.771285129238820
   0.920045856054974  -0.182231797144263  -1.173612798731205  -2.608612613277375
   0.798080670589029  -0.600959664702187  -2.187726830513882  -3.915776325315786
```

Figura 7: No convergencia para 2504

3. Comparación

En primer lugar, veamos una comparación de los resultados obtenidos con cade método, haciendo la diferencia absoluta:

```
>> diferencia_valores

diferencia_valores =

   1.0e-08 *

   0.016457326267163
   0.011273815214707
   0.005903899591431
   0.000152600154735
   0.007877176688709
   0.019150458996364
   0.037613645531565
   0.069890293552533
   0.125784060944767
   0.213516027081084
   0.327558311030707
```

Figura 8: Diferencia entre valores obtenidos

Si pintamos por pantalla el resultado de ambas, vemos que es muy similar. (Para pintar la aproximación de la función hemos usado $n = 1000$ en el método de Newton)

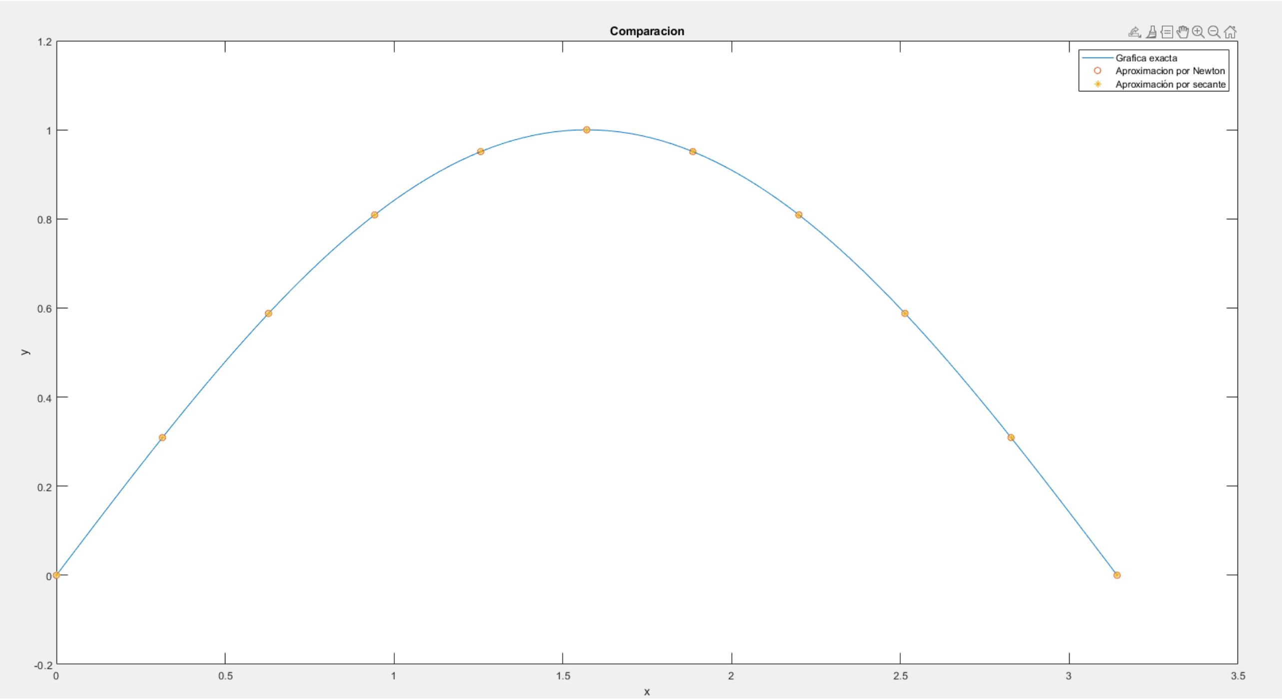

Figura 9: Gráfica comparativa

Pese a que los resultados son muy similares, podemos observar que el número de iteraciones necesarias para Newton es inferior que para el método de la secante, lo cual es una ventaja para la velocidad de computación.

4. Anexo

4.1. Tabla de resultados para la secante.

Y	Y'
-0.000001890486749	1.000001890486749
0.309015638419666	0.951058551004207
0.587784223712039	0.809019026418072
0.809016084904228	0.587786554919273
0.951055908061074	0.309017360198323
1.000000273700096	0.000001129819622
0.951058403273128	-0.309012560037805
0.809021158881650	-0.587776422299194
0.587791950360454	-0.809005272470042
0.309025762268878	-0.951045290821290
0.000011320190490	-0.999994339905795

4.2. Tabla de resultados para el método de Newton.

Y	Y'	z	z'
-0.000001890651322	1.000001890651322	1.000000000000000	-1.00000000000000
0.309015638306928	0.951058551170473	0.685032763534710	-1.010286315788151
0.587784223653000	0.809019026596256	0.358739003974427	-1.082708155926336
0.809016084905754	0.587786555131658	-0.009273571689806	-1.290517847855979
0.951055908139845	0.309017360487226	-0.478645776496755	-1.755472945894445
1.000000273891600	0.000001130266983	-1.163647243817050	-2.718318141485182
0.951058403649264	-0.309012559275063	-2.285532315206288	-4.634677595051508
0.809021159580553	-0.587776420953029	-4.246768263650964	-8.179742248331138
0.587791951618295	-0.809005270207018	-7.643059569941641	-13.750885817557659
0.309025764404038	-0.951045287522051	-12.973947484575090	-20.047280041687362
0.000011323466073	-0.999994336097312	-19.903538985870068	-23.141613137356121