# CAMILO_HURTADO_Tarea9

May 1, 2020

CAMILO HURTADO

Ecuaciones diferenciales parciales

Universidad Sergio Arboleda

Maestría en matemáticas aplicadas, 4° semestre

TAREA 9

PROFESOR: Alejandro Garzón Leyton

Ejercicio tomado de *Introduction to Partial Differential Equations* de Peter J. Olver, Springer International Publishing (2014)

## 1 Punto 5.2.2.

Solve the following initial-boundary value problem: $u_t = u_{xx}$ , $u(t,0) = u(t,1) = 0$ , $u(0,x) = f(x)$ , $0 \leq x \leq 1$ ,

with initial data

$$f(x) = \begin{cases} 2|x - \frac{1}{6}| - \frac{1}{3}, & 0 \leq x \leq \frac{1}{3} \\ 0, & \frac{1}{3} \leq x \leq \frac{2}{3} \\ \frac{1}{2} - 3|x - \frac{5}{6}| & \frac{2}{3} \leq x \leq 1 \end{cases}$$

using:

- The explicit scheme (5.14)
- The implicit scheme (5.29)
- The Crank - Nicolson scheme (5.32)

Use space step sizes $\Delta x = 0.1$ and $0.05$, and suitably chosen time steps $\Delta t$.

### 1.1 Método explícito

[224]:
```
import numpy as np
import matplotlib.pyplot as plt
```

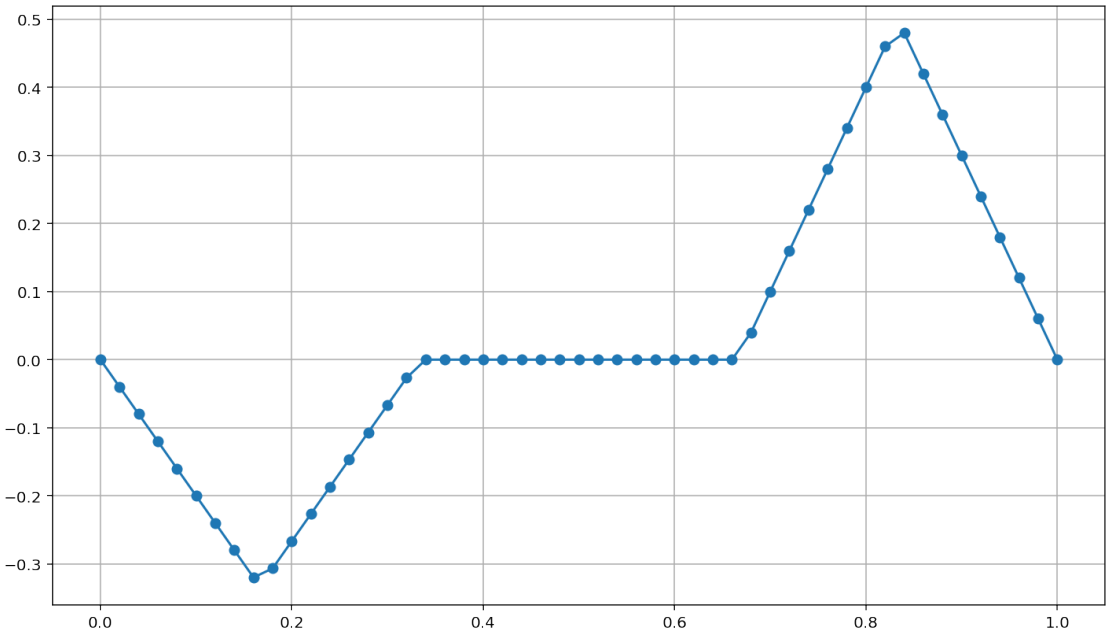
[225]:
```
#definimos las condiciones iniciales
def f(x):
    if x < 1/3:
```

```
        return 2*np.abs(x - 1/6)-1/3
    elif x < 2/3:
        return 0
    else:
        return 1/2 -3*np.abs(x-5/6)
```

[226]:
```
N = 50
x = np.linspace(0,1,N+1)
```

[227]:
```
#Graficamos la condición inicial
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
```

[227]:




### 1.1.1 Con un $\Delta x = 0.1$

[228]:
```
#Definimos el tamaño de los pasos en tiempo y espacio
deltax = 0.1
deltat = deltax**2/2
print("El valor de deltat es", deltat)
```

```
El valor de deltat es 0.005000000000000001
```

[229]:
```
#Construimos la matriz A
deltax = 0.1
deltat = deltax**2/2
```

```
mu = deltat/deltax**2
print("El valor de mu es ", mu)
A = np.zeros([N-1,N-1])
np.fill_diagonal(A,1-2*mu)
for i in range(1,N-1):
    A[i,i-1] = mu
    A[i-1,i] = mu
print("La matriz A es ", A)
```

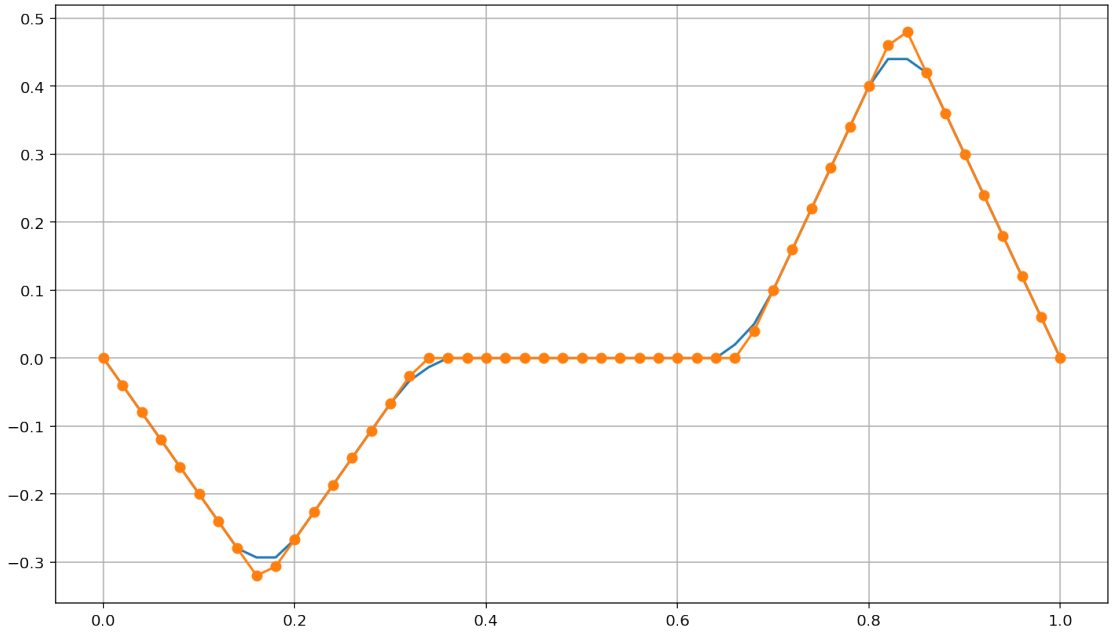
```
El valor de mu es  0.5
La matriz A es  [[0.  0.5 0.  … 0.  0.  0. ]
 [0.5 0.  0.5 … 0.  0.  0. ]
 [0.  0.5 0.  … 0.  0.  0. ]
 …
 [0.  0.  0.  … 0.  0.5 0. ]
 [0.  0.  0.  … 0.5 0.  0.5]
 [0.  0.  0.  … 0.  0.5 0. ]]
```

[230]:
```
#Creamos el vector de las temperaturas
u = np.vectorize(f)(x)
```

**Tiempo final = 0.01**

[231]:
```
#Creamos el vector de las temperaturas
u = np.vectorize(f)(x)
deltax = 0.1
deltat = deltax**2/2
#Graficamos la función de la temperatura, almacenando solo sus valores en el␣
 ↪tiempo final
tiempofinal = 0.01
nt = int(tiempofinal/deltat)
for i in range(nt):
    u[1:N] = A.dot(u[1:N])
plt.plot(x,u)
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
```

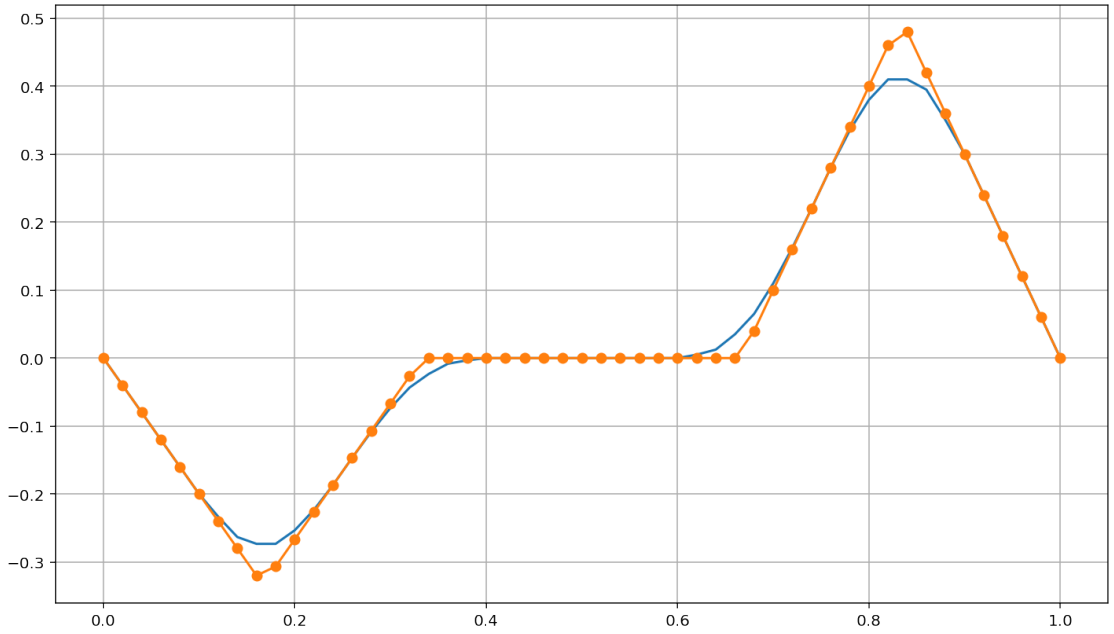
[231]:




**Tiempo final = 0.02**

[232]:
```
#Creamos el vector de las temperaturas
u = np.vectorize(f)(x)
deltax = 0.1
deltat = deltax**2/2
#Graficamos la función de la temperatura, almacenando solo sus valores en el␣
 ↪tiempo final
tiempofinal = 0.02
nt = int(tiempofinal/deltat)
for i in range(nt):
    u[1:N] = A.dot(u[1:N])
plt.plot(x,u)
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
```

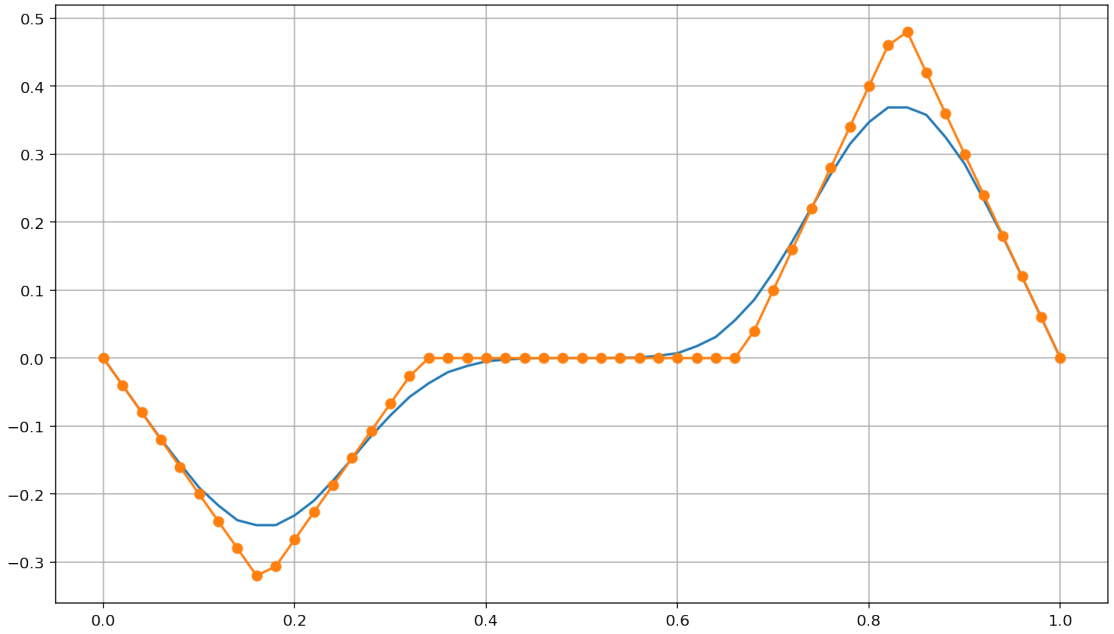
[232]:




**Tiempo final = 0.04**

[233]:
```
#Creamos el vector de las temperaturas
u = np.vectorize(f)(x)
deltax = 0.1
deltat = deltax**2/2
#Graficamos la función de la temperatura, almacenando solo sus valores en el␣
 ↪tiempo final
tiempofinal = 0.04
nt = int(tiempofinal/deltat)
for i in range(nt):
    u[1:N] = A.dot(u[1:N])
plt.plot(x,u)
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
```

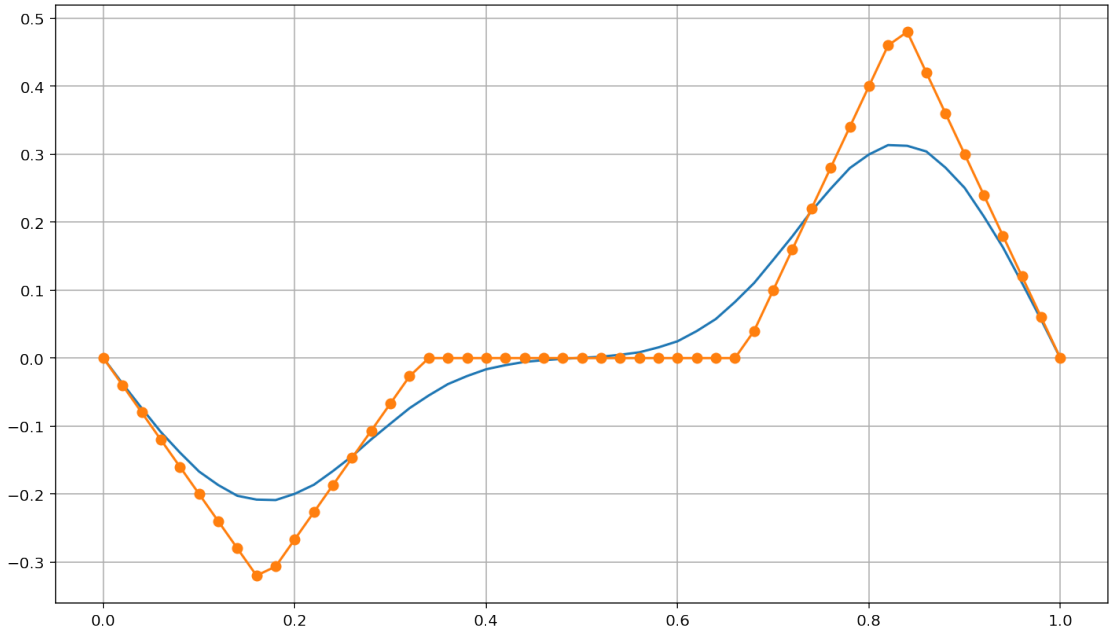
[233]:




**Tiempo final = 0.08**

[234]:
```
#Creamos el vector de las temperaturas
u = np.vectorize(f)(x)
deltax = 0.1
deltat = deltax**2/2
#Graficamos la función de la temperatura, almacenando solo sus valores en el␣
 ↪tiempo final
tiempofinal = 0.08
nt = int(tiempofinal/deltat)
for i in range(nt):
    u[1:N] = A.dot(u[1:N])
plt.plot(x,u)
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
```

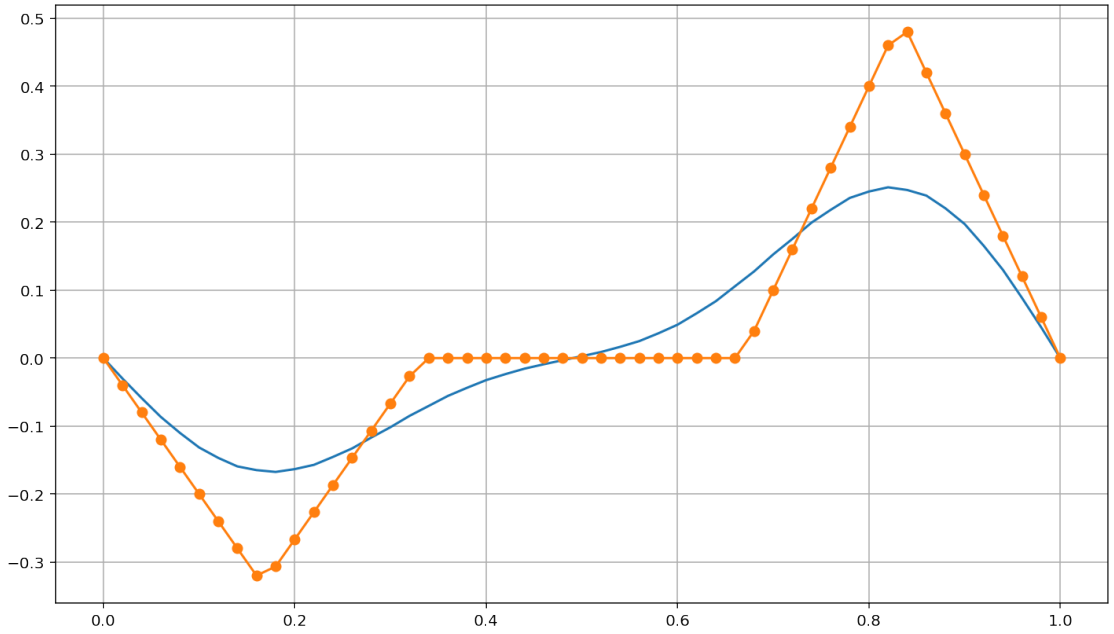
[234]:




**Tiempo final = 0.15**

[235]:
```
#Creamos el vector de las temperaturas
u = np.vectorize(f)(x)
deltax = 0.1
deltat = deltax**2/2
#Graficamos la función de la temperatura, almacenando solo sus valores en el␣
 ↪tiempo final
tiempofinal = 0.15
nt = int(tiempofinal/deltat)
for i in range(nt):
    u[1:N] = A.dot(u[1:N])
plt.plot(x,u)
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
```

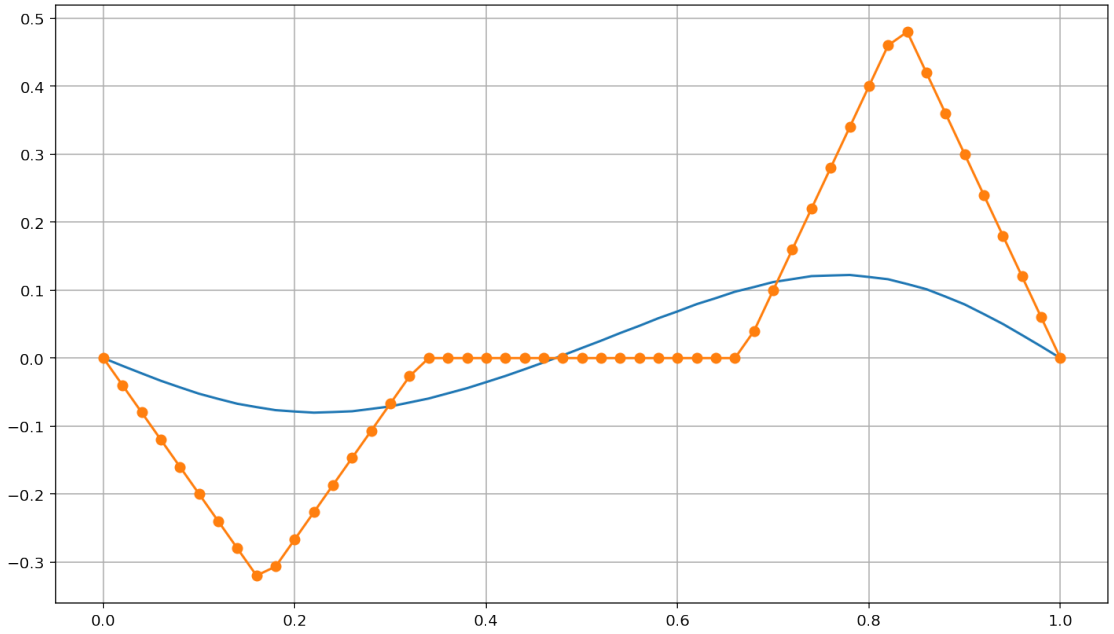
[235]:




**Tiempo final = 0.5**

[236]:
```
#Creamos el vector de las temperaturas
u = np.vectorize(f)(x)
deltax = 0.1
deltat = deltax**2/2
#Graficamos la función de la temperatura, almacenando solo sus valores en el␣
 ↪tiempo final
tiempofinal = 0.5
nt = int(tiempofinal/deltat)
for i in range(nt):
    u[1:N] = A.dot(u[1:N])
plt.plot(x,u)
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
```

[236]:

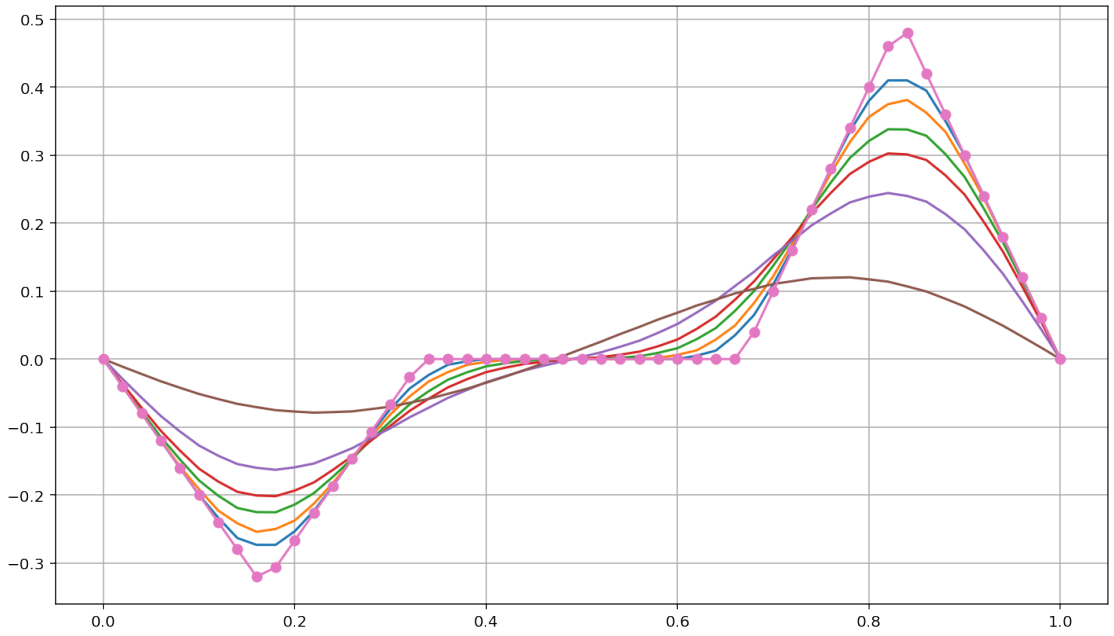



**Tiempo final = 1**

[237]:
```
#Creamos el vector de las temperaturas
u = np.vectorize(f)(x)
deltax = 0.1
deltat = deltax**2/2
#Graficamos la función de la temperatura, almacenando solo sus valores en el␣
 ↪tiempo final
tiempofinal = 1
nt = int(tiempofinal/deltat)
listatiempos = np.array([0.01, 0.025, 0.05, 0.08, 0.15, 0.5, 1])
#print(len(listatiempos))
lista_enteros = (listatiempos/deltat)
lista_enteros = [round(x) for x in lista_enteros]
#print(lista_enteros)
count = 0
for i in range(nt+1):
    u[1:N] = A.dot(u[1:N])
    if i == lista_enteros[count]:
        count = count + 1
        plt.plot(x,u)
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
print("Numero de pasos para llegar al segundo 1:", nt)
```

```
Numero de pasos para llegar al segundo 1: 199
```

[237]:




### 1.1.2 Con un $\Delta x = 0.05$

[238]:
```
#Definimos el tamaño de los pasos en tiempo y espacio
deltax = 0.05
deltat = deltax**2/2
print("El valor de deltat es", deltat)
```

```
El valor de deltat es 0.0012500000000000002
```

[239]:
```
#Construimos la matriz A
deltax = 0.05
deltat = deltax**2/2
mu = deltat/deltax**2

print("El valor de mu es ", mu)
A = np.zeros([N-1,N-1])
np.fill_diagonal(A,1-2*mu)
for i in range(1,N-1):
    A[i,i-1] = mu
    A[i-1,i] = mu
print("La matriz A es ", A)
```

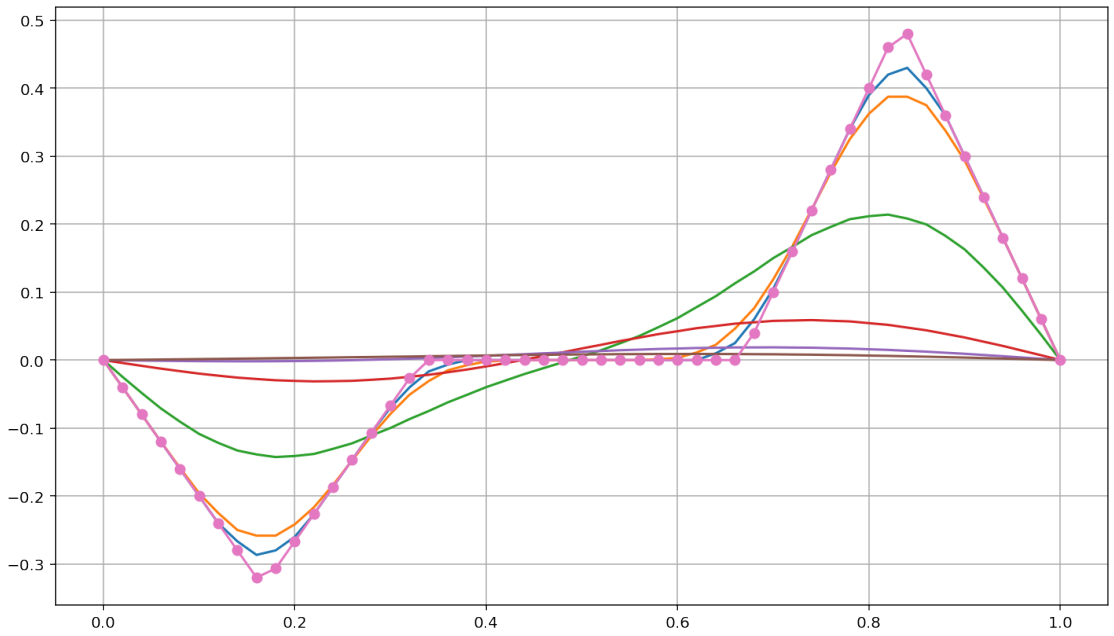
```
El valor de mu es  0.5
La matriz A es  [[0.  0.5 0.  … 0.  0.  0. ]
 [0.5 0.  0.5 … 0.  0.  0. ]
```

```
[0.  0.5 0.  … 0.  0.  0. ]
…
[0.  0.  0.  … 0.  0.5 0. ]
[0.  0.  0.  … 0.5 0.  0.5]
[0.  0.  0.  … 0.  0.5 0. ]]
```

[240]:
```
#Creamos el vector de las temperaturas
u = np.vectorize(f)(x)
deltax = 0.05
deltat = deltax**2/2
#Graficamos la función de la temperatura, almacenando solo sus valores en el␣
 ↪tiempo final
tiempofinal = 1
nt = int(tiempofinal/deltat)
listatiempos = np.array([0.00125, 0.005, 0.05, 0.25, 0.5, 0.75, 1])
lista_enteros = (listatiempos/deltat)
lista_enteros = [round(x) for x in lista_enteros]
count = 0
for i in range(nt+1):
    u[1:N] = A.dot(u[1:N])
    if i == lista_enteros[count]:
        count = count + 1
        plt.plot(x,u)
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
print("Numero de pasos para llegar al segundo 1:", nt)
```

```
Numero de pasos para llegar al segundo 1: 799
```

[240]:



## 1.2 Método implícito

### 1.2.1 Con un $\Delta x = 0.1$

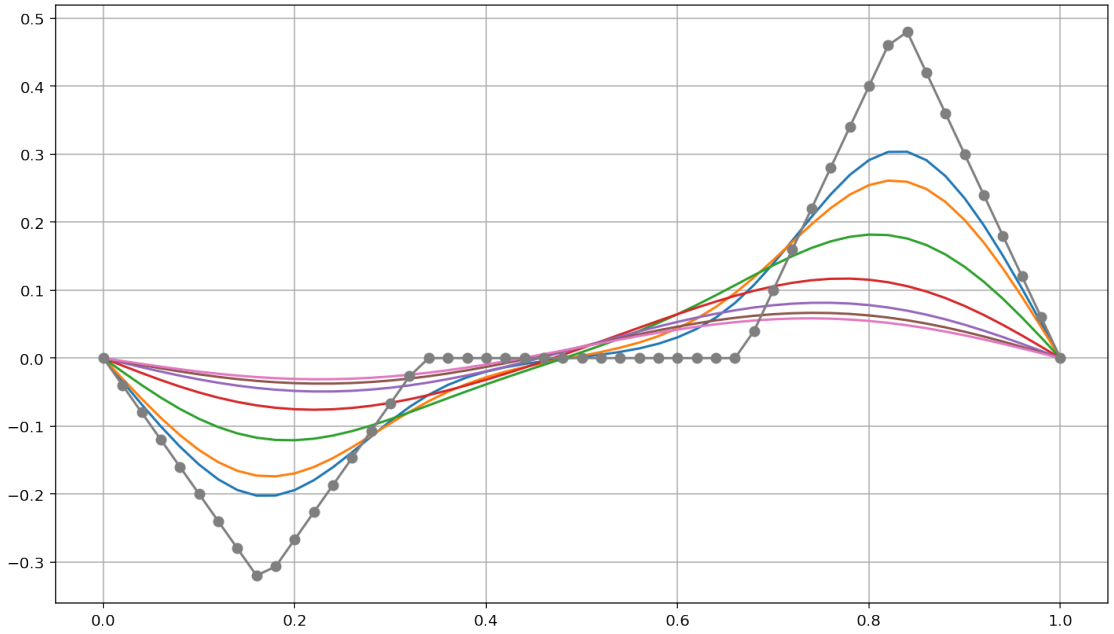
[241]:
```
N = 50
x = np.linspace(0,1,N+1)
```

[242]:
```
#Definimos el tamaño de los pasos en tiempo y espacio
deltax = 0.1
deltat = 0.05
```

[243]:
```
#Construimos la matriz A
deltax = 0.1
deltat = 0.05
mu = deltat/deltax**2
A = np.zeros([N-1,N-1])
np.fill_diagonal(A,1+2*mu)
for i in range(1,N-1):
    A[i,i-1] = -mu
    A[i-1,i] = -mu

u = np.vectorize(f)(x)

tiempofinal = 1
nt = int(tiempofinal/deltat)
listatiempos = np.array([0.05, 0.1, 0.25, 0.5, 0.75, 0.9, 1])
lista_enteros = (listatiempos/deltat)
lista_enteros = [round(x) for x in lista_enteros]
count = 0
for i in range(nt+1):
    u[1:N] = np.linalg.solve(A,u[1:N])
    if i == lista_enteros[count]:
        count = count + 1
        plt.plot(x,u)
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
print("Numero de pasos para llegar al segundo 1:", nt)
```

```
Numero de pasos para llegar al segundo 1: 20
```

[243]:




### 1.2.2 Con un $\Delta x = 0.05$

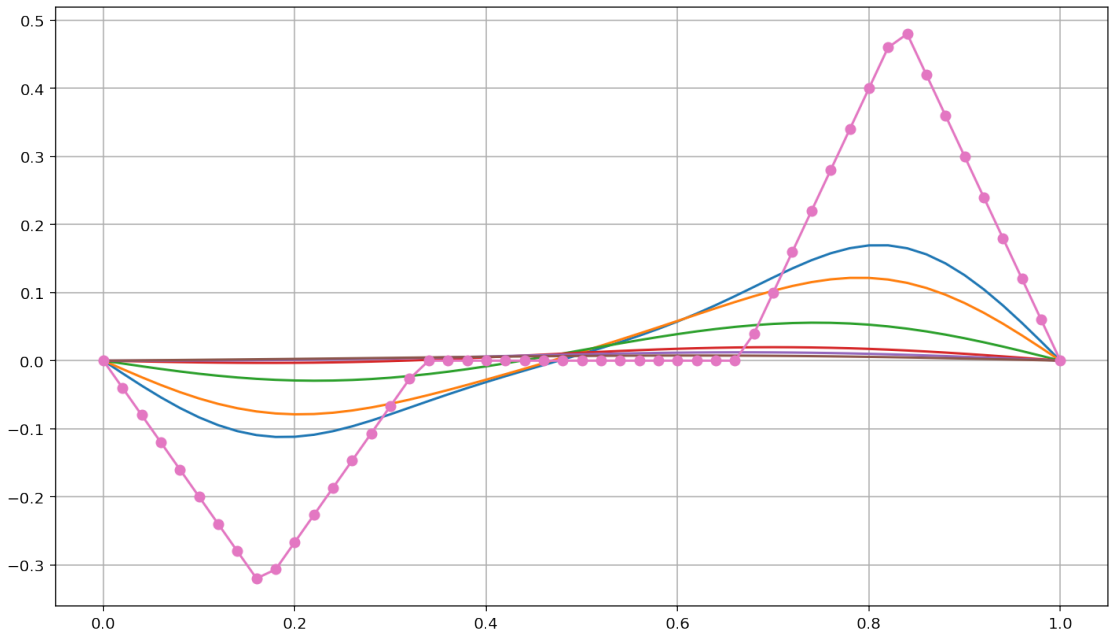
[244]:
```
#Definimos el tamaño de los pasos en tiempo y espacio
deltax = 0.05
deltat = 0.05
```

[245]:
```
#Construimos la matriz A
deltax = 0.05
deltat = 0.05
mu = deltat/deltax**2
A = np.zeros([N-1,N-1])
np.fill_diagonal(A,1+2*mu)
for i in range(1,N-1):
    A[i,i-1] = -mu
    A[i-1,i] = -mu

u = np.vectorize(f)(x)

tiempofinal = 1
nt = int(tiempofinal/deltat)
listatiempos = np.array([0.05, 0.1, 0.25, 0.5, 0.65, 0.85, 1])
lista_enteros = (listatiempos/deltat)
lista_enteros = [round(x) for x in lista_enteros]
count = 0
for i in range(nt):
```

```
    u[1:N] = np.linalg.solve(A,u[1:N])
    if i == lista_enteros[count]:
        count = count + 1
        plt.plot(x,u)
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
print("Numero de pasos para llegar al segundo 1:", nt)
```

```
Numero de pasos para llegar al segundo 1: 20
```

[245]:




## 1.3 Método Crank-Nicolson

### 1.3.1 Con un $\Delta x = 0.1$

[246]:
```
N = 50
x = np.linspace(0,1,N+1)
```

[247]:
```
#Definimos el tamaño de los pasos en tiempo y espacio
deltax = 0.1
deltat = 0.05
```

[248]:
```
deltax = 0.1
deltat = 0.05
mu = deltat/deltax**2
B_gorro = np.zeros([N-1,N-1])
```

```
np.fill_diagonal(B_gorro,1 + mu)
for i in range(1,N-1):
    B_gorro[i,i-1] = -0.5*mu
    B_gorro[i-1,i] = -0.5*mu
print(B_gorro)
```

```
[[ 6.  -2.5  0.  …  0.   0.   0. ]
 [-2.5  6.  -2.5 …  0.   0.   0. ]
 [ 0.  -2.5  6.  …  0.   0.   0. ]
 …
 [ 0.   0.   0.  …  6.  -2.5  0. ]
 [ 0.   0.   0.  … -2.5  6.  -2.5]
 [ 0.   0.   0.  …  0.  -2.5  6. ]]
```

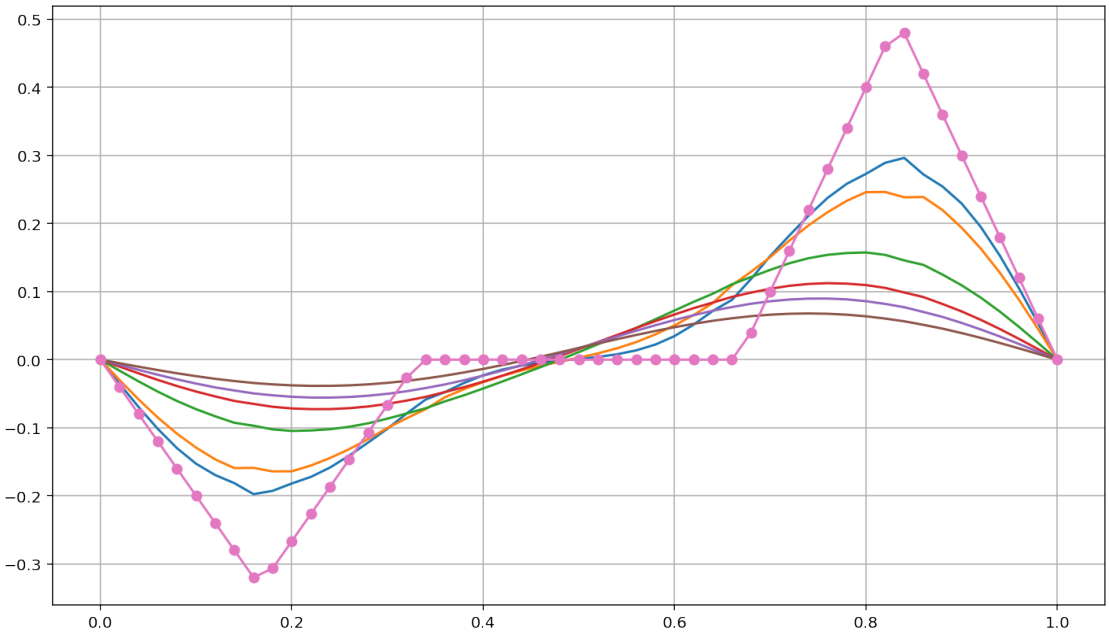
[249]:
```
B = np.zeros([N-1,N-1])
np.fill_diagonal(B,1 - mu)
for i in range(1,N-1):
    B[i,i-1] = 0.5*mu
    B[i-1,i] = 0.5*mu
print(B)
```

```
[[-4.   2.5  0.  …  0.   0.   0. ]
 [ 2.5 -4.   2.5 …  0.   0.   0. ]
 [ 0.   2.5 -4.  …  0.   0.   0. ]
 …
 [ 0.   0.   0.  … -4.   2.5  0. ]
 [ 0.   0.   0.  …  2.5 -4.   2.5]
 [ 0.   0.   0.  …  0.   2.5 -4. ]]
```

[250]:
```
deltax = 0.1
deltat = 0.05
u = np.vectorize(f)(x)

tiempofinal = 1
nt = int(tiempofinal/deltat)
listatiempos = np.array([0.05, 0.1, 0.3, 0.5, 0.65, 0.85, 1])
lista_enteros = (listatiempos/deltat)
lista_enteros = [round(x) for x in lista_enteros]
count = 0
for i in range(nt):
    u[1:N] = np.linalg.solve(B_gorro, B.dot(u[1:N]))
    if i == lista_enteros[count]:
        count = count + 1
        plt.plot(x,u)
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
print("Numero de pasos para llegar al segundo 1:", nt)
```

```
Numero de pasos para llegar al segundo 1: 20
```

[250]:




### 1.3.2 Con un $\Delta x = 0.05$

[251]:
```
#Definimos el tamaño de los pasos en tiempo y espacio
deltax = 0.05
deltat = 0.05
```

[252]:
```
deltax = 0.05
deltat = 0.05
mu = deltat/deltax**2
B_gorro = np.zeros([N-1,N-1])
np.fill_diagonal(B_gorro,1 + mu)
for i in range(1,N-1):
    B_gorro[i,i-1] = -0.5*mu
    B_gorro[i-1,i] = -0.5*mu
print(B_gorro)
```

```
[[ 21. -10.   0. …   0.   0.   0.]
 [-10.  21. -10. …   0.   0.   0.]
 [  0. -10.  21. …   0.   0.   0.]
 …
 [  0.   0.   0. …  21. -10.   0.]
 [  0.   0.   0. … -10.  21. -10.]
 [  0.   0.   0. …   0. -10.  21.]]
```

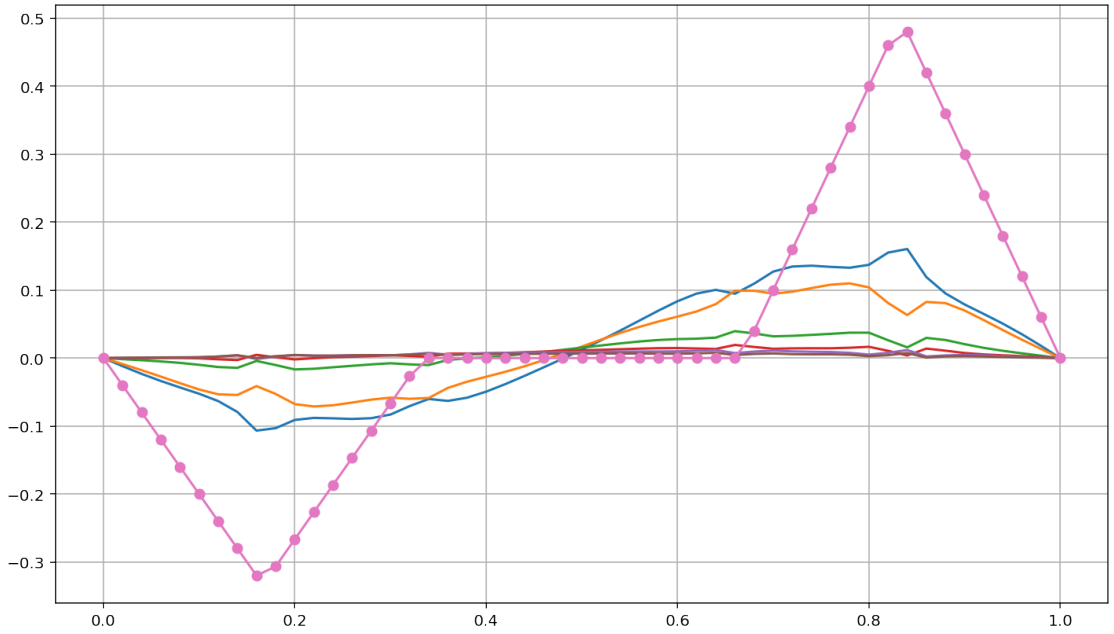
[253]:
```
B = np.zeros([N-1,N-1])
np.fill_diagonal(B,1 - mu)
for i in range(1,N-1):
    B[i,i-1] = 0.5*mu
    B[i-1,i] = 0.5*mu
print(B)
```

```
[[-19.  10.   0. …   0.   0.   0.]
 [ 10. -19.  10. …   0.   0.   0.]
 [  0.  10. -19. …   0.   0.   0.]
 …
 [  0.   0.   0. … -19.  10.   0.]
 [  0.   0.   0. …  10. -19.  10.]
 [  0.   0.   0. …   0.  10. -19.]]
```

[254]:
```
deltax = 0.05
deltat = 0.05
u = np.vectorize(f)(x)

tiempofinal = 1
nt = int(tiempofinal/deltat)
listatiempos = np.array([0.05, 0.1, 0.3, 0.5, 0.65, 0.85, 1])
lista_enteros = (listatiempos/deltat)
lista_enteros = [round(x) for x in lista_enteros]
count = 0
for i in range(nt):
    u[1:N] = np.linalg.solve(B_gorro, B.dot(u[1:N]))
    if i == lista_enteros[count]:
        count = count + 1
        plt.plot(x,u)
plt.plot(x,np.vectorize(f)(x),'o-')
plt.grid(True)
print("Numero de pasos para llegar al segundo 1:", nt)
```

```
Numero de pasos para llegar al segundo 1: 20
```

[254]:




[0]: