INFORMACIÓN BÁSICA					
ASIGNATURA:	**Física Computacional.**				
TÍTULO DE LA PRÁCTICA:	**Práctica de Ecuación diferencial de Laplace.**				
NÚMERO DE PRÁCTICA:	**05**	AÑO LECTIVO:	**2022-A**	NRO. SEMESTRE:	**VII**
FECHA DE PRESENTACIÓN:	27/06/2022	HORA DE PRESENTACIÓN:	**10:15pm**		
Integrante(s): Alván Ventura Edsel Yael			NOTA		
DOCENTE(s): Danny Giancarlo Apaza Veliz.					

Práctica 6
Física Computacional

Escrito por
Alván Ventura, Edsel Yael
ealvan@unsa.edu.pe

Profesor
Apaza Veliz, Danny Giancarlo
dapazav@unsa.edu.pe

27/06/2022

1 Problema 1

Implementar un código para la ecuación 15 de las diapositivas.

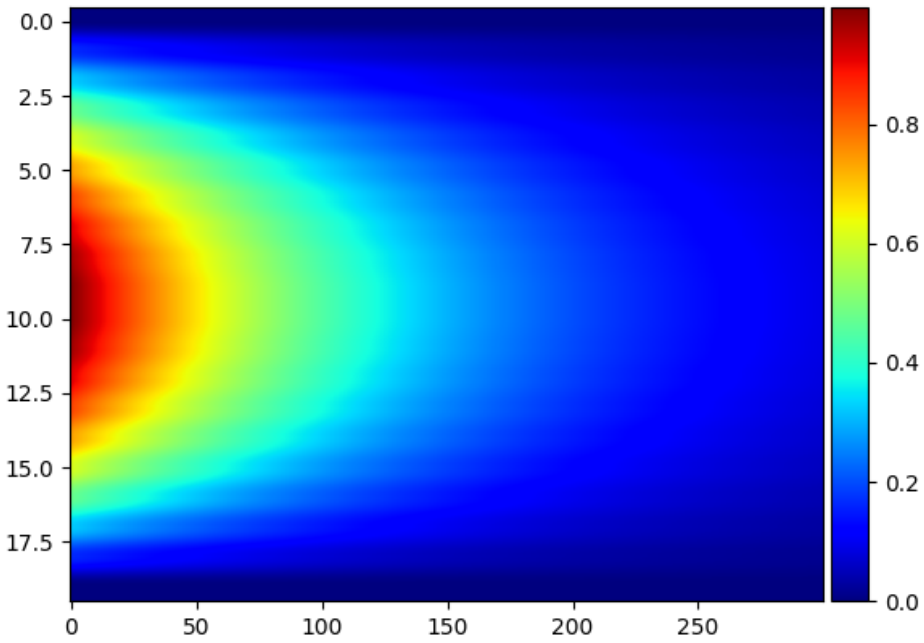
```
1 import matplotlib.pyplot as plt
2 import matplotlib
3 from mpl_toolkits.axes_grid1 import make_axes_locatable
4 import numpy as np
5 import math as m
6
7 V = 1.1#la constante V de la formula
```

```
8
9  def createMalla(m,n):
10     return np.zeros((m,n))
11 #funcion para obtener los colores para nuestra
12 #imagen(obtenida de la documentacion de python)
13 def cmap_map(function, cmap):
14     """ Applies function (which should operate on vectors
15     of shape 3: [r, g, b]), on colormap cmap.
16     This routine will break any discontinuous points in a colormap.
17     """
18     cdict = cmap._segmentdata
19     step_dict = {}
20     # Firt get the list of points where the segments start or end
21     for key in ('red', 'green', 'blue'):
22         step_dict[key] = list(map(lambda x: x[0], cdict[key]))
23     step_list = sum(step_dict.values(), [])
24     step_list = np.array(list(set(step_list)))
25     # Then compute the LUT, and apply the function to the LUT
26     reduced_cmap = lambda step : np.array(cmap(step)[0:3])
27     old_LUT = np.array(list(map(reduced_cmap, step_list)))
28     new_LUT = np.array(list(map(function, old_LUT)))
29     # Now try to make a minimal segment definition of the new LUT
30     cdict = {}
31     for i, key in enumerate(['red','green','blue']):
32         this_cdict = {}
33         for j, step in enumerate(step_list):
34             if step in step_dict[key]:
35                 this_cdict[step] = new_LUT[j, i]
36             elif new_LUT[j,i] != old_LUT[j, i]:
37                 this_cdict[step] = new_LUT[j, i]
38         colorvector = list(map(lambda x: x + (x[1], ), this_cdict.items()))
39         colorvector.sort()
40         cdict[key] = colorvector
41
42     return matplotlib.colors.LinearSegmentedColormap('colormap',cdict,1024)
43
44 #f: es una lambda function
45 #malla: es la matriz
46 #x: es el array con la condicion inicial
47 def fillMalla(malla,f,x):
48     # x = a:h:b es un array
49     x_f = np.array([f(item) for item in x])
50     malla[:,0] = x_f
51
52 def calculate(malla,h,k):
53     r = (V**2*k)/(h**2)#por el momento... k/h^2
54     # for _ in range(40):
55     for j in range(0,malla.shape[1]-1):
56         for i in range(1,malla.shape[0]-1):
57             malla[i,j+1] = r*malla[i-1,j] + (1-2*r)*(malla[i,j]) + malla[i+1,j]*r
```

```
58
59  def getCalorMap(arr):
60      dark_jet = cmap_map(lambda x: x*1, matplotlib.cm.jet)
61      ax = plt.subplot()
62      im = ax.imshow(arr,cmap=dark_jet,aspect='auto')
63      divider = make_axes_locatable(ax)
64      cax = divider.append_axes("right", size="5%", pad=0.05)
65      plt.colorbar(im,cax = cax)
66      plt.show()
67
68  def ecuacionCalor(a,b,t0,tf,f,mx,ny):
69      malla = createMalla(mx,ny)
70      h = (b-a)/(mx-1)
71      k = (tf-t0)/(ny-1)
72      x = np.linspace(a,b,malla.shape[0])
73      t = np.linspace(t0,tf, malla.shape[1])
74      fillMalla(malla,f,x)
75      calculate(malla,h,k)
76      getCalorMap(malla)
77      print(malla[:,0])
78      print(malla[:,malla.shape[1]-1])
79
80  def main():
81      #malla dimensions
82      #x:
83      mx = 20
84      #y:
85      ny = 300
86      #Para x axis
87      a = 0
88      b = 1
89      h = (b-a)/(mx-1)
90      #Para y axis
91      t0 = 0
92      tf = 0.2
93      k = (tf-t0)/(ny-1)
94      #Condicion inicial
95      n = 1
96      f = lambda x: m.sin(m.pi*x)#m.sin(n*m.pi*x) + m.sin(n+1)*m.pi*x#
97      #ecuacion de calor:
98      ecuacionCalor(a,b,t0,tf,f,mx,ny)
99
100 if __name__ == "__main__":
101     main()
```

1.1 Resultados

El resultado de la implementación anterior es:



2 Problema 2

Modifique el código anterior de manera que permitan resolver problemas con condiciones en la frontera de la forma $u(0,t) = g1(t) = 1$ y $u(a,t) = g2(t) = -1$.

2.1 Resolución

A continuación se muestra la implementación modificada para considerar las condiciones frontera:

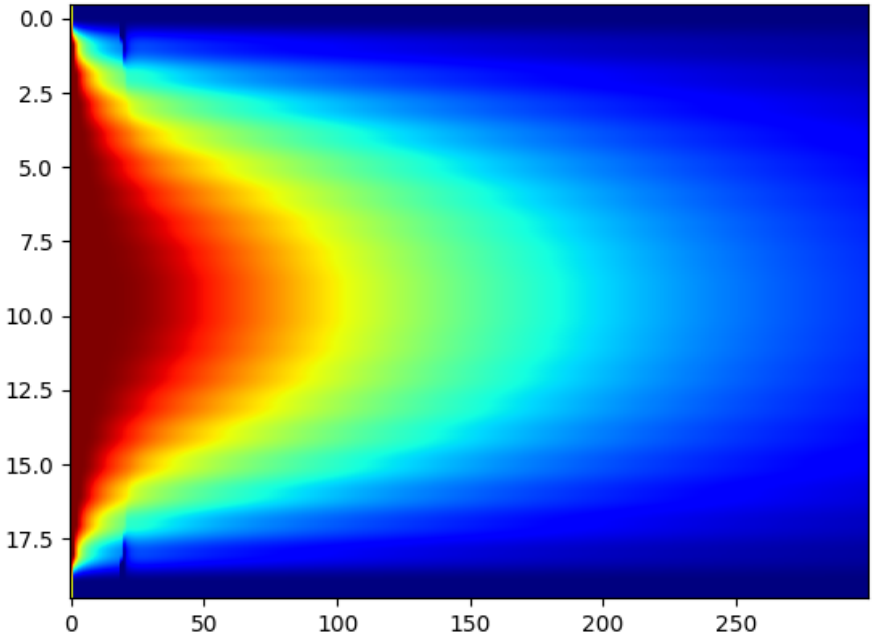
```
1  import matplotlib.pyplot as plt
2  import matplotlib
3  from mpl_toolkits.axes_grid1 import make_axes_locatable
4  import numpy as np
5  import math as m
6
7
8  def createMalla(m,n):
9      return np.zeros((m,n))
10
11 #funcion para obtener los colores para nuestra
12 #imagen(obtenida de la documentacion de python)
13 def cmap_map(function, cmap):
```

```
14      cdict = cmap._segmentdata
15      step_dict = {}
16      # Firt get the list of points where the segments start or end
17      for key in ('red', 'green', 'blue'):
18          step_dict[key] = list(map(lambda x: x[0], cdict[key]))
19      step_list = sum(step_dict.values(), [])
20      step_list = np.array(list(set(step_list)))
21      # Then compute the LUT, and apply the function to the LUT
22      reduced_cmap = lambda step : np.array(cmap(step)[0:3])
23      old_LUT = np.array(list(map(reduced_cmap, step_list)))
24      new_LUT = np.array(list(map(function, old_LUT)))
25      # Now try to make a minimal segment definition of the new LUT
26      cdict = {}
27      for i, key in enumerate(['red','green','blue']):
28          this_cdict = {}
29          for j, step in enumerate(step_list):
30              if step in step_dict[key]:
31                  this_cdict[step] = new_LUT[j, i]
32              elif new_LUT[j,i] != old_LUT[j, i]:
33                  this_cdict[step] = new_LUT[j, i]
34          colorvector = list(map(lambda x: x + (x[1], ), this_cdict.items()))
35          colorvector.sort()
36          cdict[key] = colorvector
37
38      return matplotlib.colors.LinearSegmentedColormap('colormap',cdict,1024)
39
40  #f: es una lambda function
41  #malla: es la matriz
42  #x: es el array con la condicion inicial
43  def fillMalla(malla,g1,g2):
44      # x = a:dx:b es un array
45      # x_f = np.array([f(item) for item in x])
46      # malla[:,0] = x_f
47      # print(malla)
48      malla[:,0] = g1
49      malla[:,malla.shape[0]-1] = g2
50
51  def calculate(malla,dx,dt):
52      s = (dt)/(dx**2)#por el momento... dt/dx^2
53      # for _ in range(40):
54      for j in range(0,malla.shape[1]-1):
55          for i in range(1,malla.shape[0]-1):
56              malla[i,j+1] = s*malla[i-1,j] + (1-2*s)*(malla[i,j]) + malla[i+1,j]*s
57
58  def getCalorMap(arr):
59      dark_jet = cmap_map(lambda x: x*1, matplotlib.cm.jet)
60      ax = plt.subplot()
61      im = ax.imshow(arr,cmap=dark_jet,aspect='auto')
62      # cs = ax.contourf(arr, levels=np.linspace(0, 1, 25))
63      # divider = make_axes_locatable(ax)
```

```
64      # cax = divider.append_axes("right", size="5%", pad=0.05)
65      im.set_clim(0,1)
66      # plt.colorbar(im, cax = cax)
67      plt.show()
68
69  def ecuacionCalor(a,b,t0,tf,f,mx,ny,g1,g2):
70      malla = createMalla(mx,ny)
71      dx = (b-a)/(mx-1)
72      dt = (tf-t0)/(ny-1)
73      x = np.linspace(a,b,malla.shape[0])
74      t = np.linspace(t0,tf, malla.shape[1])
75      fillMalla(malla,g1,g2)
76      calculate(malla,dx,dt)
77      getCalorMap(malla)
78      # print(malla[:,malla.shape[0]-1])
79
80
81  def main():
82      #malla dimensions
83      #x:
84      mx = 20
85      #y:
86      ny = 300
87      #for x axis
88      a = 0
89      b = 1
90      dx = (b-a)/(mx-1)
91      #for y axis
92      t0 = 0
93      tf = 0.2
94      dt = (tf-t0)/(ny-1)
95      #Condicion inicial
96      f = lambda x: m.sin(m.pi*x)
97      #Condiciones frontera:
98      g1 = 1
99      g2 = -1
100     #ecuacion de calor:
101     ecuacionCalor(a,b,t0,tf,f,mx,ny,g1,g2)
102
103 if __name__ == "__main__":
104     main()
```

3 Problema 3

Sea $f(x) = sen(n\pi x) + sen(n+1)\pi x$, busque valores de h y k para que tenga una solución estable y varié n para ver comportamiento diferentes en las condiciones iniciales.



4 Problema 4

Sea $f(x) = 4 - |4x - 1| - |4x - 3|$, busque valores de h y k para tener una solución estable.