Manual de métodos numéricos para EDO's y EDP's

Rodolfo M. Turpo R.

08 de agosto de 2025

Índice

1. Gráfica de Solución de una EDO

El objetivo de este informe es mostrar cómo graficar la solución de una ecuación diferencial ordinaria (EDO) utilizando Matlab/Octave. Para ello, se resolverá una EDO específica y se graficará su solución junto con algunos puntos evaluados en la misma.

Problema a Resolver:

$$y' + 2y = x \quad , y(0) = 0 \quad x \in [0, 1] \tag{1}$$

La solución de la ecuación diferencial es

$$y = \frac{2x - 1}{4} + \frac{e^{-2x}}{4} \tag{2}$$

Queremos encontrar los valores en $y(1), y(1/2), y(\pi/4)$, de ahí, graficar junto a la solución en **Matlab/octave**. Para ese creamos el siguiente programa:

```
y = @(x) (2*x-1)/4+exp(-2*x)/4; %definimos la función solución de la EDO
% evaluamos la función en los puntos deseados
y(1)
y(1/2)
y(pi/4)
```

Guardamos el programa con el nombre de su preferencia y lo ejecutamos, nos da el siguiente resultado en consola.

```
ans = 0.2838
ans = 0.091970
ans = 0.1947
```

Graficamos la solución en el intervalo $[0, 1]$ con los puntos calculados, para eso abajo del codigo del último programa escribimos estas dos lineas de codigo, las cuales al ejecutar el programa nos devolvera la siguiente gráfica.

```
x=0:0.01:1;
plot(x,y(x),1,y(1),'ro',1/2,y(1/2),'ro',pi/4,y(pi/4),'ro')
```

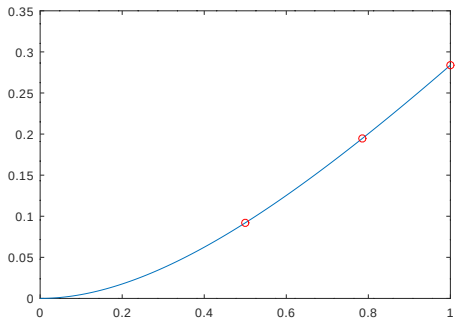


Figura 1: Gráfica de la solución

2. Método de Euler

Es un método numérico para resolver una edo de la forma

$$y' = f(t, y), \quad y(t_0) = y_0 \quad a \leq t \leq b$$

esta dado en términos matemáticos

$$y_{i+1} = y_i + h \cdot f(x_i, y_i) \tag{3}$$

donde

- y_i es el valor de la función en el punto x_i.
- h es el tamaño del paso, que se define como $h = (b - a)/N$, donde N es el número de pasos.
- $f(x_i, y_i)$ es la función que define la ecuación diferencial evaluada en el punto (x_i, y_i).
- x_i es el valor de la variable independiente en el paso i, que se define como $x_i = a + i \cdot h$.
- y_{i+1} es el valor de la función en el siguiente paso, que se calcula a partir del valor actual y_i y la pendiente dada por $f(x_i, y_i)$.
- N es el número de pasos que se desea realizar en el intervalo $[a, b]$.
- a es el límite inferior del intervalo.
- b es el límite superior del intervalo.
- t_0 es el valor inicial de la variable independiente.
- y_0 es el valor inicial de la función en el punto t_0.

Esta formula se conoce como **método de Euler**(o de *Euler-Cauchy* o de *Punto pendiente*)

Ejemplo: Resolver la EDO con el método

$$y' = y^{1,5} + 1, \quad y(0) = 10 \text{ con } 0 \leq t \leq 1 \tag{4}$$

Con $h = 1/2$, $h = 1/4$, $h = 0{,}1$
Solución
Creamos un script en *Matlab/Octave* para resolver la ecuación diferencial con nombre **euler.m**.
Código en Matlab/Octave:

```
function [t,w] = euler(a,b,N,alpha,f)
 h=(b-a)/N;
 t = zeros(N+1,1); % Para guardar los valores de t
 w = zeros(N+1,1); % Para guardar los valores de w
 t(1)=a;
 w(1)=alpha;
 for i=1:N
  w(i+1)=w(i)+h*f(t(i),w(i));
  t(i+1)=a+i*h;
 end
end
```

Luego para ejecutar el script, creamos otro script llamado **run_euler.m** que llama a la función **euler.m** y grafica los resultados. Como se quiere para un $h = 1/2$, entonces $N = 2$, esto se obtiene resolviendo la formula $h = (b - a)/N$.

```
f = @(t, y) y^(3/2) + 1; %función de la EDO
a=0;b=1; %intervalo
N=2; %número de pasos
y0=10; %condición inicial
%-----------------------------
[T1, W1] = euler(a, b, N, y0, f)
plot(T1, W1, 'ro-');
grid on;
```

Manual:

- Abrir el archivo **run_euler.m** en Matlab/Octave.
- En las lineas

```
f = @(t, y) y^(3/2) + 1; %función de la EDO
a=0;b=1; %intervalo
N=2; %número de pasos
y0=10; %condición inicial
```

Se ingresa la ecuación diferencial ordinaria a solucionar, la condición inicial y el intervalo de solución.

- En la línea **N=2;** se define el número de pasos a realizar.
- Ejecutar el script con el comando **run_euler** en la consola de Matlab/Octave.

Corrida del Programa:

```
T1 =

    0
0.5000
1.0000

W1 =

10.000
26.311
94.293
```

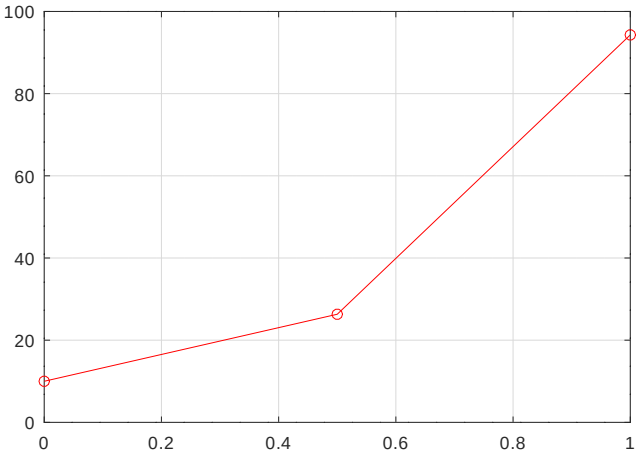


Figura 2: Gráfica de la solución con $h = 1/2$

Para $h = 1/4$, entonces $N = 4$.

```
f = @(t, y) y^(3/2) + 1; %función de la EDO
a=0;b=1; %intervalo
N=4; %número de pasos
y0=10; %condición inicial
%-----------------------------
[T2, W2] = euler(a, b, N, y0, f)
plot(T2, W2, 'ro-');
grid on;
```

Corrida del Programa:

```
T2 =

    0
0.2500
0.5000
0.7500
1.0000

W2 =

10.000
18.156
37.746
95.971
331.266
```

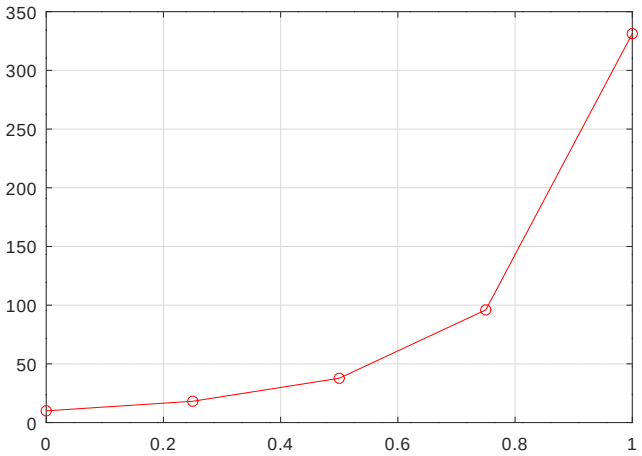


Figura 3: Gráfica de la solución con $h = 1/4$

Para $h = 0{,}1$, entonces $N = 10$.

```
f = @(t, y) y^(3/2) + 1; %función de la EDO
a=0;b=1; %intervalo
N=10; %número de pasos
y0=10; %condición inicial
%-----------------------------
[T3, W3] = euler(a, b, N, y0, f)
plot(T3, W3, 'ro-');
```

```
grid on;
```

Corrida del Programa:

```
T3 =

         0
    0.1000
    0.2000
    0.3000
    0.4000
    0.5000
    0.6000
    0.7000
    0.8000
    0.9000
    1.0000

W3 =

    1.0000e+01
    1.3262e+01
    1.8192e+01
    2.6051e+01
    3.9448e+01
    6.4325e+01
    1.1601e+02
    2.4107e+02
    6.1548e+02
    2.1425e+03
    1.2060e+04
```

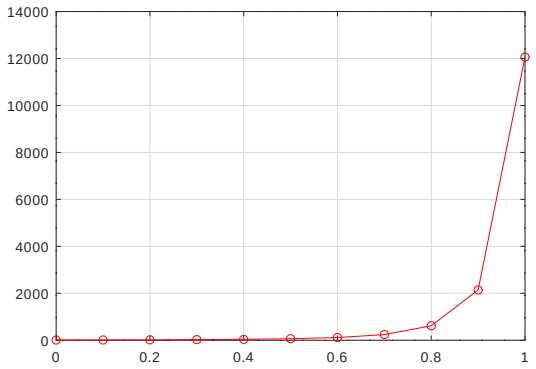


Figura 4: Gráfica de la solución con $h = 0{,}1$

3. Cotas del error de Euler

Es un programa que calcula las cotas del error del método de Euler para una ecuación diferencial ordinaria (EDO) dada. El método de Euler es un método numérico para resolver EDOs, y las cotas del error proporcionan una estimación de la precisión de la solución aproximada en comparación con la solución exacta.

Teorema 3.1

Suponga que f es continua y satisface la condición de Lipschitz con constante L en

$$D = \{(t, y)|a \leq t \leq b \text{ y } -\infty < y < \infty\}$$

y que existe una constante M con

$$|y''(t)| \leq M, \text{ para todas las } t \in [a, b],$$

donde $y(t)$ denota la única solución para el problema de valor inicial

$$y' = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha.$$

Sean $w_0, w_1, ..., w_N$ las aproximaciones generadas por el método de Euler para un entero positivo N. Entonces, para cada $i = 0, 1, 2, \cdots, N$,

$$|y(t_i) - w_i| \leq \frac{hM}{2L}\left[e^{L(t_i-a)} - 1\right] \tag{5}$$

Ejemplo: Encontrar las cotas del error de Euler para la ecuación diferencial

$$y' = t - y, \quad y(0) = 0 \text{ con } 0 \leq t \leq 1$$

Solución:
La solución de la ecuación diferencial es

$$y = t - 1 + e^{-t}$$

Creamos un script llamado **cotas_euler.m** que llama a la función **euler.m** del informe anterior.
Código en Matlab/Octave:

```
%a,b: intervalo donde se define la EDO
%M : constante
%N : constante
%L : constante de Lipschitz
%f : f(x,y) de y'=f(x,y)
%y : solución analitica de y'=f(x,y) y0=alpha
a=0; b=1; %t en [a,b]
f = @(t,y) t-y; %función f(t,y) de la EDO
M=1;
L=1;
N=3; %para h=1/3
y = @(t) t-1+exp(-t); %sol. analítica de la EDO
y0=0; %condición inicial
%-----------------------------
% Cotas del error de Euler
%-----------------------------
h=(b-a)/N;
[t,w]=euler(a,b,N,y0,f);
fprintf('%8s %15s %20s\n', 't_i', '|y_i - w_i|', 'cota de error');
for i = 1:N
    ti = t(i);
    err = abs(y(ti) - w(i));
    bound = (h * M / (2 * L)) * abs(exp(L * (ti - a)) - 1);
    fprintf('%8.4f %15.6f %20.6f\n', ti, err, bound);
end
```

Manual:

- Abrir el archivo **cotas_euler.m** en Matlab/Octave.

- En las lineas

```
a=0; b=1; %t en [a,b]
f = @(t,y) t-y; %función f(t,y) de la EDO
M=1;
L=1;
N=3; %para h=1/3
y = @(t) t-1+exp(-t); %sol. analítica de la EDO
y0=0; %condición inicial
```

  Se ingresa el intervalo de la EDO, la función $f(t, y)$, las constantes M y L, y la solución analítica de la EDO.

- En la línea **N=3;** se define el número de pasos a realizar.
- Ejecutar el script con el comando **cotas_euler2** en la consola de Matlab/Octave.

Corrida del Programa:

```
>> cotas_euler
     t_i     |y_i - w_i|        cota de error
  0.0000        0.000000             0.000000
  0.3333        0.049865             0.065935
  0.6667        0.068973             0.157956
```

4. Método de taylor

El método de Taylor es un método numérico para resolver ecuaciones diferenciales ordinarias. Se basa en la expansión en serie de Taylor de la función solución alrededor de un punto inicial. La idea es aproximar la solución mediante una serie de potencias, utilizando las derivadas de la función en el punto inicial. La fórmula general del método de Taylor de orden n es:

$$y(t+h) = y(t) + h \cdot y'(t) + \frac{h^2}{2!} \cdot y''(t) + \frac{h^3}{3!} \cdot y'''(t) + \ldots + \frac{h^n}{n!} \cdot y^{(n)}(t) \tag{6}$$

donde $y(t)$ es la solución en el punto inicial, h es el tamaño del paso y $y^{(k)}(t)$ es la k-ésima derivada de la función solución evaluada en el punto inicial.

Ejemplo: Encontrar la solución de la ecuación diferencial por el método de taylor y compararla con la solución analítica.

$$y' = t - y, \quad y(0) = 0 \text{ con } 0 \leq t \leq 1$$

Solución
Código en Matlab/Octave:

```
a=0;b=1; %intervalo
t=a:0.1:b;
f1=@(t) t-1+exp(-t); %solución analítica
ff=@(t) t.*t/2-t.^3/6+t.^4/24; %solución aproximada
plot(t,f1(t),t,ff(t));
```

Manual:

- Abrir el archivo **M__Taylor.m** en Matlab/Octave.
- Se modifica el intervalo de la solución en la primera línea, se modifica la función **f1** para la solución analítica y la función **ff** para la aproximación de Taylor.
- Ejecutar el script con el comando **M__Taylor** en la consola de Matlab/Octave.

Corrida del Programa:

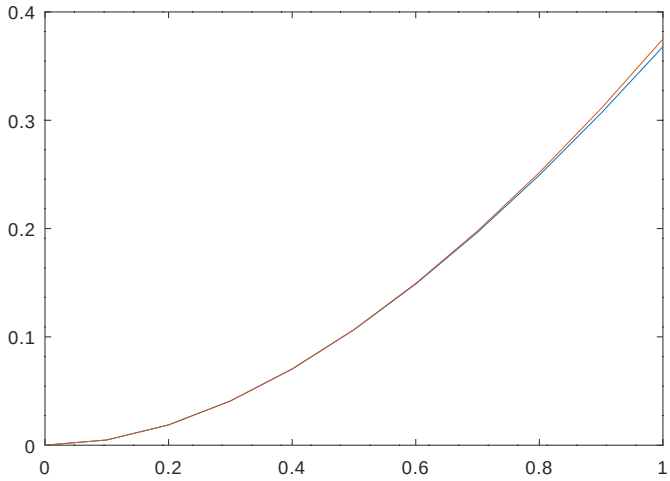


Figura 5: Comparación de la solución analítica y la aproximación de Taylor

5. Runge Kutta

5.1. Runge kutta de orden 2

Es un método numérico para resolver ecuaciones diferenciales ordinarias (EDO) de la forma

$$y' = f(t, y), \quad y(t_0) = y_0 \quad a \leq t \leq b$$

donde $f(t, y)$ es una función continua y y_0 es el valor inicial de la solución en el punto t_0.
El método Runge-Kutta de orden 2 es una extensión del método de Euler, proporcionando una mayor precisión en la aproximación de la solución.
RUNGE-KUTTA de orden 2

$$y_{i+1} = y_i + (a_1 k_1 + a_2 k_2)h \tag{7}$$

donde:

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h) \\ h &= (b-a)/N \end{aligned}$$

Éste es el método del punto medio.
Sea la EDO

$$y' = t - y^{1,5} \quad y(1) = 1 \text{ con } 1 \leq t \leq 2$$

Hacemos el Algoritmo Runge-Kutta de orden 2
Código en Matlab/Octave:

```
%Runge Kutta de orden 2
y1 = @(t,y) t-y^(3/2); %ec diff ordinaria
y0=1; %condición inicial
a=1;b=2; %intervalo donde se quiere la solución
%----------------------------------------
N = 8;
yt= zeros(1,N); yt(1)=y0;
h = (b-a)/N;
a1=a;
for i=1:N
    k1=h*y1(a1,y0);
    k2=h*y1(a1+h,y0+k1);
    yp=y0+(k1+k2)/2;yt(i+1)=yp;
```

```
        fprintf('%d| yn=%f| k1=%f| k2=%f| y_n+1=%f\n',i,y0,k1,k2,yp)
        y0=yp; a1=a1+h;
    end
     t1=a:h:b;
     plot(t1,yt,'.r','MarkerSize',10)
```

Manual:

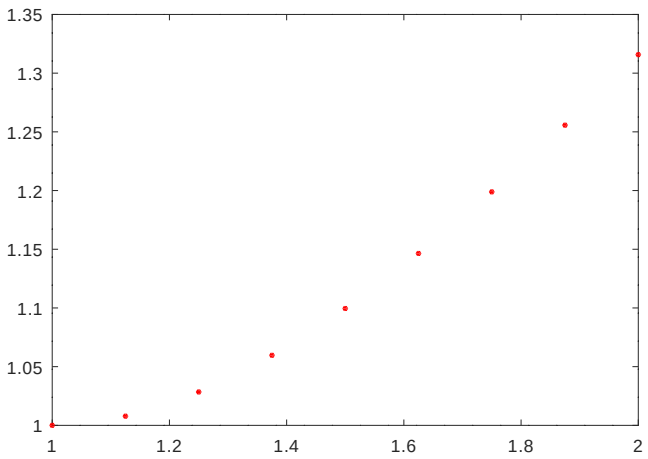
- Abrir el archivo **Rkorden2.m** en Matlab/Octave.
- En las lineas

  ```
  y1 = @(t,y) t-y^(3/2); %ec diff ordinaria
  y0=1; %condición inicial
  a=1;b=2; %intervalo donde se quiere la solución
  ```

  Se ingresa la ecuación diferencial ordinaria a solucionar, la condición inicial y el intervalo de solución.
- En la línea **N=8;** se define el número de pasos a realizar.
- Ejecutar el script con el comando **Rkorden2** en la consola de Matlab/Octave.

Corrida del Programa:

```
octave:12> RKorden2
1| yn=1.000000| k1=0.000000| k2=0.015625| y_n+1=1.007812
2| yn=1.007812| k1=0.014157| k2=0.027108| y_n+1=1.028445
3| yn=1.028445| k1=0.025879| k2=0.036552| y_n+1=1.059661
4| yn=1.059661| k1=0.035523| k2=0.044235| y_n+1=1.099540
5| yn=1.099540| k1=0.043379| k2=0.050392| y_n+1=1.146425
6| yn=1.146425| k1=0.049688| k2=0.055231| y_n+1=1.198885
7| yn=1.198885| k1=0.054662| k2=0.058938| y_n+1=1.255685
8| yn=1.255685| k1=0.058489| k2=0.061683| y_n+1=1.315771
```



5.2. Runge kutta de orden 3

Es un método numérico para resolver ecuaciones diferenciales ordinarias (EDO) de la forma

$$y' = f(t, y), \quad y(t_0) = y_0 \quad a \leq t \leq b$$

donde $f(t, y)$ es una función continua y y_0 es el valor inicial de la solución en el punto t_0. El método Runge-Kutta de orden 3 es una extensión del método de Runge-Kutta de orden 2, proporcionando una mayor precisión en la aproximación de la solución.
Para $n = 3$, es posible efectuar un desarrollo similar al del método de segundo orden. Una versión comun que se obtiene es

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 4k_2 + k_3)h \tag{8}$$

donde

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h) \\ k_3 &= f(x_i + h, y_i - k_1 h + 2k_2 h) \end{aligned}$$

Sea la EDO que no se puede resolver analiticamente

$$y' = t - ty^{1,5}$$

Entonces utilizando Runge-Kutta de orden 3, creamos un archivo en matlab/octave llamado RKorden3.m, y escribimos el siguiente codigo.
Código en Matlab/Octave:

```
%Runge Kutta de orden 3
y1 = @(t,y) t-y^(3/2); %Ec. Diff. Ordinaria a solucionar
y0=1; %condición inicial
a=1;b=2; %Intervalos
N=8;
h = (b-a)/N;
yt=zeros(1,N);yt(1)=y0;
a1=a;
for i=1:N
    k1=h*y1(a1,y0);
    k2=h*y1(a1+h/2,y0+k1/2);
    k3=h*y1(a1+h,y0+k1+2*k2);
    yp=y0+(k1+4*k2+k3)/6;yt(i+1)=yp;
    fprintf('%d| yn=%f| k1=%f| k2=%f| k3=%f y_n+1=%f\n',i,y0,k1,k2,k3,yp)
    y0=yp; a1=a1+h;
end
  t=a:h:b;
  pa=polyfit(t,yt,4); % minimos cuadrados
  pt=polyval(pa,t);
  plot(t,pt,t,yt,'.r','MarkerSize',10)
```

Manual:

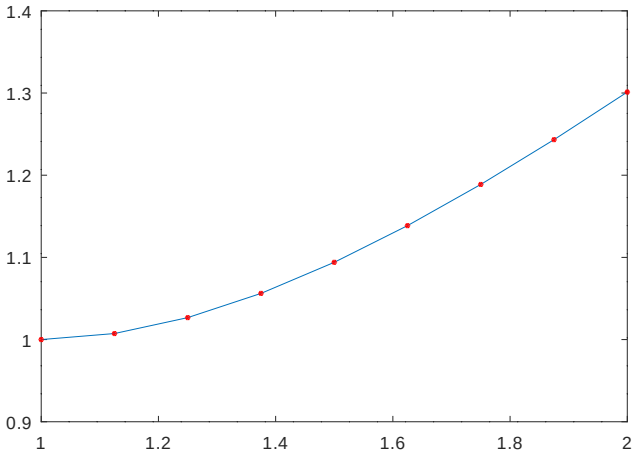
- Abrir el archivo **Rkorden3.m** en Matlab/Octave.
- En las lineas

  ```
  y1 = @(t,y) t-y^(3/2); %Ec. Diff. Ordinaria a solucionar
  y0=1; %condición inicial
  a=1;b=2; %Intervalos
  ```

  Se ingresa la ecuación diferencial ordinaria a solucionar, la condición inicial y el intervalo de solución.
- En la línea **N=8;** se define el número de pasos a realizar.
- Ejecutar el script con el comando **Rkorden3** en la consola de Matlab/Octave.

Corrida del Programa:

```
>> RKorden3
1| yn=1.000000| k1=0.000000| k2=0.007812| k3=0.012684 y_n+1=1.007322
2| yn=1.007322| k1=0.014250| k2=0.020719| k3=0.019251 y_n+1=1.026718
3| yn=1.026718| k1=0.026207| k2=0.031522| k3=0.024512 y_n+1=1.056186
4| yn=1.056186| k1=0.036193| k2=0.040504| k3=0.028619 y_n+1=1.093991
5| yn=1.093991| k1=0.044469| k2=0.047899| k3=0.031722 y_n+1=1.138622
6| yn=1.138622| k1=0.051252| k2=0.053909| k3=0.033965 y_n+1=1.188764
7| yn=1.188764| k1=0.056736| k2=0.058714| k3=0.035482 y_n+1=1.243277
8| yn=1.243277| k1=0.061090| k2=0.062477| k3=0.036398 y_n+1=1.301176
```



5.3. Runge kutta de orden 4

Es un procedimiento iterativo para resolver ecuaciones diferenciales ordinarias (EDO) de la forma

$$y' = f(t, y), \quad y(t_0) = y_0 \quad a \leq t \leq b$$

donde $f(t, y)$ es una función continua y y_0 es el valor inicial de la solución en el punto t_0. El método Runge-Kutta de orden 4 es uno de los métodos más utilizados debido a su precisión y estabilidad.
RUNGE-KUTTA de orden 4

$$\begin{aligned}
w_0 &= \alpha, \\
k_1 &= h \cdot f(t_i, w_i), \\
k_2 &= h \cdot f(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_1), \\
k_3 &= h \cdot f(t_i + \frac{h}{2}, w_i + \frac{1}{2}k_2), \\
k_4 &= h \cdot f(t_{i+1}, w_i + k_3), \\
w_{i+1} &= w_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}$$

Sea la EDO

$$y' = t - ty \quad y(0) = \frac{1}{2} \quad t \in [0, 1]$$

con solucion analítica

$$y(t) = 1 - \frac{1}{2}e^{-t^2/2}$$

Código en Matlab/Octave:

```
%Runge Kutta de orden 4
y1 = @(t,y) t-t*y; %ec. diff. ordinaria a solucionar
y0=1/2; %condición inicial
a=0;b=1; %intervalos
%----------------------
yt=[];yt(1)=y0;
N=4;
h = (b-a)/N;
a1=a;
for i=1:N
    k1=h*y1(a1,y0);
    k2=h*y1(a1+h/2,y0+k1/2);
    k3=h*y1(a1+h/2,y0+k2/2);
    k4=h*y1(a1+h,y0+k3);
    yp=y0+(k1+2*(k2+k3)+k4)/6;yt(i+1)=yp;
    fprintf('%d| yn=%f| k1=%f| k2=%f| k3=%f | k4=%f| y_n+1=%f\n',i,y0,k1,k2,k3,k4,yp)
    y0=yp; a1=a1+h;
end
  t1=a:h:b;
  t=a:0.01:b; y3=1-exp(-t.^2/2)/2;
  plot(t,y3,t1,yt,'.m','MarkerSize',20)
```

Manual:

- Abrir el archivo **Rkorden4.m** en Matlab/Octave.
- En las lineas

  ```
  y1 = @(t,y) t-t*y; %ec. diff. ordinaria a solucionar
  y0=1/2; %condición inicial
  a=0;b=1; %intervalos
  ```

  Se ingresa la ecuación diferencial ordinaria a solucionar, la condición inicial y el intervalo de solución.
- En la línea **N=4;** se define el número de pasos a realizar.
- Ejecutar el script con el comando **Rkorden4** en la consola de Matlab/Octave.

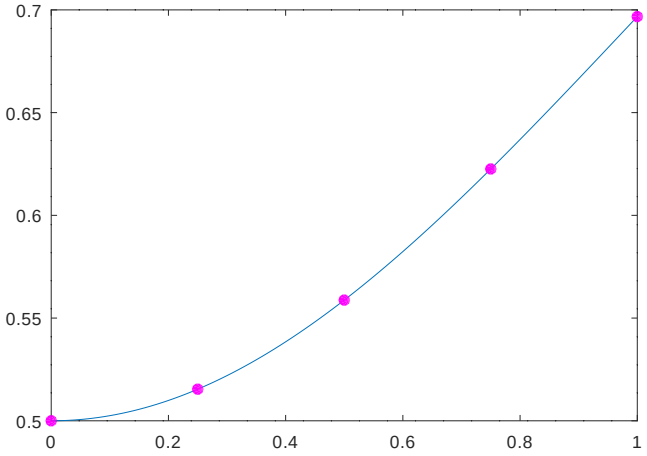
Corrida del Programa:

```
>> Rkorden4
1| yn=0.500000| k1=0.000000| k2=0.015625| k3=0.015381 | k4=0.030289| y_n+1=0.515383
2| yn=0.515383| k1=0.030289| k2=0.044013| k3=0.043370 | k4=0.055156| y_n+1=0.558752
3| yn=0.558752| k1=0.055156| k2=0.064636| k3=0.063895 | k4=0.070754| y_n+1=0.622580
4| yn=0.622580| k1=0.070766| k2=0.074820| k3=0.074377 | k4=0.075761| y_n+1=0.696734
```

5.4. Runge kutta como función en matlab/octave

Es el método de Runge-Kutta de orden 4, que a diferencia de como se escribió el programa en el informe 7, se ha implementado como una función que puede ser reutilizada por otros programas. Se crea un nuevo programa llamado **runge11.m**. **Este programa no se ejecuta directamente, sino que se llama desde otro programa. En el informe 10, se presenta un ejemplo de su uso.**
Código en Matlab/Octave:

```
%Runge Kutta de orden 4 como función
function yx=runge11(a,b,h,n,y0)
t=a;yx=[];yx(1)=y0;
for i=1:n
    k1=h*ff(t,y0);
```




```
    k2=h*ff(t+h/2,y0+k1/2);
    k3=h*ff(t+h/2,y0+k2/2);
    k4=h*ff(t+h,y0+k3);
    yy=y0+(k1+2*(k2+k3)+k4)/6;yx(i+1)=yy;
    y0=yy; t=t+h;
end
```

Donde

- `a` es el límite inferior del intervalo.
- `b` es el límite superior del intervalo.
- `h` es el tamaño del paso.
- `n` es el número de pasos.
- `y0` es la condición inicial.
- `ff` es la función que define la EDO a resolver.
- `yx` es el vector que contiene los valores de la solución aproximada.

la función **ff** se va definir en otro programa, que estara detallado en el siguiente informe. Este programa no se ejecuta directamente, sino que se llama desde otro programa. En el informe 10, se presenta un ejemplo de su uso.

5.5. función ff.m para Runge-Kutta

Es una función que define la ecuación diferencial ordinaria (EDO) a resolver. Esta función se llama desde el programa `runge11` y tiene la siguiente forma:

$$y' = t - ty \quad y(0) = \frac{1}{2} \quad t \in [0, 1]$$

El código siguiente se guarda como `ff.m`

```
function y1=ff(t,y)
    y1=t-t*y;
end
```

Este programa no se ejecuta directamente, sino que se llama desde otro programa. Es usado por `runge11.m`, que se detalla en el informe 8. En el informe 10, se presenta un ejemplo de su uso.

5.6. Ejemplo de uso de la función runge11

Es un ejemplo de uso de la función `runge11.m` que se definió en el informe 8. Esta función implementa el método de Runge-Kutta de orden 4 para resolver ecuaciones diferenciales ordinarias (EDO). En este caso, se utiliza para resolver la EDO:

$$y' = t - ty \quad y(0) = \frac{1}{2} \quad t \in [0, 1]$$

que se definio en el informe 9 mediante la función `ff.m`.
Se crea un nuevo programa llamado **llamadorRK.m** que llama a la función `runge11` y tiene la siguiente forma:

```
a=0;b=1;n=4;h=(b-a)/n;
y0=1/2;
y35=runge11(a,b,h,4,y0)
```

Se ejecuta el programa **llamadorRK.m** y se obtiene el siguiente resultado:

```
>> llamadorRK

y35 =

    0.5000    0.5154    0.5588    0.6226    0.6967
```

5.7. Runge kutta para sistemas de EDO's

Es un program que resuelve un sistema de EDO's de orden 1:

$$\begin{cases} x' = 2x + 3y \\ y' = \frac{2}{3}x + 3y \end{cases} \qquad x(0) = 0, y(0) = 1 \quad t \in [0, 1]$$

Se crea un nuevo programa con el nombre de **RK_sistemas.m**.
Código en Matlab/Octave:

```
%R-K de orden 4 para sistemas
clear all
%format long
format shortG
%---------------------
x1 = @(t,x,y) 2*x+3*y;
y1 = @(t,x,y) 2/3*x+3*y;
a=0;b=1;n=10;
t=a;x0=0;y0=1;
%---------------------
h=(b-a)/n; %paso
xt=[];yt=[];
xt(1)=x0;yt(1)=y0;
for i=1:n
    k11=h*x1(t,x0,y0);
    k12=h*y1(t,x0,y0);
    k21=h*x1(t+h/2,x0+k11/2,y0+k12/2);
    k22=h*y1(t+h/2,x0+k11/2,y0+k12/2);
    k31=h*x1(t+h/2,x0+k21/2,y0+k22/2);
    k32=h*y1(t+h/2,x0+k21/2,y0+k22/2);
    k41=h*x1(t+h,x0+k31,y0+k32);
    k42=h*y1(t+h,x0+k31,y0+k32);
    xs=x0+(k11+2*(k21+k31)+k41)/6;
    ys=y0+(k12+2*(k22+k32)+k42)/6;
    xt(i+1)=xs;yt(i+1)=ys;
    t=t+h;x0=xs;y0=ys;
```

```
end%i
u=0:h:1;
disp([u' xt' yt'])
```

Manual:

- Abrir el archivo **RK_sistemas.m** en Matlab/Octave.
- En las lineas

```
x1 = @(t,x,y) 2*x+3*y;
y1 = @(t,x,y) 2/3*x+3*y;
a=0;b=1;n=10;
t=a;x0=0;y0=1;
```

  Se define el sistema de EDO's a resolver y sus parámetros.

- Ejecutar el script con el comando **RK_sistemas** en la consola de Matlab/Octave.

Corrida del Programa:

```
>> RK_sistemas
  0            0            1
0.1      0.38656       1.3629
0.2       1.0039       1.8906
0.3       1.9696        2.663
0.4         3.46       3.7985
0.5       5.7381       5.4741
0.6        9.197       7.9535
0.7       14.424        11.63
0.8       22.295       17.089
0.9       34.118       25.205
  1       51.846       37.283
```

6. Método de Adams-Bashforth

6.1. Adams-Bashforth de orden 2

Es un ejemplo de uso del método de Adams-Bashforth de orden 2 para resolver la ecuación diferencial ordinaria (EDO):

$$y' = t - ty \quad y(0) = \frac{1}{2} \quad t \in [0, 1]$$

Este método utiliza el resultado de un método de Runge-Kutta de orden 4 para calcular el w_1 y luego aplica el método de Adams-Bashforth para los siguientes pasos. Por ese motivo, se utiliza la función `runge11.m` y `ff.m` que se definió en el informe 8 y 10 respectivamente.
Sea crea un nuevo programa con el nombre de **bashfordos11.m**
Código en Matlab/Octave:

```
%ejecuta adams-bashforth de orden 2
y0=1/2; %w1=? por R-K
a=0;b=1;n=4;k=1;%k=1 porque es de orden 2
%-----------------------
h=(b-a)/n;
yx=runge11(a,b,h,k,y0);
t=a+k*h;
for i=2:n
    Wx=yx(i)+h*(3*ff(t,yx(i))-ff(t-h,yx(i-1)))/2;
    yx(i+1)=Wx;
    t=t+h;
end
```

```
yx
```

Manual:

- Abrir el archivo **bashfordos11.m** en Matlab/Octave.
- En el programa `ff.m` se define la EDO a resolver.
- En las lineas

```
y0=1/2; %w1=? por R-K
a=0;b=1;n=4;k=1;%k=1 porque es de orden 2
```

  se definen los parámetros de la EDO a resolver
- En la línea **n=4;** se define el número de pasos a realizar.
- Ejecutar el script con el comando **bashfordos11** en la consola de Matlab/Octave.

Corrida del Programa:

```
>> bashforthdos11

yx =

    0.5000    0.5154    0.5608    0.6280    0.7052
```

6.2. Adams-Bashforth de orden 3

Es un ejemplo de uso del método de Adams-Bashforth de orden 3 para resolver la ecuación diferencial ordinaria (EDO):

$$y' = t - ty \quad y(0) = \frac{1}{2} \quad t \in [0, 1]$$

Este método utiliza el resultado de un método de Runge-Kutta de orden 4 para calcular el w_1 y luego aplica el método de Adams-Bashforth para los siguientes pasos. Por ese motivo, se utiliza la función `runge11.m` y `ff.m` que se definió en el informe 8 y 10 respectivamente.
Sea crea un nuevo programa con el nombre de **bashforthtres11.m**
Código en Matlab/Octave:

```
%ejecuta adams-bashforth de orden 3
y0=1/2; %w1=? por R-K
a=0;b=1;n=4;k=2;%k=2 porque es de orden 3
%---------------------
h=(b-a)/n;
yx=runge11(a,b,h,k,y0);
t=a+k*h;
for i=k+1:n
    Wx=yx(i)+h*(23*ff(t,yx(i))-16*ff(t-h,yx(i-1))+5*ff(t-2*h,yx(i-2)))/12;
    yx(i+1)=Wx;
    t=t+h;
end
yx
```

Manual:

- Abrir el archivo **bashforthtres11.m** en Matlab/Octave.
- En el programa `ff.m` se define la EDO a resolver.
- En las lineas

```
y0=1/2; %w1=? por R-K
a=0;b=1;n=4;k=2;%k=2 porque es de orden 3
```

se definen los parámetros de la EDO a resolver

- En la línea **n=4;** se define el número de pasos a realizar.
- Ejecutar el script con el comando **bashforthtres11** en la consola de Matlab/Octave.

Corrida del Programa:

```
>> bashforthtres11

yx =

    0.5000    0.5154    0.5588    0.6241    0.6983
```

6.3. Adams-Bashforth de orden 4

Es un ejemplo de uso del método de Adams-Bashforth de orden 4 para resolver la ecuación diferencial ordinaria (EDO):

$$y' = t - ty \quad y(0) = \frac{1}{2} \quad t \in [0, 1]$$

Este método utiliza el resultado de un método de Runge-Kutta de orden 4 para calcular el w_1 y luego aplica el método de Adams-Bashforth para los siguientes pasos. Por ese motivo, se utiliza la función `runge11.m` y `ff.m` que se definió en el informe 8 y 10 respectivamente.
Sea crea un nuevo programa con el nombre de **bashforthcuatro11.m**.
Código en Matlab/Octave:

```
%ejecuta adams-bashforth de orden 4
y0=1/2; %w1=? por R-K
a=0;b=1;n=4;k=3;%k=3 porque es de orden 4
%----------------------
h=(b-a)/n;
yx=runge11(a,b,h,k,y0);
t=a+k*h;
for i=k+1:n
    Wx=yx(i)+h*(55*ff(t,yx(i))-59*ff(t-h,yx(i-1))...
                +37*ff(t-2*h,yx(i-2))-9*ff(t-3*h,yx(i-3)))/24;
    yx(i+1)=Wx;
    t=t+h;
end
yx
```

Manual:

- Abrir el archivo **bashforthcuatro11.m** en Matlab/Octave.
- En el programa `ff.m` se define la EDO a resolver.
- En las lineas

```
y0=1/2; %w1=? por R-K
a=0;b=1;n=4;k=3;%k=3 porque es de orden 4
```

se definen los parámetros de la EDO a resolver

- En la línea **n=4;** se define el número de pasos a realizar.

- Ejecutar el script con el comando **bashforthcuatro11** en la consola de Matlab/Octave.

Corrida del Programa:

```
>> bashforthcuatro11

yx =

    0.5000    0.5154    0.5588    0.6226    0.6959
```

7. Método de Moulton

Es un ejemplo de uso del método de Moulton para resolver la ecuación diferencial ordinaria (EDO):

$$y' = t - ty \quad y(0) = \frac{1}{2} \quad t \in [0, 1]$$

Este método es un método implícito de predicción-corrección que utiliza el valor de la función en el paso siguiente para corregir el valor actual. Por ese motivo, se utiliza la función `runge11.m` que se definió en el informe 8.
Sea crea un nuevo programa con el nombre de **moulton.m**.
Código en Matlab/Octave:

```
clear all;
y0=1/2; %condición inicial
a=0;b=1;% intervalo
n=4; % número de pasos
%-----------------------
t=a;
h=(b-a)/n;
yt=[];yt(1)=y0;
u=runge11(a,b,h,1,y0);
yt=u;
t=a+h;
y0=yt(2);
for i=2:n
  v=y0;
  for k=1:5
    v1=y0+h/12*(5*ff(t+h,v)+8*ff(t,yt(i))-ff(t-h,yt(i-1)));
    if abs(v1-v)<0.0001
        break
    end
    v=v1;
  end%k
  y0=v1;t=t+h;yt(i+1)=y0;
end%i
u=a:h:b;
disp([u',yt']);
```

Manual:

- Abrir el archivo **moulton.m** en Matlab/Octave.
- En el programa `ff.m` se define la EDO a resolver.
- En las lineas

```
y0=1/2; %condición inicial
a=0;b=1;% intervalo
n=4; % número de paso
```

se definen los parámetros de la EDO a resolver

- En la línea **n=4;** se define el número de pasos a realizar.
- Ejecutar el script con el comando **moulton** en la consola de Matlab/Octave.

Corrida del Programa:

```
>> Moulton
    0    0.5000
0.2500    0.5154
0.5000    0.5586
0.7500    0.6223
1.0000    0.6965
```

7.1. Método de Moulton de orden 3

Es un ejemplo de uso del método de Moulton de orden 3 para resolver la ecuación diferencial ordinaria (EDO):

$$y' = t - ty \quad y(0) = \frac{1}{2} \quad t \in [0, 1]$$

Este método es un método implícito de predicción-corrección que utiliza el valor de la función en el paso siguiente para corregir el valor actual. Por ese motivo, se utiliza la función `runge11.m` que se definió en el informe 8.
Sea crea un nuevo programa con el nombre de **moulton3.m**.
Código en Matlab/Octave:

```
clear all;
y0=1/2;%condición inicial
a=0;b=1;% intervalo
n=4;% número de pasos
%--------------------
t=a;
h=(b-a)/n;
yt=[];yt(1)=y0;
u=runge11(a,b,h,2,y0);
yt=u;
t=a+2*h;
y0=yt(3);
for i=3:n
  v=y0;
  for k=1:5
    v1=y0+h/24*(9*ff(t+h,v)+19*ff(t,yt(i))-5*ff(t-h,yt(i-1))+ff(t-2*h,yt(i-2)));
    if abs(v1-v)<0.0001
        break
    end
    v=v1;
  end%k
  y0=v1;t=t+h;yt(i+1)=y0;
end%i
u=a:h:b;
disp([u',yt']);
```

Manual:

- Abrir el archivo **moulton3.m** en Matlab/Octave.
- En el programa `ff.m` se define la EDO a resolver.
- En las lineas

```
y0=1/2; %condición inicial
a=0;b=1;% intervalo
n=4; % número de paso
```

se definen los parámetros de la EDO a resolver

- En la línea **n=4;** se define el número de pasos a realizar.
- Ejecutar el script con el comando **moulton3** en la consola de Matlab/Octave.

Corrida del Programa:

```
>> Moulton3
    0    0.5000
0.2500    0.5154
0.5000    0.5588
0.7500    0.6226
1.0000    0.6968
```

8. Graficar sistemas de ecuaciones diferenciales(EDO) de orden 1

Es un program que gráfica las soluciones de un sistema de EDO's de orden 1:

$$\begin{cases} x' = 2x + 3y \\ y' = \frac{2}{3}x + 3y \end{cases} \qquad x(0) = 0, y(0) = 1 \quad t \in [0, 1]$$

Sea crea un nuevo programa con el nombre de **runge4sistema.m**.
Código en Matlab/Octave:

```
format long G
t=0:0.01:1;%dominio
x=@(t) exp(4*t)-exp(t); %soluciones
y=@(t) 2/3*exp(4*t)+1/3*exp(t); %soluciones
plot(t,x(t),'r',t,y(t),'b')
u=0:1/2:1;
[u' x(u)' y(u)']
```

Manual:

- Abrir el archivo **runge4sistema.m** en Matlab/Octave.
- Ejecutar el script con el comando **runge4sistema** en la consola de Matlab/Octave.

Corrida del Programa:

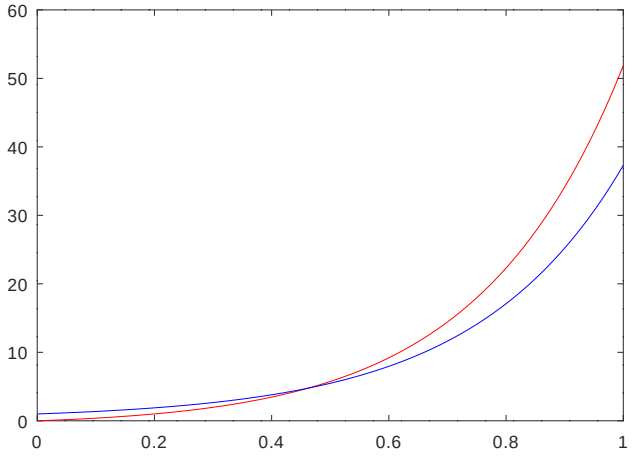
```
ans =

   0                        0                        1
 0.5          5.74033482823052          5.47561115618714
   1          51.8798682046852          37.3048606315825
```

9. Disparo lineal

Este programa resuelve un problema de contorno para una ecuación diferencial lineal de segundo orden:

$$y'' + 3y' + 2y = t$$

usando el método del disparo lineal. Este método convierte un problema de contorno en un conjunto de problemas de valor inicial que se resuelven con técnicas numéricas, como Runge-Kutta. El resultado se compara con la solución exacta conocida del problema.
Variables:



- `a`, `b`: Extremos del intervalo $[a, b]$, donde se resuelve la ecuación diferencial.
- `n`: Número de subintervalos para el método numérico.
- `h`: Tamaño del paso, calculado como $h = \frac{b-a}{n}$.
- `t`: Variable que recorre el intervalo $[a, b]$, utilizada tanto en la integración como en la solución exacta.
- `x0`, `y0`: Condiciones iniciales para x e y en el método del disparo.
- `k`: Parámetro que se pasa a la función diferencial definida en `ff.m`, dependiendo de la proporción que se está resolviendo.
- `yy`, `y11`, `y22`: Vectores con las soluciones numéricas obtenidas de las dos proporciones resueltas mediante el método de Runge-Kutta.
- `ys`: Solución lineal combinada del método de disparo, ajustada para satisfacer la condición de frontera.
- `yv`: Solución exacta del problema, usada para comparar contra `ys`.
- `c1`, `c2`: Constantes calculadas para definir la solución exacta del problema.

Programa en Matlab/Octave:

```
%disparo lineal y''+3y'+2y=t
a=0;b=1;n=10;h=(b-a)/n;t=a;
%y1(x) es la primera proporción
k=1;x0=0;y0=1;
yy=runge_sis(a,b,x0,0,n,1);
y11=yy(1,:);
%y2(x) la segunda proporción
k=0;yy=runge_sis(a,b,0,01,n,0);
y22=yy(1,:);
ys=y11+(y0-y11(n+1))*y22/y22(n+1);
ys'
t=a:h:b;
c2=(5-3*exp(-2))/(4*(exp(-1)-exp(-2)));c1=3/4-c2;
yv=c1*exp(-2*t)+c2*exp(-t)+t/2-3/4;%solucion exacta
plot(t,ys,t,yv)
%
%
%
```

Manual:
1. Este archivo debe guardarse como `disparo_lineal.m`.
2. Requiere tener definido el archivo `ff.m`, que contenga la ecuación diferencial como función de entrada.
3. También requiere el archivo `runge_sis.m`, que implementa el método de Runge-Kutta para sistemas.
4. Se ejecuta desde la consola con:

```
>> disparo_lineal
```

5. El programa grafica la solución numérica aproximada y la solución exacta, y además imprime la solución aproximada `ys` en consola.

Corrida del Programa:

```
octave:4> disparo_lineal
ans =

         0
    0.2982
    0.4959
    0.6303
    0.7251
    0.7950
    0.8492
    0.8938
    0.9324
    0.9673
    1.0000
```

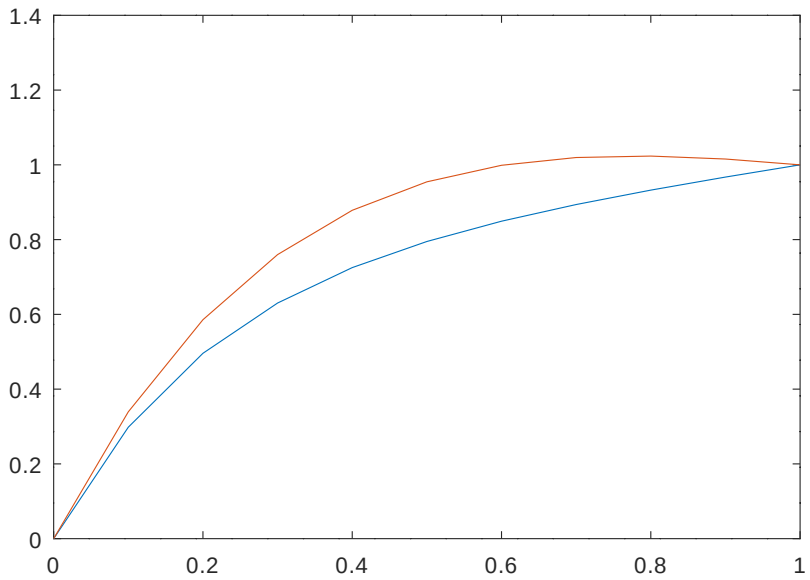


Figura 6: Gráfico de la solución del disparo lineal.

9.1. Método del disparo con Runge-Kutta de orden 4

Este programa implementa el método del disparo para resolver un problema de valor en la frontera (boundary value problem, BVP) para un sistema de ecuaciones diferenciales de primer orden. El método convierte el BVP en un problema de valor inicial (IVP), que luego se resuelve usando el método de Runge-Kutta de orden 4.

En particular, el sistema resuelto es:

$$\begin{cases} x' = 2x + 3y \\ y' = \frac{2}{3}x + 3y \\ x(0) = 0, \quad y(0) = 1, \quad t \in [0, 1] \end{cases}$$

Este tipo de técnica es común cuando se conocen condiciones en los extremos del intervalo y se requiere una solución aproximada de alta precisión.

—

Variables:

- `a`, `b`: Extremos del intervalo de integración en el tiempo ($t \in [a, b]$).
- `x`, `y`: Condiciones iniciales $x(a)$, $y(a)$.
- `n`: Número de subintervalos en los que se divide el intervalo $[a, b]$; define la precisión del método.
- `h`: Tamaño del paso $h = \frac{b-a}{n}$.
- `t`: Variable del tiempo que avanza en cada iteración.
- `x0`, `y0`: Valores actuales de las variables x e y en cada paso.
- `xt`, `yt`: Vectores donde se almacenan los valores aproximados de $x(t)$ y $y(t)$.
- `kij`: Coeficientes intermedios del método de Runge-Kutta para calcular la solución numérica, tanto para x como para y.
- `fff(t,x,y)`: Función que representa la derivada de x en el sistema (es decir, $x' = f(x, y)$).
- `fffg(t,x,y)`: Función que representa la derivada de y en el sistema (es decir, $y' = g(x, y)$).
- `yy`: Matriz de dos filas: la primera contiene los valores de $x(t)$, y la segunda los de $y(t)$.

—

Programa en Matlab/Octave:

```
% x' = 2x   + 3y
% y' = 2x/3 + 3y ;       x(0)=0;  y(0)=1;  t en [0,1]
function yy=disparo_r_k_11(a,b,x,y,n)
h=(b-a)/n; t=a;
x0=x; y0=y; xt(1)=x0; yt(1)=y0;
% k11 k12 k13 k14 para fff(t)
% k21  k22  k23 k24 para fffg(t)
for i=1:n
  k11 = h*fff(t, x0,y0);                  k21=h*fffg(t,x0,y0);
  k12 = h*fff(t+h/2, x0+k11/2,y0+k21/2);  k22=h*fffg(t+h/2,x0+k11/2,y0+k21/2);
  k13 = h*fff(t+h/2, x0+k12/2,y0+k22/2);  k23=h*fffg(t+h/2,x0+k12/2,y0+k22/2);
  k14 = h*fff(t+h, x0+k13,y0+k23);        k24=h*fffg(t+h,x0+k13,y0+k23);
  xs=x0+(k11 + 2*(k12+k13) + k14)/6; xt(i+1)=xs;
  ys=y0+(k21 + 2*(k22+k23) + k24)/6; yt(i+1)=ys;
  t=t+h; x0=xs; y0=ys;
end;
yy=[xt ; yt];
```

Manual: Se escribe un nuevo script de nombre disparo_r_k_11.m, que es el programa que resuelve el problema de valor de frontera con el método de Runge-Kutta de orden 4. Este programa usa las funciones fff(t,x,y) y fffg(t,x,y) que se escribieron en los informes siguientes.
Corrida del Programa: Este programa no se ejecuta directamente, sino que se llama desde otro programa de nombre llamador.m.

9.2. Método del disparo con fórmula de las secantes

Este programa implementa el método del disparo combinado con la fórmula de las secantes para resolver un problema de valor en la frontera (BVP) de una ecuación diferencial ordinaria (EDO) de segundo orden. El objetivo es encontrar la condición inicial desconocida (en este caso, $y'(0)$) que hace que la solución numérica satisface la condición de frontera en el extremo derecho del intervalo. El método consiste en probar diferentes valores para $y'(0)$, integrando el sistema con Runge-Kutta (usando `disparo_r_k_11`), y ajustando el valor usando la fórmula de las secantes hasta cumplir con la condición deseada en $y(b) \approx B$.
Variables:

- `a`, `b`: Extremos del intervalo de integración $[a, b] = [0, 1]$.
- `n`: Número de subintervalos para la integración numérica.
- `x`, `y`: Valores iniciales de la función $x(t)$ e $y(t)$ en $t = a$. Aquí $x = y(0) = 0$.
- `B`: Valor deseado en la frontera derecha $y(b) = B = 0{,}5$, usado como condición de contorno.
- `yn`, `yn1`, `yn2`: Sucesivos valores aproximados de la condición inicial desconocida $y'(0)$, ajustados usando la fórmula de las secantes.
- `gt`, `gr`: Matrices de resultados devueltos por `disparo_r_k_11` para diferentes valores de $y'(0)$.
- `g1`: Fila que representa los valores aproximados de $x(t)$ (o $y(t)$) en el tiempo.
- `ytp1`, `ytp2`: Valores de $y(b)$ obtenidos en las integraciones numéricas anteriores, usados en la fórmula de las secantes.

Programa en Matlab/Octave:

```
a=0;b=1;n=4;x=0;y=0;B=0.5;
yn=0;yn1=1;
for i=1:5
gt=disparo_r_k_11(a,b,x,yn,n);
gr=disparo_r_k_11(a,b,x,yn1,n);
g1=gt(1,:);
ytp1=g1(n+1);
g1=gr(1,:);
ytp2=g1(n+1);
%formula de secantes
yn2=yn1-(ytp2-B)*(yn1-yn)/(ytp2-ytp1)
if abs(yn1-yn2)<0.001 break;end;
yn=yn1;yn1=yn2;
end
```

Manual:
1. Este código debe guardarse como `llamador.m`.
2. Requiere previamente tener definidos los archivos `disparo_r_k_11.m`, `fff.m` y `fffg.m`.
3. Ejecutar en la consola de Octave o Matlab con:

```
>> llamador
```

4. El programa aplicará el método del disparo usando dos soluciones aproximadas con valores diferentes de $y'(0)$ e irá ajustando esos valores usando la fórmula de las secantes, hasta que se cumpla la condición de frontera $y(1) = 0{,}5$ con una tolerancia de error menor a 0.001.
Corrida del Programa:

```
octave:2> llamador
yn2 = 1.7904
yn2 = 1.7904
```

9.3. Método del disparo para ecuaciones diferenciales no lineales

Este conjunto de programas resuelve un problema de valor en la frontera no lineal (BVP) de la forma:

$$y'' + 3y' + 2y = t, \quad y(a) = ya, \quad y(b) = yb,$$

utilizando el método del disparo, que transforma el problema BVP en un problema de valor inicial (IVP). La idea es proponer una pendiente inicial $y'(a) = s$, resolver el sistema con esta condición inicial mediante el método de Runge-Kutta, y ajustar s iterativamente hasta que la solución cumpla con la condición de frontera en b.
Se implementa una versión no lineal del método del disparo, en la cual se aplica el método de la secante para aproximar la pendiente correcta que satisface la condición final. La no linealidad del problema requiere un ajuste iterativo de esta pendiente hasta obtener el valor deseado en $y(b)$.

—

Variables:

- `a`, `b`: Extremos del intervalo del dominio de la solución.
- `n`: Número de subintervalos para la discretización del intervalo.
- `h`: Tamaño del paso, calculado como $h = (b - a)/n$.
- `y1`, `y2`: Pendientes iniciales propuestas para el disparo, que se actualizan con el método de la secante.
- `k`: Parámetro que representa el control de la no homogeneidad del sistema (presencia del término $r(t)$).
- `runge_sis_`: Función externa que aplica el método de Runge-Kutta de cuarto orden para resolver el sistema asociado a una EDO.
- `y11`, `y22`: Soluciones del IVP con distintas pendientes iniciales.
- `ys1`: Nueva pendiente calculada mediante el método de la secante.
- `ff`: Función que define la EDO como $y'' = f(t, y, y')$, y que se implementa por separado en `ff.m`.

Programa en Matlab/Octave de la función ff.m:

```
% AQUI PONGA LA FUNCION DE y' = ay + r(t)
% por ejemplo:   y'' +3y' + 2y = t
% cambio de variable   y1 = y',  y1' = -2*y -3*y1 + t * k
%   x1=@(t,x,y)    y;
%   y1=@(t,x,y) -2*y - 3* y1 + t * k;
%
function yy=ff(t, y , y1 , k)  % k para r(t) <> 0   o  r(t)=0
   yy =  -2*y - 3.*y1 + t;
end
```

Programa en Matlab/Octave de Disparo_No_lineal.m:

```
% METODO DEL DISPARO NO LINEAL
% por ejemplo:   y'' +0y' + y^2/100 = t;  y(a)=ya ,  y(b)=yb;
% cambio de variable   y1 = y',  y1' = -2*y -3*y1 + t * k
%   x1=@(t,x,y)    y;
%   y1=@(t,x,y) x*x/10  + t ;
%
% % llama a funtion yy=runge_sis_(a,b,x,y,n,k)
%
%  y(t)  =  y1(t) + (yb - y1(b))*y2(t)/y2(b) ,  es la solucion
%
clear all;
a=0; b=1; n=8;  h=(b-a)/n;  err1=.00000001;
x=0; y=01/2;    B=y;% fronteras
```

```
% ingresar pendiente 1  y 2
y1=.3; y2=4; %y11=y22=[ ];
k=1;  %   pendiente yn
d=runge_sis_(a,b,x,y1,n,k);
y11=d(1,:);

for i=1: 8000


   % para y2(t)   no homogeneo  r(t) <> 0
   % pendiente yn1
   d=runge_sis_(a,b,x,y2,n,k);
   y22=d(1,:);
   % SOLUCION
   ys1 = y2 - (y22(n+1) - B)/(y22(n+1)-y11(n+1))*(y2-y1);
   printf('%d  yn=%f  yn+1 =%f y(b)=%f \n',i , y2, ys1, y22(n+1));
   if abs(ys1-y2)<err1 break; end;
   y1=y2; y2=ys1; y11=y22;
end; ys1, d(1, n+1)
i, y22
t=a:h:b;
plot(t,y22);
grid;
```

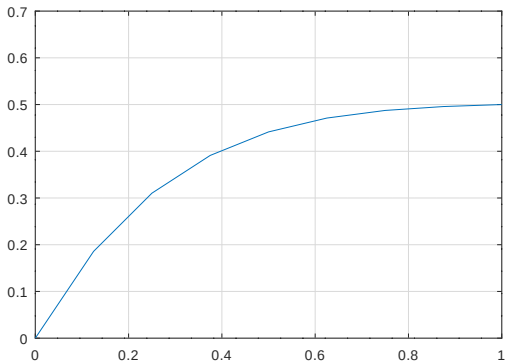
Manual:
1. Guarde los archivos como `ff.m` y `Disparo_No_lineal.m`.
2. Asegúrese de tener la función `runge_sis_` correctamente implementada y disponible en el mismo directorio.
3. Ejecute el programa principal con:

```
>> Disparo_No_lineal
```

4. El script ajusta la pendiente inicial para que la solución del problema de valor inicial cumpla con la condición de frontera en $t = b$.
5. Al finalizar, se grafica la solución aproximada.

Corrida del Programa:

```
1  yn=4.000000  yn+1 =1.788797 y(b)=1.014180
2  yn=1.788797  yn+1 =1.788797 y(b)=0.500000
ys1 = 1.7888
ans = 0.5000
i = 2
y22 =

         0    0.1858    0.3103    0.3911    0.4414    0.4711    0.4874    0.4957    0.5000
```



10. Método de diferencias finitas

10.1. Método de diferencias finitas para problemas lineales

Este programa resuelve un problema de valor en la frontera (BVP) para una ecuación diferencial lineal de segundo orden utilizando el método de diferencias finitas. En lugar de construir la matriz del sistema de forma automática, en este caso se ha calculado manualmente la matriz de coeficientes A y el vector del lado derecho b, lo que simplifica la implementación y permite concentrarse en el concepto del método. Se supone que la EDO tiene la forma general:

$$y'' + p(x)y' + q(x)y = r(x), \quad y(0) = 0,\ y(1) = 1$$

y que el dominio $[0, 1]$ se ha discretizado con $n = 4$ puntos (3 nodos interiores).

—

Variables:

- `A`: Matriz de coeficientes del sistema lineal generado por el método de diferencias finitas, con los valores ya ingresados manualmente.
- `b`: Vector del lado derecho del sistema, calculado según las condiciones de frontera y los valores de la función $r(x)$ en los nodos internos.
- `y`: Vector solución del sistema lineal $A \cdot y = b$, que contiene las aproximaciones de la función en los nodos interiores. Se completa con las condiciones de frontera: $y(0) = 0$ y $y(1) = 1$.
- `x`: Vector que contiene los puntos del dominio equiespaciados en el intervalo $[0, 1]$, incluyendo los extremos.

Programa en Matlab/Octave:

```
A=[-15/8 11/8 0; 5/8 -15/8 11/8;0 5/8 -15/8];
b=[1/64 1/32 -85/64];
y=inv(A)*b'
y=[0 y' 1];
x=0:1/4:1;
plot(x,y)
```

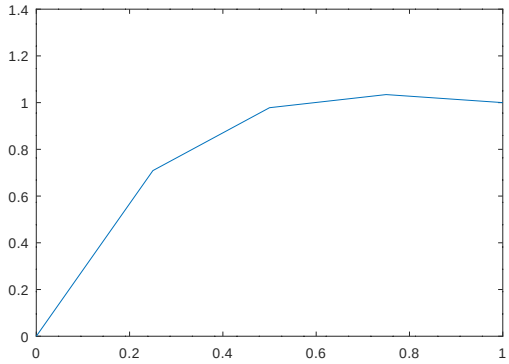
Manual:
1. Guarde el código en un archivo llamado `dif_finitas_manual.m`.
2. Ejecute desde la consola de Octave o Matlab con:

```
>> dif_finitas_manual
```

3. El programa resuelve el sistema lineal ingresado manualmente y grafica la función aproximada $y(x)$ en el intervalo $[0, 1]$, incluyendo los valores de frontera.
4. La matriz A y el vector b se han deducido manualmente a partir de la discretización por diferencias finitas y de las condiciones de contorno.

Corrida del Programa:

```
y =

   0.7091
   0.9783
   1.0344
```



10.2. Método de diferencias finitas con contrucción automática de matriz

Este programa resuelve un problema de valor en la frontera (BVP) para una ecuación diferencial de segundo orden utilizando el método de diferencias finitas, donde la matriz de coeficientes y el vector del segundo miembro se generan automáticamente dentro del código.
El problema considerado es de la forma:

$$y'' + py' + qy = r(x), \quad \text{en } [a, b], \quad y(a) = y_a,\ y(b) = y_b$$

En este caso particular:
- $p = 3$, $q = 2$
- $r(x) = x$ (función lineal)
- Condiciones de frontera: $y(0) = 0$, $y(1) = 1$

Se discretiza el intervalo $[0, 1]$ en $n = 10$ subintervalos y se construye un sistema lineal $A \cdot y = b$, que se resuelve para encontrar las aproximaciones de $y(x)$ en los nodos interiores.

—

Variables:

- `p`, `q`: Coeficientes de la EDO en los términos y' y y respectivamente.
- `r(x)`: Función fuente del lado derecho de la ecuación. En este caso, es $r(x) = x$.
- `a`, `b`: Extremos del intervalo donde se resuelve el problema.
- `n`: Número de subintervalos de la discretización (hay $n - 1$ nodos interiores).
- `ya`, `yb`: Valores de las condiciones de frontera en $x = a$ y $x = b$, respectivamente.
- `h`: Paso de la malla, $h = \frac{b-a}{n}$.
- `A`: Matriz de coeficientes del sistema lineal construida con base en la discretización de la EDO.
- `b`: Vector del segundo miembro construido en función de $r(x)$ y de las condiciones de frontera.
- `cc`: Solución del sistema lineal, concatenada con los valores de frontera para obtener la solución completa.
- `t`: Vector que contiene los nodos de la discretización.

Programa en Matlab/Octave:

```
% y''+py''+qy=r(x)
p=3;q=2;a=0;b=1;n=10;
ya=0;yb=1;
%------------------
h=(b-a)/n;
A=[];b1=b;
A(1,1)=2*h.^2-2;
A(1,2)=1+3*h/2;
b(1)=(a+h)*h.^2-(1-3*h/2)*ya;
```

```
for i=2:(n-2)
        A(i,i-1)=1-3*h/2;
        A(i,i)=2*h*h-2;
        A(i,i+1)=1+3*h/2;
        b(i)=h.^2*(i*h);
end%i
A(n-1,n-2)=1-3*h/2;A(n-1,n-1)=2*h.^2-2;b(n-1)=(a+(n-1)*h)*h.^2-(1+3*h/2)*yb;
[A b']
cc=A\b';cc=[ya cc' yb];
t=a:h:b1;
plot(t,cc,'LineWidth', 2);  % Cambia el 2 por el grosor que desees
```

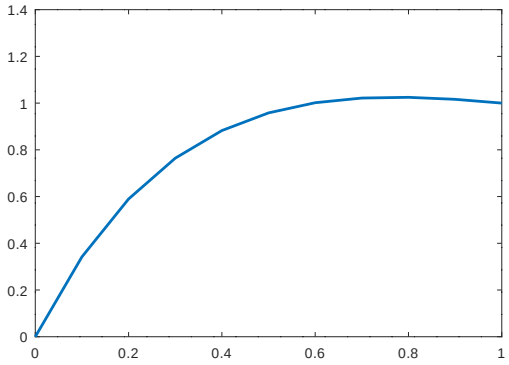
Manual:
1. Guarde el código en un archivo llamado `dif_finitas_auto.m`.
2. Ejecute en Octave o Matlab con:

```
>> dif_finitas_auto
```

3. El programa construye automáticamente la matriz A y el vector b a partir de los coeficientes p, q, y de la función $r(x) = x$.
4. Resuelve el sistema $A \cdot y = b$ e incorpora las condiciones de frontera para graficar la solución aproximada.

Corrida del Programa:

```
ans =

  -1.9800   1.1500        0        0        0        0        0        0        0   0.0010
   0.8500  -1.9800   1.1500        0        0        0        0        0        0   0.0020
        0   0.8500  -1.9800   1.1500        0        0        0        0        0   0.0030
        0        0   0.8500  -1.9800   1.1500        0        0        0        0   0.0040
        0        0        0   0.8500  -1.9800   1.1500        0        0        0   0.0050
        0        0        0        0   0.8500  -1.9800   1.1500        0        0   0.0060
        0        0        0        0        0   0.8500  -1.9800   1.1500        0   0.0070
        0        0        0        0        0        0   0.8500  -1.9800   1.1500   0.0080
        0        0        0        0        0        0        0   0.8500  -1.9800  -1.1410
```



10.3. Método de diferencias finitas iterativo para problema no lineal

Este programa resuelve un problema de valor en la frontera no lineal mediante el método de diferencias finitas iterativo. La ecuación diferencial a resolver es:

$$y'' - \frac{y^2}{10} = t, \quad \text{con } y(0) = 0, \quad y(1) = 1$$

Este problema no es lineal debido al término y^2, por lo que se emplea una linealización iterativa, donde se sustituye y^2 por una función $s(t)y$, y se actualiza $s(t)$ con el valor de y obtenido en cada iteración. De esta forma, en cada paso se resuelve un sistema lineal que aproxima la solución del problema no lineal.

—

Variables:

- `p`: Coeficiente del término y', que es cero en este caso.
- `a`, `b`: Extremos del intervalo en el que se resuelve la EDO.
- `n`: Número de subintervalos para la discretización. Esto define $n + 1$ nodos.
- `h`: Tamaño del paso, calculado como $h = (b - a)/n$.
- `ya`, `yb`: Condiciones de frontera $y(a) = ya$ y $y(b) = yb$.
- `st`: Vector que almacena los valores de la función $s(t)$, que se actualiza con los valores de y en cada iteración.
- `u`: Término auxiliar igual a h^2, utilizado para simplificar expresiones.
- `A`: Matriz del sistema lineal generado por el método de diferencias finitas.
- `bb`: Vector del segundo miembro del sistema, dependiendo de la función fuente $r(x) = t$ y de las condiciones de frontera.
- `cc`: Solución del sistema lineal en cada iteración; representa la aproximación de la solución y en los nodos interiores.
- `t`: Vector de nodos del intervalo $[a, b]$.

Programa en Matlab/Octave:

```
%No lineal Iteraciones Metodo de diferencias finitas
%y''-y^2/10=t  y(0)=0; y(1)=1
%y''-s(t)y/10=t; donde s(t)=y
p=0;n=4;%q=st
a=0;b=1;h=(b-a)/n;A=[];st=a:h:b;st=st(2:n);
ya=0;yb=1;u=h.^2;
A(1,1)=st(1)*u-2;A(1,2)=1+0*h/2;bb(1)=(a+h)*u-ya;
for j=1:5
for i=2:(n-2)
        A(i,i-1)=1;A(i,i)=st(i)*u-2;A(i,i+1)=1;
        bb(i)=u*(i*h);
end%i
A(n-1,n-2)=1;A(n-1,n-1)=st(n-1)*u-2;
bb(n-1)=(a+(n-1)*h)*u-yb;
[A,bb']
cc=A\bb';
if norm(cc-st)<0.01 break;end
st=cc/10;
end;%j
cc=[ya cc' yb]
t=a:h:b;
plot(t,cc);
```

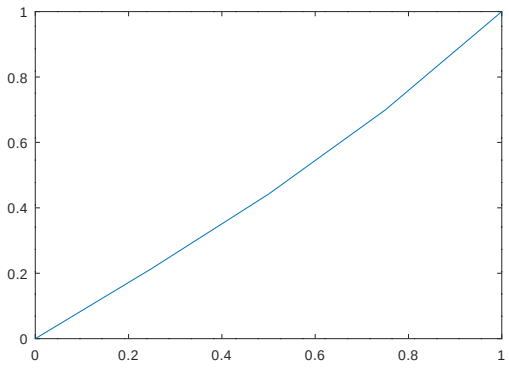
Manual:
1. Guarde el programa en un archivo llamado `nolineal.m`.
2. Ejecute el script desde la consola de Octave o Matlab con:

```
>> nolineal
```

3. El código implementa un proceso iterativo para resolver una EDO no lineal utilizando el método de diferencias finitas.
4. En cada iteración se linealiza la ecuación con una estimación de $s(t) = y$, se forma el sistema lineal correspondiente, y se resuelve. El proceso continúa hasta que el cambio entre iteraciones es menor a 0.01.
5. Al final, se grafica la solución aproximada en el intervalo $[0, 1]$.

Corrida del Programa:

```
octave:10> nolineal
ans =

  -1.9844   1.0000        0   0.0156
   1.0000  -1.9688   1.0000   0.0312
        0   1.0000  -1.9531  -0.9531

ans =

  -1.9844   1.0000        0   0.0156
   1.0000  -1.9971   1.0000   0.0312
        0   1.0000  -1.9954  -0.9531

ans =

  -1.9844   1.0000        0   0.0156
   1.0000  -1.9972   1.0000   0.0312
        0   1.0000  -1.9956  -0.9531

ans =

  -1.9844   1.0000        0   0.0156
   1.0000  -1.9972   1.0000   0.0312
        0   1.0000  -1.9956  -0.9531

ans =

  -1.9844   1.0000        0   0.0156
   1.0000  -1.9972   1.0000   0.0312
        0   1.0000  -1.9956  -0.9531

cc =

        0   0.2148   0.4419   0.6991   1.0000
```



10.4. Diferencias finitas para EDP's de tipo parabólico

Este programa implementa el método de diferencias finitas para resolver una ecuación diferencial parcial (EDP) de tipo parabólico, utilizando Matlab/Octave. Se calcula la evolución temporal de la solución en una malla discreta, mostrando los resultados numéricos y la gráfica correspondiente.
Variables:

- `la`: Parámetro de estabilidad, calculado como $(4 \times 1/65)/(0{,}5^2)$.

- `A`: Matriz de coeficientes del sistema lineal generado por el método de diferencias finitas.
- `n`: Número de divisiones espaciales (nodos menos uno).
- `h`: Tamaño de paso espacial, calculado como $(2-0)/n$.
- `x`: Vector de posiciones espaciales.
- `w0`: Vector de condiciones iniciales en los nodos internos.
- `B`: Matriz que almacena la evolución de la solución en cada paso temporal.
- `w1`: Vector de solución en el siguiente paso temporal.

Programa en Matlab/Octave:

```
la=(4*1/65)/(0.5^2);A=[];n=4;h=(2-0)/n;
A(1,1)=1-2*la;A(1,2)=la;
for i=2:n-2
    A(i,i-1)=la;A(i,i)=1-2*la;A(i,i+1)=la;
end
A(n-1,n-2)=la;A(n-1,n-1)=1-2*la;
x=0:h:2;
w0=x.^2-2*x;w0=w0';B=[];w0=w0(2:n);B(:,1)=w0;
for i=1:10
    w1=A*w0
    B(:,i+1)=w1;
    w0=w1;
end
mesh(B)
```

Manual:
Para ejecutar el programa, siga estos pasos:

1. Abra Matlab o Octave en su computadora.
2. Copie el código proporcionado en una nueva ventana de script y guárdelo como `dif_finitas.m`.
3. Ejecute el script presionando el botón de ejecución o escribiendo `dif_finitas` en la consola.
4. Observe la salida numérica en la consola, que muestra la evolución de la solución en cada paso temporal.
5. Se generará una gráfica de tipo `mesh` que representa la evolución de la solución en la malla discreta.

Asegúrese de tener instalado Matlab o GNU Octave y de que el archivo se encuentre en el directorio de trabajo. **Corrida del Programa:**

```
w1 =

  -0.6269
  -0.8769
  -0.6269

w1 =

  -0.5341
  -0.7538
  -0.5341

w1 =

  -0.4567
```

```
  -0.6457
  -0.4567

w1 =

  -0.3908
  -0.5527
  -0.3908

w1 =

  -0.3345
  -0.4730
  -0.3345

w1 =

  -0.2862
  -0.4048
  -0.2862

w1 =

  -0.2450
  -0.3464
  -0.2450

w1 =

  -0.2096
  -0.2965
  -0.2096

w1 =

  -0.1794
  -0.2537
  -0.1794

w1 =

  -0.1535
  -0.2171
  -0.1535
```

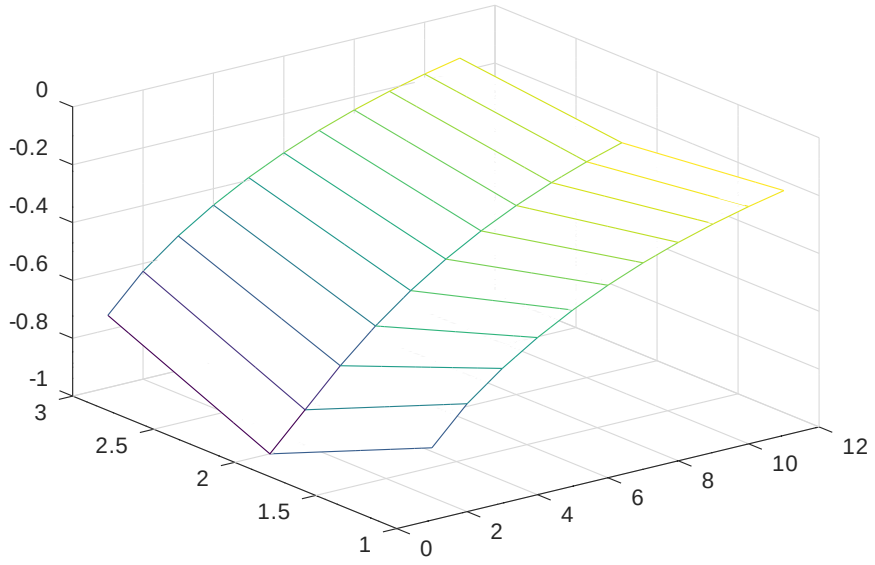


Figura 7: Solución de la EDP

11. Colocación Base

11.1. Programa 1

Este informe presenta la resolución de una ecuación diferencial ordinaria (EDO) de segundo orden utilizando el método de colocación de base. Se emplean funciones base y sus derivadas para aproximar la solución, implementando el procedimiento en Matlab/Octave.
Variables:

- `u1, u2`: Funciones base utilizadas para aproximar la solución.
- `du1, du2`: Derivadas de las funciones base.
- `ddu1, ddu2`: Segundas derivadas de las funciones base.
- `p, q`: Coeficientes de la EDO.
- `A`: Matriz de coeficientes del sistema lineal.
- `b`: Vector de términos independientes.
- `x`: Puntos de colocación y vector de evaluación.
- `c`: Coeficientes de la combinación lineal de las bases.
- `y`: Aproximación de la solución de la EDO.

Programa en Matlab/Octave:

```
%19/06/2025
%colocacion base
%bases
u1=@(x) (1-x).*x;
u2=@(x) u1(x).*x;
%derivadas
du1=@(x) (1-2*x);
du2=@(x) 2*x-3*x.*x;
%segundas derivadas
```

```
ddu1=@(x) -2;
ddu2=@(x) 2-6*x;
%y''+y'-2y=x y(0)=0=y(1)
p=1;q=-2;
A=[];b=[];x=[1/2 4/3];
A(1,1)=ddu1(x(1))+p*du1(x(1))+q*u1(x(1));
A(1,2)=ddu2(x(1))+p*du2(x(1))+q*u2(x(1));
A(2,1)=ddu1(x(2))+p*du1(x(2))+q*u1(x(2));
A(2,2)=ddu2(x(2))+p*du2(x(2))+q*u2(x(2));
b(1)=x(1);b(2)=x(2);
c=inv(A)*b';
x=0:0.01:1;
y=c(1)*u1(x)+c(2)*u2(x);
plot(x,y)
hold on
%tarea, hacer que funciones para {1/4 1/2 3/4}
%u3=(1-x)x^3
```

Manual:
Para ejecutar el programa en Matlab o Octave, siga estos pasos:

1. Copie el código proporcionado en una nueva ventana de script.
2. Guarde el archivo con extensión `.m`, por ejemplo, `colocacion_base.m`.
3. Ejecute el script en el entorno Matlab/Octave. El programa calculará la aproximación de la solución de la EDO y mostrará la gráfica correspondiente.
4. Puede modificar los puntos de colocación en el vector `x` y agregar nuevas funciones base para experimentar con diferentes aproximaciones.

El gráfico generado muestra la solución aproximada obtenida mediante el método de colocación de base.
Corrida del Programa:

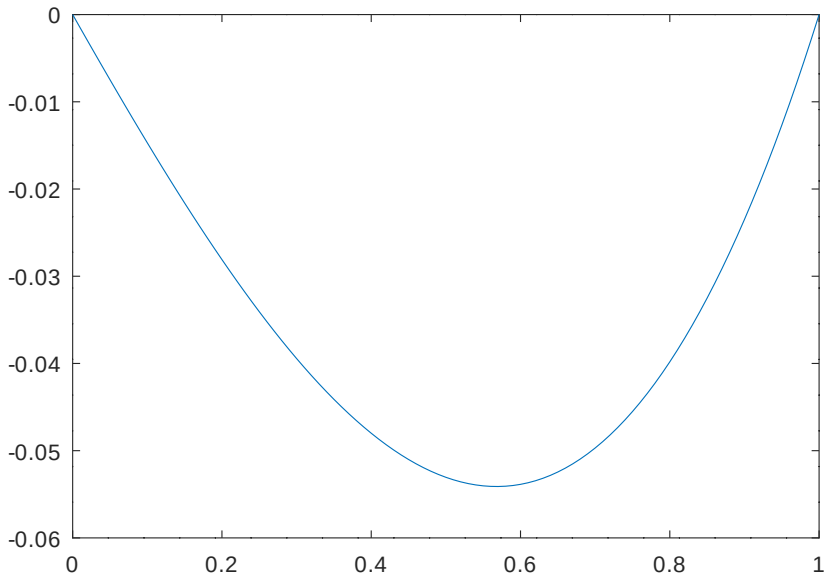


Figura 8: Solución de la EDO

11.2. Programa 2

Este informe presenta la resolución de una ecuación diferencial ordinaria (EDO) utilizando el método de colocación con funciones base polinomiales. Se desarrolla el procedimiento teórico y se implementa el algoritmo en Matlab/Octave para obtener la solución aproximada.
Variables:

- $u_1(x), u_2(x)$: Funciones base utilizadas en el método de colocación.
- $du_1(x), du_2(x)$: Derivadas de las funciones base.
- $ddu_1(x), ddu_2(x)$: Segundas derivadas de las funciones base.
- A: Matriz de coeficientes del sistema lineal generado por el método.
- b: Vector de términos independientes.
- p, q: Coeficientes de la EDO.
- c: Vector de coeficientes de la combinación lineal de las funciones base.
- x: Vector de puntos en el intervalo de solución.
- y: Solución aproximada de la EDO en los puntos x.

Programa en Matlab/Octave:

```
%19/06/2025
u1=@(x) (1-x).*x;u2=@(x) u1(x).*x;
du1=@(x) (1-2.*x);du2=@(x) 2.*x-3.*x.*x;
ddu1=@(x) -2;ddu2=@(x) 2-6.*x;
A=[]; b=[]; p=1; q=-2;
pp=@(x) (ddu1(x)+p.*du1(x)+q.*u1(x)).*u1(x);
A(1,1)=quad(pp,0,1);
pp=@(x) (ddu2(x)+p.*du2(x)+q.*u2(x)).*u1(x);
A(1,2)=quad(pp,0,1);
pp=@(x) (ddu1(x)+p.*du1(x)+q.*u1(x)).*u2(x);
A(2,1)=quad(pp,0,1);
pp=@(x) (ddu2(x)+p.*du2(x)+q.*u2(x)).*u2(x);
A(2,2)=quad(pp,0,1);
pp=@(x) x.*u1(x);
b(1)=quad(pp,0,1);
pp=@(x) x.*u2(x);
b(2)=quad(pp,0,1);
c=A\b';
x=0:0.001:1;
y=c(1).*u1(x)+c(2).*u2(x);
plot(x,y,'m');
grid;
```

Manual:

1. Definir las funciones base $u_1(x)$ y $u_2(x)$, junto con sus derivadas primeras y segundas.
2. Plantear la ecuación diferencial ordinaria y los coeficientes p y q.
3. Construir la matriz de coeficientes A y el vector de términos independientes b mediante la integración de los productos adecuados.
4. Resolver el sistema lineal $Ac = b$ para obtener los coeficientes c de la solución aproximada.
5. Evaluar la solución aproximada $y(x)$ en el intervalo deseado utilizando la combinación lineal de las funciones base.
6. Graficar la solución obtenida para visualizar el comportamiento de la EDO resuelta.

Corrida del Programa:

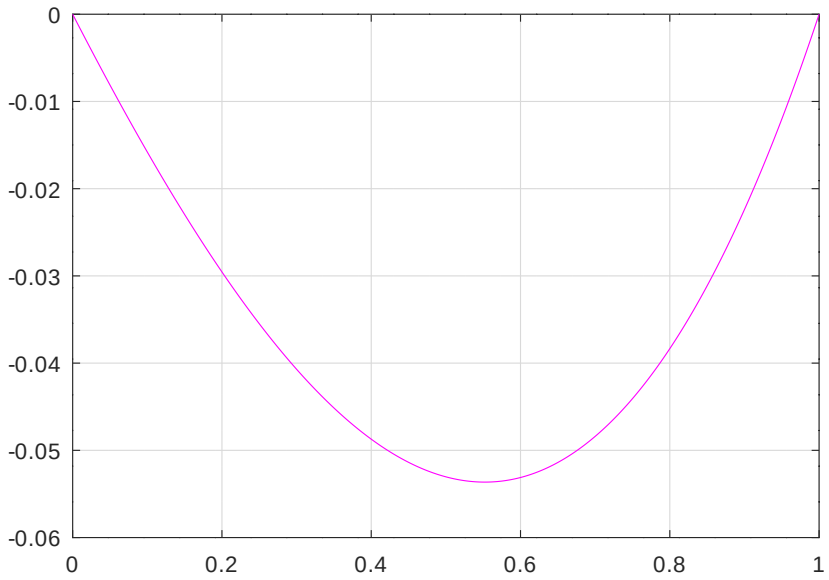


Figura 9: Solución de la EDO

12. Rayleigth-Ritz

12.1. Programa 1

El método de Rayleigh-Ritz es una técnica aproximada utilizada para resolver ecuaciones diferenciales, especialmente en problemas de valores propios y en mecánica estructural. Consiste en aproximar la solución mediante funciones de prueba y minimizar una forma funcional asociada al problema, obteniendo así un sistema de ecuaciones algebraicas que se puede resolver numéricamente.
Variables:

- `p, q, r`: Funciones que definen la ecuación diferencial.
- `x`: Vector de nodos en el intervalo de solución.
- `h`: Tamaño de subintervalo.
- `kk`: Matriz de coeficientes del sistema lineal.
- `b`: Vector de términos independientes.
- `c`: Solución aproximada en los nodos internos.
- `yl`: Vector de la solución en todos los nodos (incluyendo condiciones de frontera).
- `y`: Solución analítica para comparación.

Programa en Matlab/Octave:

```
%-(1*y')'+2*y=-x x\in[0,1] y(0)=0=y(1)
p=@(x) 1; q=@(x) 2; r=@(x) -x;
x=[0 1/3 2/3 1]; h=1/3;

n=length(x)-2;
Ipl=@(t) p(t);Ipl=quad(Ipl,x(1),x(2))/h^2;
Iql=@(t) (t-x(1)).^2.*q(t);Iql=quad(Iql,x(1),x(2))/h^2;
for i=1:n-1
    IpN=@(t) p(t);IpN=quad(IpN,x(i),x(i+1))/h^2;
    IqN=@(t) q(t).*(t-x(i+1)).^2;IqN=quad(IqN,x(i),x(i+1))/h^2;
    Ia=@(t) (x(i+2)-t).^2*q(t);Ia=quad(Ia,x(i+1),x(i+2))/h^2;
```

```
    Ib=@(t) (x(i+2)-t).*(t-x(i+1)).*q(t);Ib=quad(Ib,x(i+1),x(i+2))/h^2;
    Ic=@(t) (t-x(i)).*r(t);Ic=quad(Ic,x(i),x(i+1))/h;
    Id=@(t) (x(i+2)-t).*r(t);Id=quad(Id,x(i+1),x(i+2))/h;
    kk(i,i)=Ipl+IpN+Iql+Ia;kk(i,i+1)=-IpN+Ib;
    kk(i+1,i)=kk(i,i+1);b(i)=Ic+Id;
    Ipl=IpN;Iql=IqN;
end

i=n-1;
IpN=quad(p,x(i+2),x(i+3))/h^2;
Ia=@(t) (x(i+3)-t).^2*q(t);Ia=quad(Ia,x(i+2),x(i+3))/h^2;
Ic=@(t) (t-x(i+1)).*r(t);Ic=quad(Ic,x(i+1),x(i+2))/h;
Id=@(t) (x(i+3)-t).*r(t);Id=quad(Id,x(i+2),x(i+3))/h;
kk(i+1,i+1)=Ipl+IpN+Iql+Ia;
b(2)=Ic+Id;
[kk b']
c=kk\b'
yl=[0 c' 0];
t=0:0.01:1;
y=@(x) sinh(sqrt(2)*x)/(2*sinh(sqrt(2)))-x/2;
plot(x,yl,t,y(t))
```

Manual:

1. Copie el código Matlab/Octave en un archivo llamado `rayleigh_ritz.m`.
2. Ejecute el archivo en Matlab o GNU Octave.
3. Verifique que los resultados impresos y la gráfica coincidan con los valores esperados.
4. Compare la solución numérica (`yl`) con la solución analítica (`y(x)`).
5. Modifique los valores de `x` o las funciones `p`, `q`, `r` para experimentar con otros problemas.

Corrida del Programa:

```
ans =

   6.4444  -2.8889  -0.1111
  -2.8889   6.4444  -0.2222

c =

  -0.040923
  -0.052827
```

12.2. Programa 2

El método de Rayleigh-Ritz es una técnica variacional utilizada para aproximar soluciones de ecuaciones diferenciales, especialmente en problemas de valores propios y en física matemática. Consiste en buscar una solución aproximada como combinación lineal de funciones base, minimizando una forma funcional asociada al problema.

Variables:

- `p, q, r`: Funciones coeficientes de la ecuación diferencial.
- `n`: Número de subintervalos para la discretización.
- `x`: Vector de puntos en el intervalo $[0, 1]$.
- `h`: Tamaño de cada subintervalo.

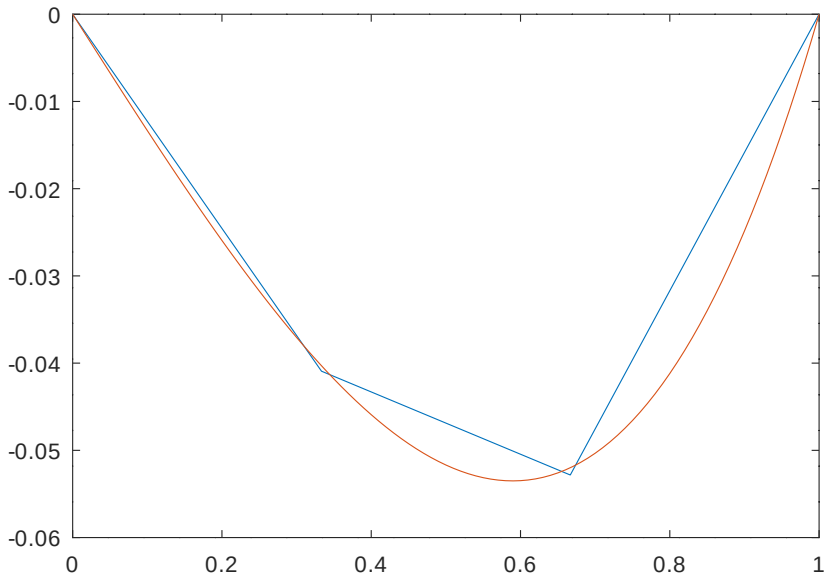


Figura 10: Solución de la EDO

- `kk`: Matriz de coeficientes del sistema lineal.
- `b`: Vector de términos independientes.
- `c`: Solución del sistema, coeficientes de la aproximación.
- `yl`: Vector de valores aproximados de la solución en los nodos.
- `y`: Solución analítica para comparación.

Programa en Matlab/Octave:

```
%rayleigth Ritz
%- (1+y')'+2*y=x x\in[0 1] y(0)=0=y(1)
p=@(x) 1; q=@(x) 2; r=@(x) -x; n=5;
x=0:1/n:1; h=1/n;
n=length(x)-2;
Ip1=@(t) p(t);Ip1=quad(Ip1,x(1),x(2))/h^2;
Iq1=@(t) (t-x(1)).^2.*q(t);Iq1=quad(Iq1,x(1),x(2))/h^2;
for i=1:n-1
    IpN=@(t) p(t);IpN=quad(IpN,x(i+1),x(i+2))/h^2;
    IqN=@(t) q(t).*(t-x(i+1)).^2;IqN=quad(IqN,x(i+1),x(i+2))/h^2;
    Ia=@(t) (x(i+2)-t).^2*q(t);Ia=quad(Ia,x(i+1),x(i+2))/h^2;
    Ib=@(t) (x(i+2)-t).*(t-x(i+1)).*q(t);Ib=quad(Ib,x(i+1),x(i+2))/h^2;
    Ic=@(t) (t-x(i)).*r(t);Ic=quad(Ic,x(i),x(i+1))/h;
    Id=@(t) (x(i+2)-t).*r(t);Id=quad(Id,x(i+1),x(i+2))/h;
    kk(i,i)=Ip1+IpN+Iq1+Ia;kk(i,i+1)=-IpN+Ib;
    kk(i+1,i)=kk(i,i+1);b(i)=Ic+Id;
    Ipl=IpN;Iql=IqN;
end
i=n-1;
IpN=quad(p,x(i+2),x(i+3))/h^2;
Ia=@(t) (x(i+3)-t).^2*q(t);Ia=quad(Ia,x(i+2),x(i+3))/h^2;
Ic=@(t) (t-x(i+1)).*r(t);Ic=quad(Ic,x(i+1),x(i+2))/h;
Id=@(t) (x(i+3)-t).*r(t);Id=quad(Id,x(i+2),x(i+3))/h;
kk(i+1,i+1)=Ip1+IpN+Iq1+Ia;
```

```
b(i+1)=Ic+Id;
[kk b']
c=kk\b'
yl=[0 c' 0];
t=0:0.01:1;
y=@(x) sinh(sqrt(2)*x)/(2*sinh(sqrt(2)))-x/2;
plot(x,yl,t,y(t))
```

Manual:
Para ejecutar el programa, copie el código Matlab/Octave en un archivo llamado 'rayleigh_ritz.m' y ejecútelo en el entorno de Octave o Matlab. El programa calcula la aproximación de la solución de la ecuación diferencial usando el método de Rayleigh-Ritz, mostrando la matriz del sistema, los coeficientes obtenidos y la comparación gráfica entre la solución aproximada y la analítica.
Pasos:

1. Defina las funciones coeficientes $p(x)$, $q(x)$ y $r(x)$ según el problema.
2. Establezca el número de subintervalos n para la discretización.
3. Ejecute el script para obtener los coeficientes y la gráfica.
4. Compare los resultados numéricos y gráficos con la solución analítica.

Corrida del Programa:

```
ans =

   10.2667   -4.9333         0         0   -0.0400
   -4.9333   10.2667   -4.9333         0   -0.0800
         0   -4.9333   10.2667   -4.9333   -0.1200
         0         0   -4.9333   10.2667   -0.1600

c =

  -0.026079
  -0.046165
  -0.053777
  -0.041425
```

13. Crank-Nicolson para una EDP en especifico

El método Crank-Nicolson es un esquema numérico implícito de segundo orden para resolver ecuaciones diferenciales parciales, especialmente la ecuación de difusión o calor. Combina los métodos de Euler hacia adelante y hacia atrás, proporcionando mayor estabilidad y precisión.
Variables:

- a: coeficiente de difusión (en este caso $a = -1$).
- h: tamaño de paso espacial.
- k: tamaño de paso temporal.
- x: vector de posiciones espaciales.
- $w0$: condición inicial de la solución.
- BB: matriz que almacena las soluciones en cada paso temporal.
- L: parámetro auxiliar para el método.
- A, B: matrices del sistema lineal generado por el método.

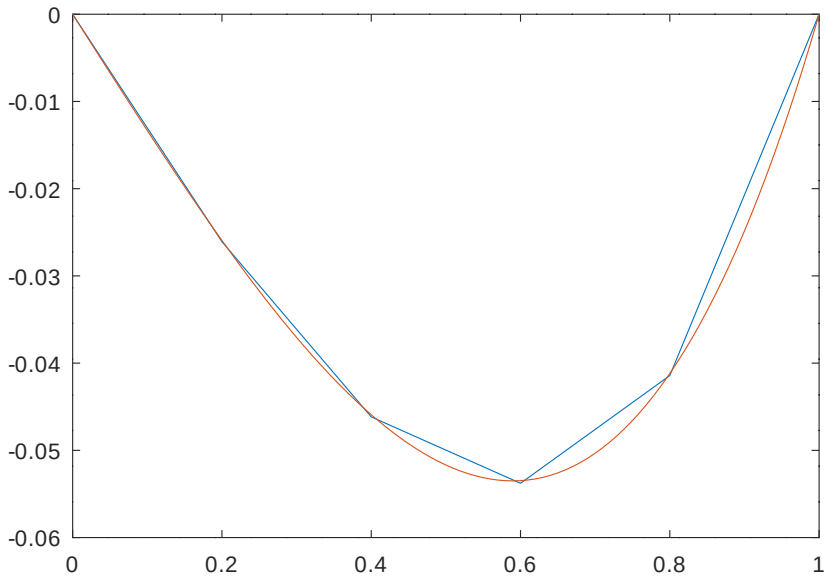


Figura 11: Solución de la EDO

- *wi*: vector auxiliar para la iteración.
- *cc*: solución del sistema en cada paso temporal.

Programa en Matlab/Octave:

```
%07/07/2025
% Crank-Nicolson para un problema en especifico
%u_t=1*u_xx, u(t,0)=u(t,1) , u(0,x)=x-x^2
a=-1;h=1/4;k=(1/5);
format rat
BB=[];
x=0:1/4:1;
w0=x-x.^2;
w0=w0(2:4)';
BB(:,1)=[0 w0' 0]';
L=a^2*k/h^2;
A=[1+L -L/2 0;-L/2 1+L -L/2;0 -L/2 1+L]
B=[1-L L/2 0;L/2 1-L L/2;0 L/2 1-L]
%Mecanica de solucion
wi=B*w0;
format
cc=A\wi;w0=cc;BB(:,2)=[0 w0' 0]';
wi=B*w0;
format
cc=A\wi;w0=cc;BB(:,3)=[0 w0' 0]';
wi=B*w0;
format
cc=A\wi;w0=cc;BB(:,4)=[0 w0' 0]';
wi=B*w0;
format
cc=A\wi;w0=cc;BB(:,5)=[0 w0' 0]';
wi=B*w0;
format
cc=A\wi;w0=cc;BB(:,6)=[0 w0' 0]';
```

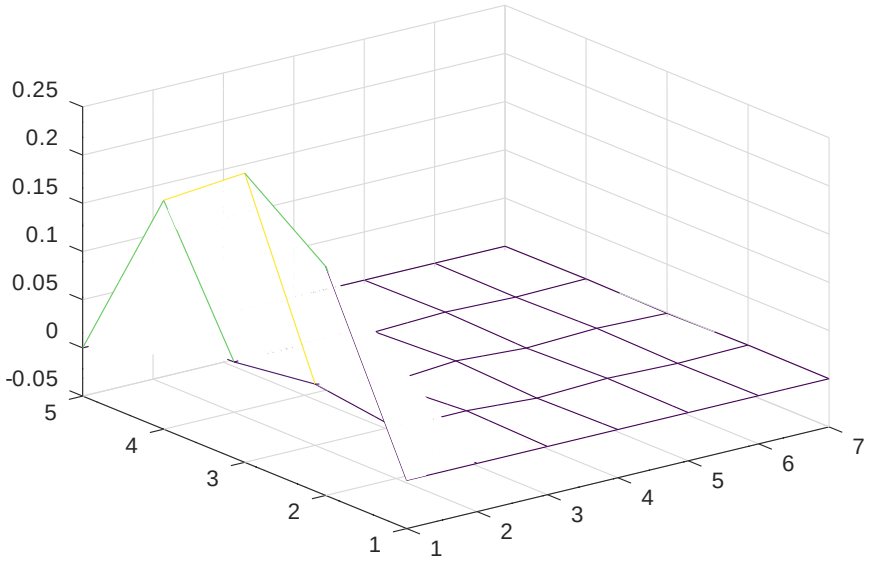


Figura 12: Solución de la EDP

```
wi=B*w0;
format
cc=A\wi;w0=cc;BB(:,7)=[0 w0' 0]';
mesh(BB)
```

Manual:
Para ejecutar el programa, siga estos pasos:
1. Abra Matlab u Octave y copie el código proporcionado en un archivo llamado 'crank_nicolson.m'. 2. Ejecute el archivo en la consola con el comando 'crank_nicolson'. 3. El programa calculará la solución numérica de la ecuación de difusión usando el método Crank-Nicolson y mostrará la evolución de la solución en una gráfica de malla ('mesh'). 4. Los resultados intermedios de las matrices A y B se muestran en la salida, y la figura generada ('im11.pdf') representa la solución aproximada en los distintos pasos temporales.
Asegúrese de tener instalado Matlab u Octave y de que el archivo se encuentre en el directorio de trabajo.
Corrida del Programa:

```
A =

       21/5       -8/5          0
       -8/5       21/5       -8/5
          0       -8/5       21/5

B =

      -11/5        8/5          0
        8/5      -11/5        8/5
          0        8/5      -11/5
```

14. Crank-Nicolson para una EDP en general con $\theta = 1/2$

El método Crank-Nicolson es un esquema numérico implícito para resolver ecuaciones en derivadas parciales (EDP) de tipo parabolico, como la ecuación de difusión o calor. Es una combinación de los métodos explícito e implícito, usando el promedio entre ambos en cada paso temporal. Para $\theta = 1/2$, se obtiene el método clásico de Crank-Nicolson, que es estable y de segundo orden en el tiempo y espacio.
Variables:

- a, b: Extremos del intervalo espacial.
- n: Número de subintervalos espaciales.
- h: Tamaño de paso espacial, $h = (b - a)/n$.
- tt: Tiempo final de la simulación.
- k: Tamaño de paso temporal.
- α: Parámetro de difusión (α^2).
- L: Número de Fourier, $L = \alpha k/h^2$.
- x: Vector de nodos espaciales.
- $w0$: Condición inicial en los nodos.
- A, B: Matrices del sistema lineal para el método.
- BB: Matriz que almacena la solución en cada paso temporal.
- m: Número de pasos temporales.

Programa en Matlab/Octave:

```
%Crank_nicolson general para theta=1/2
a=0;b=1;n=10;h=(b-a)/n;
tt=1;k=1/12; alp=1; %alp: alpha cuadrado
L=alp*k/h^2;BB=A=B=[];bb=[];
x=a:h:b;w0=x-x.^2; BB(:,1)=w0;
w0=w0(2:n)';
A(1,1)=1+L;A(1,2)=-L/2;
B(1,1)=1-L;B(1,2)=L/2;
for i=2:n-2
    A(i,i-1)=-L/2;A(i,i)=1+L;A(i,i+1)=-L/2;
    B(i,i-1)=L/2;B(i,i)=1-L;B(i,i+1)=L/2;
end
i=n-1;
A(i,i-1)=-L/2;A(i,i)=1+L;
B(i,i-1)=L/2;B(i,i)=1-L;
m=ceil(tt/k);
for i=1:m
    wx=B*w0;wx=A\wx;BB(:,i+1)=[0 wx' 0]';
    w0=wx;
end
BB
mesh(BB)
```

Manual:
Para ejecutar el programa, siga estos pasos:
1. Copie el código Matlab/Octave en un archivo llamado ‘crank_nicolson.m‘. 2. Abra Matlab o GNU Octave y navegue hasta el directorio donde guardó el archivo. 3. Ejecute el archivo escribiendo ‘crank_nicolson‘ en la consola. 4. El resultado será la matriz ‘BB‘, que contiene la solución numérica en cada paso temporal, y una gráfica de la evolución de la solución.
La matriz ‘BB‘ muestra cómo la condición inicial evoluciona en el tiempo bajo el método Crank-Nicolson. La gráfica generada (‘mesh(BB)‘) permite visualizar la difusión de la solución en el dominio espacio-tiempo. **Corrida del Programa:**

```
BB =

 Columns 1 through 8:
```

```
         0         0         0         0         0         0         0         0
    0.0900    0.0272    0.0181    0.0033    0.0043   -0.0002    0.0014   -0.0005
    0.1600    0.0594    0.0290    0.0102    0.0052    0.0019    0.0008    0.0005
    0.2100    0.0875    0.0368    0.0158    0.0062    0.0030    0.0009    0.0007
    0.2400    0.1061    0.0419    0.0191    0.0073    0.0034    0.0013    0.0006
    0.2500    0.1126    0.0437    0.0201    0.0077    0.0035    0.0015    0.0005
    0.2400    0.1061    0.0419    0.0191    0.0073    0.0034    0.0013    0.0006
    0.2100    0.0875    0.0368    0.0158    0.0062    0.0030    0.0009    0.0007
    0.1600    0.0594    0.0290    0.0102    0.0052    0.0019    0.0008    0.0005
    0.0900    0.0272    0.0181    0.0033    0.0043   -0.0002    0.0014   -0.0005
         0         0         0         0         0         0         0         0

Columns 9 through 13:

         0         0         0         0         0
    0.0006   -0.0004    0.0003   -0.0003    0.0002
    0.0000    0.0002   -0.0001    0.0001   -0.0001
    0.0000    0.0002   -0.0001    0.0001   -0.0000
    0.0002    0.0001    0.0001   -0.0000    0.0000
    0.0003    0.0000    0.0001   -0.0000    0.0000
    0.0002    0.0001    0.0001   -0.0000    0.0000
    0.0000    0.0002   -0.0001    0.0001   -0.0000
    0.0000    0.0002   -0.0001    0.0001   -0.0001
    0.0006   -0.0004    0.0003   -0.0003    0.0002
         0         0         0         0         0
```

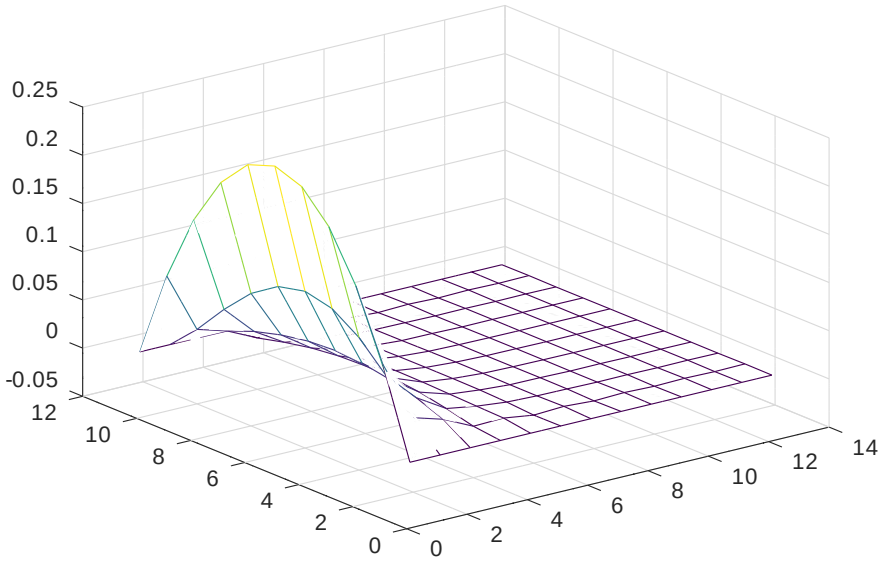


Figura 13: Solución de la EDP con Crank-Nicolson